LocKey: Location-based Key Extraction from the WiFi Environment in the User's Vicinity

Philipp Jakubeit$^{1[0000-0001-6216-6100]}$,
Andreas Peter$^{1,2[0000-0003-2929-5001]}$, and Maarten van Steen$^{1[0000-0002-5113-2746]}$

^{1} University of Twente, Drienerlolaan 5, 7522 NB Enschede, The Netherlands
^{2} University of Oldenburg, Ammerländer Heerstraße 114-118, 26129 Oldenburg, Germany

Abstract. We investigate extracting persistent information from semi-volatile signals in the user's vicinity to extend existing authentication factors. We use WiFi as a representative of semi-volatile signals, as WiFi signals and WiFi receiver hardware are ubiquitous. WiFi hardware is mostly bound to a physical location and WiFi signals are semi-volatile by nature. By comparing different locations, we confirm our expectation that location-specific information is present in the received WiFi signals. In this work, we study whether and how this information can be transformed to satisfy the following properties of a cryptographic key so that we can use it as an extension of an authentication factor: it must be uniformly random, contain sufficient entropy, and the information must be secret. We further discuss two primary use cases in the authentication domain: using extracted low-entropy information (48 bits) for password hardening and using extracted high-entropy information (128 bits and 256 bits) as a location-specific key. Using the WiFi-signal composition as an authentication component increases the usability, introduces the factor of 'location' to the authentication claims, and introduces another layer of defense against key or password extraction attacks. Next to these advantages, it has intrinsic limitations, such as the need for the receiver to be in proximity to the signal and the reliance on WiFi signals, which are outside the user's control. Despite these challenges, using signals in the proximity of a user works in situations with a fallback routine in place while increasing usability and transparency. LocKey is capable to extract low-entropy information at all locations measured, and high-entropy from 68% locations for 128-bit keys (48% of the locations respectively for 256-bit keys). We further show that with an initial measurement time of at most five minutes, we can reconstruct the key in at least 75% of the cases in less than 15, 30, and 40 seconds depending on the desired key strength.

Keywords: Location-based Authentication · Fuzzy Key Extractions · WiFi Signals.

1 Introduction

Authentication is a crucial component of ensuring the security of online transactions and information. It describes the 'provision of assurance that a claimed characteristic of an entity is correct' [16]. There are three main types of authentication factors distinguished by the claim made: knowledge factors, possession factors, and inherence factors. *Knowledge factors* involve something the user knows (e.g., password, PIN, or an answer to a security question). These are most prevalent in practice and are often used as the primary factor in multi-factor authentication. *Possession factors* involve something the user has (e.g., a smart card, a physical token, or a mobile phone). These factors are becoming more common in modern authentication methods, as users are increasingly relying on their mobile devices for authentication. *Inherence factors* involve something the user is, represented by biometric data (e.g., fingerprints, facial recognition, or iris scans). These factors are becoming more popular as a means to access a physical token or a mobile phone. Multi-factor authentication combines two or more of these factors to provide increased security. For example, a bank might use a combination of a password (knowledge factor) and an SMS (possession factor) to authenticate a user.

Location-based authentication is another type of claim that can be used during the authentication process. It is different from the other authentication factors as it is not about the user but the environment the user is in. Traditionally, a location-based authentication factor is used to localize a user (e.g., IP address ranges, GPS). However, we do not intend to localize a user but to recognize the environment the user is in. For the purpose of authentication, to validate a location claim, it is sufficient to validate that the claimed location is indeed a location associated with the user. Where the location is located is no required knowledge to validate the claim.

We propose the recognition of WiFi measurements of a location as an additional factor. In today's traditional setup, a user accesses a device either by knowledge or inherence claims to authenticate from the device towards a service with a combination of knowledge and possession claims. The user provides their password and a token or a challenge-response authentication based on a key the user owns. Our goal is to extend these factors with a claim of location in terms of WiFi measurements. Such a measurement must contain sufficient information for the desired use case. As each location differs in the information available, we choose the amount of information to extract on a per-location basis. With this, we are capable of either extending the information present in a password/key-based authentication claim or replacing an entire authentication factor. In the banking example given above, it might suffice to know that a user logging in with the correct credentials is at a typical location for this user. E.g., credit card usage consistency checks rely largely on such behavioral consistencies.

Especially in urban environments, wireless protocols based on the IEEE 802.11 standard (WiFi) and hence WiFi signals, and WiFi hardware are ubiquitous and ever-increasing. WiFi signals are known in the literature to be suited for various use cases such as indoor positioning [26], area selection [8], distance binding [10], behavioral profile construction [21], location fingerprinting [18], and key extraction [7]. In this paper, we look at the latter: extracting information from the WiFi signals surrounding us which share properties of a cryptographic key. What distinguishes our work from previous work is that we only rely on measurements without changing the existing infrastructure. By this, we introduce an extension that can be applied seamlessly (see Section 8 for more details on the differences with the related work). At first glance this might seem easy to achieve, however, it turned out to be more intricate. First, as our work is motivated by the quest for seamless authentication, we do not consider having control over or changing the behavior of access points (APs). We only consider the WiFi hardware of the user, the sensor. Second, as we observe electromagnetic signals, there are fluctuations and disturbances, which result in signals being inconsistently present. Third, as WiFi signals are emitted into the world we are required to find a way to make the derived information secret.

Our assumed WiFi infrastructure builds on stationary access points (APs), which constantly emit signals to indicate their presence, the so-called beacon frame. We use the information in the periodically sent beacon frames to derive location-specific information from the vicinity of a user. The setting is that we only observe signals, we need to account for inconsistencies in the volatile signals, and we need to make at least some information from the publicly available signals secret. The approach we take is (as done in biometrics e.g., [17]) not to store the information, but to generate and later reproduce it from a semi-persistent source. This is appealing as it reduces the attack surface because an adversary cannot extract this information from the hardware. Due to the potential volatility of the signals we require a backup procedure or fallback routine to be in place. To make the observed information secret, we require another source of randomness such that a vicinity key derived from the environment becomes uniformly random, and secret, and contains sufficient entropy in the information-theoretic sense. With LocKey:

- We show how to use a vicinity key to strengthen existing secret information (key or password).
- We increase the usability of multi-factor authentication.
- We get location as a claimed characteristic while preserving privacy.

To do so, we describe how to construct a WiFi measurement by observing only available APs by processing the available beacon frame features as an environmental entropy source and show how to precisely reconstruct such a WiFi measurement from a sufficiently similar measurement. Next, we show how to derive a vicinity key from such a WiFi measurement by introducing a device component. While doing so, we identify entropy estimates and evaluate the performance of our proposed method by using our real-world dataset and analyze our method in terms of the system's security. We observe that a low-entropy vicinity key of 48-bit can be extracted at all considered locations, while we can extract high-entropy vicinity keys of 128-bit at 68% of the locations, and 256-bit vicinity keys at 48% of the locations considered. We further show that with initial WiFi measurements of up to five minutes, we can reconstruct a key in less than 40 seconds for all vicinity key strengths. Further, LocKey adds an extra layer of security on top of knowledge and possession factors. When a password or key gets compromised in traditional schemes, the adversary broke the system, while with LocKey in place, an adversary is required to derive a composition of WiFi APs that is sufficiently similar to the AP composition at the user's location.

2 Foundations

In this section, we describe the two underlying foundations of LocKey. First, we focus on WiFi beacon frames, what they entail, and by which circumstances they are impacted. Second, we focus on fuzzy extractors, what they are, and why they are a perfect fit for the inconsistent AP compositions we observe.

2.1 WiFi beacon frames

The WiFi beacon frame is a management frame defined in the IEEE 802.11 standard [14]. A periodically sent beacon frame advertises the presence of the base station. The WiFi beacon frame entails information about the network, like the physical address and capabilities of the network. Which fields to use is limited by two aspects; the presence in the beacon frame itself and the receiver's operating system (OS). *The presence of the information* in the beacon frame itself is not guaranteed, as a frame contains mandatory and optional fields. *The OS of the receiver* matters, as different OSs provide different levels of access to beacon frame fields in general and based on access rights within the system itself. In the case of the Linux OS, the accessible fields are the network's name called the service set identifier (SSID), the media access control address (MAC address), a general flag, the maximum bandwidth to use, the security and capability flags, the frequency used, and the mode of the AP [12]. These fields combined have a theoretical maximum of 63 bits; entropy analyses on real-world data suggest a minimum of 9 bits [18]. Privileged access on Linux (root space) and Windows allows access to more beacon-frame fields, while OSX and mobile operating systems are more restrictive in accessing beacon-frame fields.

2.2 Fuzzy extractors

The authors of [9] coined the term fuzzy extractor. However, the idea of using sets to lock a vault goes back to [20]. One way to look at fuzzy extractors is as error-tolerant and nonuniformity-tolerant key-encapsulation mechanisms for a secret key. They can generate a uniformly random string R from an input w. This extraction process is error-tolerant, so a sufficiently similar input $w' \sim w$ reproduces the same uniformly random string R. The inputs' similarity can be expressed on the bit level by the inputs' Hamming distance. Sufficiently similar means that the Hamming distance is not greater than t, the number of errors that can be corrected. The generate and reproduce functions are the two building blocks of a fuzzy extractor and can be constructed from two components: a secure sketch and a strong extractor, as shown in Figure 1.

Secure Sketches A secure sketch is a function that recreates an input w from another input w' with a small Hamming distance to w. A secure sketch consists of two main components. The secure sketch and the reconstruct function. The secure sketch receives an input w and produces a sketch s such that a similar input w' together with that sketch s can be used by the reconstruct function to output the original input w.

Strong extractor The strong extractor function is not fuzzy in itself. It generates a defined output based on its defined inputs. A family of hash functions is used to extract a uniformly random string from an input and an entropy source. The extraction process uses the input w and an additional entropy source r. The authors of [9] show that a 2-wise independent hash function produces an optimal result, as the length of the random input r is less critical in the scenario of o fuzzy extractor. We will describe what family of hash functions we chose in our instantiation section.

Building a fuzzy extractor from a secure sketch and a strong extractor In Figure 1, we show a schematic of the components from a fuzzy extractor and how to construct it from a secure sketch and a strong extractor. The input w is used as input to the secure sketch to create a sketch s. This sketch, together with internally generated randomness r, forms the output, helper data $P = (s,r)$. This randomness r is used together with the input w as input to the strong extractor to create the uniformly distributed string R. To reproduce this R created by the generator function, the reproduce function receives the

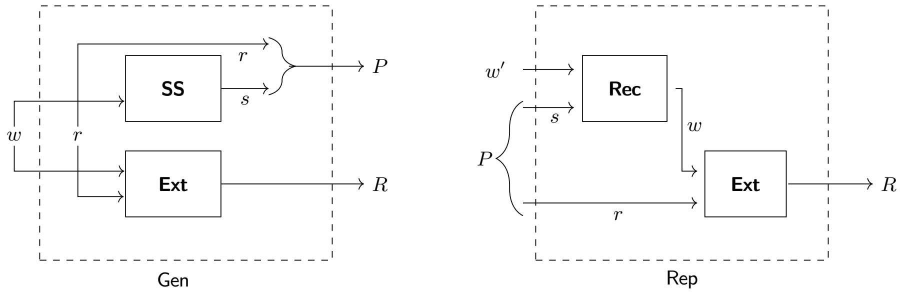


Fig. 1: Schematic of a fuzzy extractor constructed of a secure sketch (SS) and a strong extractor (Ext). The generate function takes w as input. The secure sketch creates a sketch, while the strong extractor uses the input and internally generated randomness to generate a uniformly random string R. The generator function outputs the string R and helper data P, which consists of the sketch s and internal randomness r. The reproduce function takes an input w' similar to w and the helper data P as input. Internally, a reconstruct function takes the input w' and the s element of the helper data P and outputs the reconstructed w. The reconstructed w is used together with the r part of the helper data P as input to the strong extractor, which outputs the same uniform random string R as outputted by the generator function if the Hamming distance of w and w' is sufficiently small ($\leq t$).

helper data P and a similar input w' as inputs. It inputs the sketch part s included in P and the input w' into the reconstruct function. The reconstruct function reconstructs from both its inputs the original input w. The reconstructed w is used with the random r component of the helper data P as input to the strong extractor. This strong extractor performs exactly as its counterpart in the generation construction and outputs R.

3 Overview on LocKey

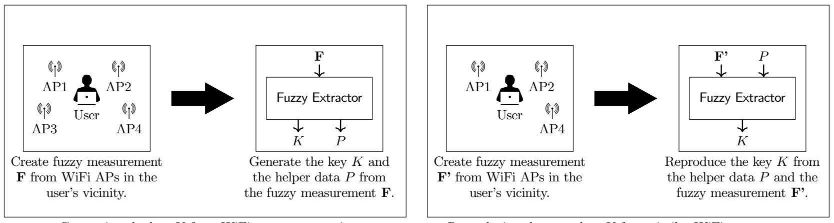
In this work, we focus on the IEEE 802.11 standard [14], which we will refer to as WiFi. However, we assume that the principle we describe will also hold for other wireless standards and somewhat persistent electromagnetic signals in general. We consider a *sensor*, a wireless receiver, capable of observing the beacon frames sent by WiFi APs. We assume both, the sensor and the APs, to be spatially static. We assume standard laptop hardware (e.g., [15]) as the sensor. The sensor gathers measurements of APs over time. We describe each AP by an AP representation (APR). We call a specific composition (set) of APRs measured at one location a WiFi measurement.

We want to see how the composition of measured WiFi signals in the vicinity of a user can be used to derive a vicinity key. The main challenge lies in the fact that the WiFi composition is fluctuating. Already a small physical area is subject to heavy fluctuations (an illustration of the fluctuation of WiFi signals in a right square prism of dimensions $36cm^2$ by 18cm can be seen at [22]). To compensate for the signal fluctuations, we choose to work with a fuzzy extractor. This works by conducting initial measurements and deriving a vicinity key and helper data. With this helper data, it is possible to derive the same vicinity key from sufficiently similar measurements. Sufficiently similar refers to two measurements differing in at most t elements. We derive t from the initial measurements as it influences the entropy of our derived key and we need to know before how much entropy is available at a location.

We focus especially on usability and transparency. An authentication system using the WiFi environment of a user does not impose a serious burden on the user. During the first generation of a vicinity key, the user is required to stay in place for the time required to measure the environment. However, afterward, the authentication can be conducted from the service and software on the user's device alone. This becomes even more prominent in the before-mentioned scenario of a confirmation SMS. Instead of retyping a code, the user is not required to engage at all. It suffices that the user resides at a known location. This leads to further applications such as transparent authentication schemes in which the service authenticates a user continuously.

We split a vicinity key into two components, the measurement component, and the device component. The helper data P forms the device component and the input set forms the measurement component. A fuzzy extractor is constructed such that P leaks only a predefined amount of information about the input set. The input set leaks no information about the helper data. Only in combination do they produce the desired, uniformly random string, the vicinity key. Contrary to the default assumption of fuzzy extractors, we propose for our application to treat both components as confidential.

To derive a key once and again, we distinguish the generation and reproduction phases. In Figure 2, we give a visual overview of the extraction process of a vicinity key in both phases.


AP1
AP2
User
AP3
AP4
Create fuzzy measurement **F** from WiFi APs in the user's vicinity.
F
Fuzzy Extractor
K
P
Generate the key K and the helper data P from the fuzzy measurement **F**.
AP1
AP2
User
AP4
Create fuzzy measurement **F'** from WiFi APs in the user's vicinity.
F'
P
Fuzzy Extractor
K
Reproduce the key K from the helper data P and the fuzzy measurement **F'**.

Generating the key K from WiFi measurements in the user's vicinity.

Reproducing the same key K from similar WiFi measurements in the user's vicinity and the helper data.

Fig. 2: A visualized overview of LocKey.

Generation phase During the generation phase, we scan the environment for a fuzzy measurement $\mathbf{F}$, which we input into the generate function of a fuzzy extractor. Internally, this results in inputting $\mathbf{F}$ into a secure sketch and inputting $\mathbf{F}$ together with freshly generated randomness r into a strong extractor. The secure sketch produces the sketch s, which we combine with r to the helper data P=(s,r). The strong extractor produces the vicinity key K. Due to being based on the fuzzy measurement $\mathbf{F}$, the vicinity key K is location specific. The fuzzy measurement $\mathbf{F}$ forms the measurement component and r, internally created randomness, forms the device component.

Reproduction phase During the reproduction phase, we scan the environment for a fuzzy measurement $\mathbf{F}'$. We input $\mathbf{F}'$ and $P=(s,r)$ into the reproduce function of a fuzzy extractor. Internally, this results in inputting $\mathbf{F}'$ and the sketch part s of the helper data P into a reconstruction function which outputs $\mathbf{F}$. This $\mathbf{F}$ forms, together with the r part of the helper data, the input for a strong extractor, which produces the vicinity key K. The vicinity key K is identical to the vicinity key K produced in the generation phase if $\mathbf{F}$ and $\mathbf{F}'$ differ in at most t APs.

The challenging aspect of deriving a key from WiFi measurements is the size of the universe and the fact that we have an unstable amount of APs in each WiFi measurement. Thus, we require for LocKey a solution that can deal with varying input sizes and perform well even in large universes of potential input sizes. However, before we find a fitting solution, we first discuss our goals for LocKey and how it can be utilized.

3.1 Application and integration suggestions

We intend to use the vicinity key K derived from the WiFi environment of the user as a claim in an authentication scheme. We discuss the use of LocKey in password hardening and challenge-response-based authentication and present further use cases in Section 7. In general a vicinity key can be seen as secure salt with a location component.

Password hardening The user extends the entropy in a password by appending the extracted bits from the derived vicinity key. For password hardening, we require only a low amount of entropy; for example, 48 bits may suffice. This application increases the entropy of a password almost for free. The only requirement we impose on a user is to reside in a specific location.

Challenge-Response The party the user is authenticating towards presents a question (challenge) and the user must provide a valid answer (response). For a challenge-response scheme, we require a higher amount of entropy; 128-bit or 256-bit are standard key sizes. Which key size to choose depends on the context of the authentication and the available entropy at a measured location. A vicinity key K could be used to prove that the environment at a specific location was measured. The two choices of challenge-response authentication are to either use a message authentication code-based approach [23] or an approach that uses asymmetric cryptography and requires the user to sign a message [11].

4 Instantiating LocKey

In this section, we elaborate on the information present in the data and our choices for the fuzzy extractor and its components, the secure sketch, and the strong extractor.

4.1 Data Representation

We build an access-point representation (APR) from the features provided by the AP. To do so, we need to estimate the information conveyed in each feature. The theoretical maximum amount of MAC address information is 48 bits [14], constructed from a 24-bit organizationally unique identifier (OUI) and a 24-bit network interface controller. As MAC addresses are assumed to be unique, we can interpret them as an index of APs. Assuming at least 10-bit for the OUI, the maximum of information has a cap of 2^{34} considering the APs min-entropy. The *min-entropy* is the negative logarithm of the probability of the most likely outcome. We choose the min-entropy, as we need to consider the most conservative entropy estimate for the strong extractor to guarantee the chosen key strength. The definition of min-entropy is:

$$H_{min} = -\log(\max_i p_i)$$

To represent the entropy of an APR in a real measurement, we considered the data observed. We measured only a fraction of potential MAC addresses with a min-entropy of 7.15 bits. We choose one byte as APR length, as we use the extendable output function SHAKE (Secure Hash Algorithm with KEccak [5]) to generate the APR by digesting the plain beacon frame features. We use SHAKE-128 with a security strength of 128 bits when it is sufficient. Also, we show the results for higher security, in which case we opt for SHAKE-256.

4.2 Fuzzy Extractor

We construct our fuzzy extractor as described in Section 2.2 by combining a secure sketch with a strong extractor. For the secure sketch, we choose PinSketch [9], and for the strong extractor, we choose the original universal hash function proposed by Carter and Wegman [6].

Secure Sketch We choose PinSketch as the secure sketch for our instantiation, as it fits our requirements perfectly. PinSketch conducts error correction on a set of elements. It can handle arbitrarily many errors, even though each error it can correct reduces the entropy. Therefore, we choose t, the number of errors to correct, tight on a per-location basis. However, PinSketch is optimal with a loss of $t\, log(n+1)$. What distinguishes PinSketch from other set-based sketches is that it can handle varying set sizes and can deal with large universes due to its optimized nature.

Assuming a universe $\mathcal{U}$, PinSketch creates a binary vector of $n = |\mathcal{U}|$ bits to represent a set. From there, it becomes equivalent to the error correction of two vectors in that space. The problematic part is the size of the universe $\mathcal{U}$. If it becomes too large, other secure sketches become infeasible. PinSketch uses two tricks to change that. PinSketch stores only the support of a vector and builds on BCH codes for error correction.

Support of a vector The support of a vector v, $supp(v)$, lists only the positions on which a vector is nonzero. Using only the support of a vector makes the description of small words very efficient. In the case of fuzzy WiFi measurements, each APR consists of 8 bits. Therefore, we can describe each APR by a vector of size $|APR| = 2^8$. To list all possible APRs and set only the APRs present in a specific WiFi measurement to 1 while all other values are 0 requires a vector of size $2^{|APR|}$. Listing only the positions present is obviously much more efficient.

BCH codes BCH codes [13] are a class of cyclic error-correcting codes constructed using polynomials over a finite field. BCH codes can correct multiple-bit errors and provide highly efficient decoding using syndrome decoding. To describe syndrome decoding, we need to dive a bit deeper into the inner workings of linear codes. We assume linear code C and a message x. The length of each code word in C is n, and the dimension of C as a vector subspace is k. To derive a code word $c = Gx$ of the linear code C corresponding to message x, we multiply x with a generator matrix G of size $k \times n$

whose rows form a basis for a linear code. G can be written in the standard form with I_k being the $k\times k$ identity matrix and P being a $k\times(n-k)$ matrix as $G=[I_k|P]$. From G the parity-check matrix H can be constructed as $H=[P^\top|I_{n-k}]$ such that $GH^\top=0$.

A *syndrome* s of any vector y of the ambient vector space is defined to be $s=Hy^\top$. Due to the construction of H, the syndrome s is zero if and only if the vector y is a code word.

Syndrome decoding makes use of the fact that $Hz=He$, the syndrome of an error pattern e is equal to the syndrome of an observation $z=c+e$ consisting of the code word c observed with this error pattern e, as we know from the definition of a syndrome that $Hc=0$.

The equality implies that a table of error patterns can be pre-computed. In the case of an observation of a value y, we know that y is no code word if $Hy^\top\neq 0$. Further, we can look up the error pattern in the generated table to retrieve the bits to correct. Deducing and correcting an error can be implemented in logarithmic time complexity.

Both choices make PinSketch quite efficient with time complexity of $poly(|w|\log n)$ [9] and also provide an optimal storage complexity of $t\,log(n+1)$. In words, this means that in a setting capable of correcting up to t errors, for a set w which consists of several $l=log(n+1)$ bit vectors, a sketch consists of only t binary vectors of length l. $t\,log(n+1)$ is also the amount of entropy loss, which makes sense as the sketch makes $t\times l$ bits public. The authors of PinSketch focused on the time complexity required. They reduced it to polynomial time based on the set size of the input set $|w|$ and being only logarithmically in the size of the universe n. Most other secure sketch solutions working on sets are time bound by the size of the universe, therefore, being dependent only on the input-set size is an improvement. In the following, we list the workings of PinSketch in the generation and reproduction phase.

PinSketch [9]

Generation phase:
Input: w, **Output:** s
Compute the syndrome $syn(w)$ of w.
Output the syndrome as sketch $s=syn(w)$.

Reproduction phase:
Input: w',s, **Output:** w
Compute the syndrome $syn(w')$ of w'.
Compute $\sigma=syn(w)-syn(w')$, the difference between both syndromes.
Find a vector v such that $syn(v)=\sigma$ for which holds that $|supp(v)|\leq t$.
As the vector v is equal to the difference of the inputs $v=w-w'$, output $v+w'=w$.

Strong extractor As a strong extractor, we choose the 2-wise independent family of universal hash function proposed by Carter and Wegman [6], which is defined by:

$$h_{a,b}(x)=((ax+b) \mod p) \mod m$$

We construct x from hashing all elements, the APRs, of an observed set w, a WiFi measurement $\mathbf{F}$, with SHAKE-128 (respectively SHAKE-256) and create an output of size $|x| \in \{128,256\}$, based on approved key sizes by the NIST [4]. The output size determines the desired security level. However, the number of APs available determines an upper bound for the security level per location.

As we intend to extract sufficient entropy, we choose our universe for hashing U_h depending on the security guarantees desired such that $|U_h| = |x|$.

It applies to the moduli that p must be a prime and $p \geq |U_h|$ and that m specifies the output range of numbers of the strong extractor. As p needs to be larger or equal to the size of the universe, we decide to go for a prime expressible by $2^z - k$ for tuples of $(z,k) \in \{(129,25),(257,93)\}$ such that p is greater than U_h, and we have a reasonably tight bound for the coefficients. We randomly choose the coefficients a and b modulo p except for $a \neq 0$. This choice implies that we have two z bit numbers as part of the helper data and hence as the device component of the key. We choose $m = |U_h|$ as we use the strong extractor to get only a uniformly distributed string based on the entropy we derived from our input x.

5 Evaluation

Data sets available online provide MAC addresses and other information. However, they do not include the capability features of WiFi beacon frames, which we need for measuring the environment. Therefore, we created our own, ethics-committee approved, data set from 37 locations measured in offices and flats[3]. We conducted the measurements with ordinary WiFi hardware from laptops (e.g., [15]) in the Linux OS. In total, we measured 1167 different APs in these locations, with the number of APs measured by the sensor varying from 1 to 100 APs in one measurement for the duration of one second. A beacon frame is received every 5 milliseconds to 1 second. This interval is configurable by the user and device-specific. Therefore, not all available APs are present in each measurement.

The main questions are whether there is sufficient entropy at a location, and for how long we need to measure to derive the desired amount of entropy. Both questions must be asked during the generation phase and during the reproduction phase. During the generation phase, we determine the locations suitable for a specific key strength. With the measured duration, we set the stage for the reproduction phase. We discuss the inherent tradeoff and parameter choices later in this section. During the reproduction phase, we determine the measurements at a location suitable for reconstruction and the duration required to reconstruct successfully.

Generation phase – location suitability. First, we determined what sufficient entropy entails. Recall that we have a set of measurements per location. Each measurement is represented by a set of APs. We represent each AP by an 8-bit APR and each AP contains at least 7.15 bits of entropy we can harvest. The key size as well as the error correction require entropy. The *key size* determines the entropy required to derive a key of the desired size. Hence, we require at least $\lceil 48/7.15 \rceil = 7$ APs to derive the low-entropy key and 18 (respective 35) APs to derive the high-entropy keys of 128 and 256 bits. The *error correction* requires additional entropy. We described in Section 4.2 that PinSketch has the optimal amount of entropy loss, $t\log(n+1)$ for n being the size of the universe $|\mathcal{U}| = 2^{|APR|}$. As we determined the length of an APR to be 8 bits, the entropy loss is $t*8$ bits. As each AP has a min-entropy of 7.15 bits we require 1.12 additional APs for each AP to correct.

[3] https://gitlab.com/lockey1/scandata

We considered 37 locations with each a total length of one hour divided into 3600 one-second measurements. We chose the first measurement as starting point for the generation phase. This represents the real-world scenario: a user enters a location and checks whether it is suited for the use of LocKey by starting to measure WiFi signals. With this procedure, we determined the number of locations providing sufficient entropy per desired key strength (see Table 1).

No. of suitable locations (out of 37)	Key strength in bits
37	48
25	128
17	256

Table 1: The number of locations from our test set of 37 locations from which we can extract sufficient entropy to guarantee the desired key strength with at least one AP to correct using the first measurement as starting point.

Generation phase – duration. We analyzed our dataset and chose generation-phase measurement lengths of 3, 4, and 5 minutes for the 48, 128, and 256-bit key extractions. We chose these measurement lengths as they provide sufficient entropy for a maximum of locations with sufficient entropy for error correction. However, at the end of this section, we discuss the implications and other choices.

Reproduction phase The helper data generated during the generation phase entails the number of APs and the number of APs we can correct. Therefore, we aggregated measurements until we have sufficient APs to attempt a reconstruction of the generation-phase WiFi measurements. This condition is met as soon as we observe at least the number of APs we are capable to correct subtracted from the total number of APs used in the generation phase. Our choices for measurement times during the generation phase imply that we have $3600-180=3420$ seconds of measurements for the 48-bit scenario, 3360 seconds for the 128-bit scenario, and 3300 seconds for the 256-bit scenario remaining to analyze the reconstruction capabilities. We chose every measurement of these remaining measurements as starting point and aggregate measurements until the condition is met to attempt a reconstruction.

Reproduction phase – measurement suitability. In Figure 3 we show the success rate of the attempted reconstructions. Note that missing bars indicate that the specific location has not sufficient entropy for the desired key strengths. We divided the bar plot into three focus areas. We reconstructed the vast majority of locations in more than 99%. For the 48-bit scenario, only locations L11 and L12 had a slightly lower performance than 99% but still more than 98%. For the 128-bit scenario, less locations provide sufficient entropy at all, but only locations L11 and L22 had fewer than 99% of starting points from which we can reconstruct. In the 256-bit scenario, even fewer locations had sufficient entropy, but also most locations had more than 99% of successful reconstructions. Only location L30 had about 98% of correct reconstruction. Locations L11 and L32 are outliers and only allowed for about 75% and 85% of the starting points to reconstruct successfully. These reconstruction performances show that LocKey performs with more than 99% of successful reconstruction for the vast majority of locations and starting measurements.

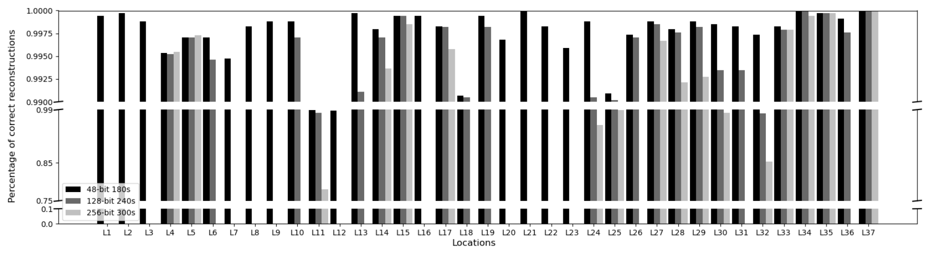
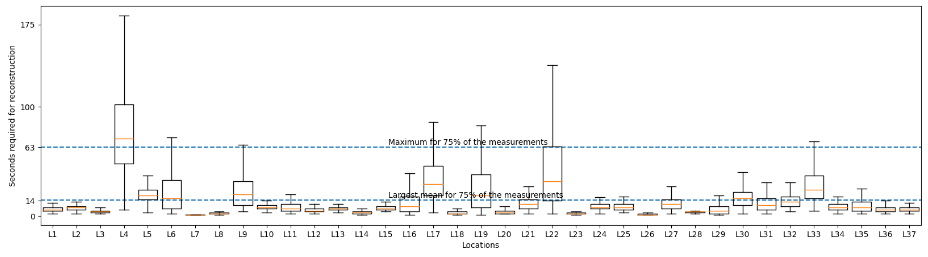


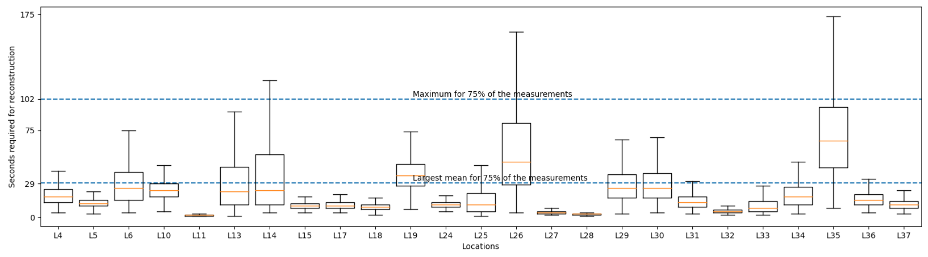
Fig. 3: These bar plots show the reconstruction performance per location for each possible starting measurement in the remaining set of valid measurements for the reproduction phase. Missing bars show that the specific location does not provide sufficient entropy for the location in question. The broken axis allows us to focus on the relevant performances.

Reproduction phase – duration. The crucial follow-up question is how long it takes to conduct these reconstructions. In Figure 4, we show box plots to convey the varying length of measurements required. For clarity, we omit the outliers in our visualization. In the 48-bit scenario we observed between 0.7% and 7.1% of outliers at all locations except L7 and L9 with the highest outlier requiring 262 seconds at location L4. In the 128-bit scenario we observed between 0.02% and 9.3% of outliers at all locations, with the highest outlier requiring 253 seconds at location L35. In the 256-bit scenario we observed between 0.3% and 6.4% of outliers at all locations except L27 with the highest outlier requiring 750 seconds at location L37. Per figure, we show the largest average and maximum time required to restore two-thirds of all measurements. We can reconstruct a set of APs on average for 75% of the measurements in less than 15, 30, and 40 seconds and require a maximum of 65, 103, and 170 seconds for the respective key sizes of 48, 128, and 256 bits. We also observed that we require less than three minutes in the 48 and 128 bits scenarios and that for most of the locations five minutes are sufficient for the 256 bits scenario, while two locations require up to 15 minutes to reconstruct. This is a direct consequence of the 5 minutes we used in the generation phase. During these 5 minutes, many APs can be aggregated, which we need to measure before attempting the reconstruction.

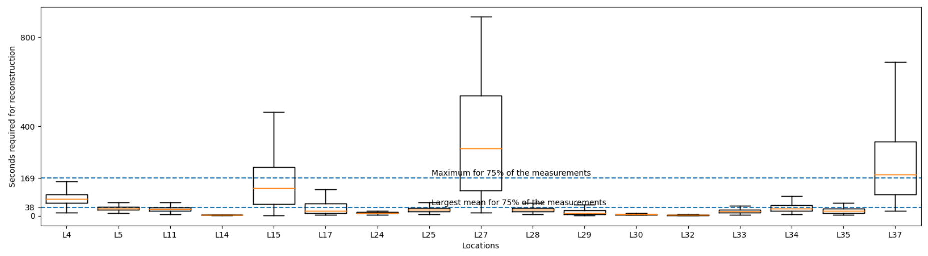
Tradeoff. Having longer generation-phase measurements introduces a tradeoff for the reproduction phase. More generation-phase measurements lead to a higher percentage of successful reconstructions, however, require longer measurement times during the reproduction phase for several locations. Less generation-phase measurements imply a shorter measuring duration required during the reproduction phase. This reduced duration implies that the reconstruct is less often successful. From our analysis, we observed that as a bare minimum, we required only generation-phase measurement lengths of 40, 60, and 160 seconds (for 48, 128, 256-bit key strength scenarios). Choosing these shorter generation-phase measurements, we required less time to gather enough APs during the reconstruction (the outliers in the 256-bit scenario take at most 200



(a) 48-bit scenario.



(b) 128-bit scenario.



(c) 256-bit scenario.

Fig. 4: The three box plots show the required time for reconstructing the set of APs in all scenario. We highlighted the average and maximum time required to reconstruct 75% of the measurements of all locations.

seconds), which makes sense as we considered fewer measurements. Therefore, we are not required to measure for that long when trying to reconstruct. The disadvantage is that fewer locations provide sufficient entropy. Additionally, the performance drops significantly for some locations as the APs not taken into consideration are still present and could poison our reconstruction attempts.

We determined how long to measure during the generation phase. This impacts the performance and measurement times of the reconstruction phase. We suggest deciding on a per-location basis for an optimal strategy. Looking at the four locations with a mean above 20 seconds for the 48-bit case in Figure 4a we observed strong signal fluctuations at the location. Therefore, we must also decide on a per-location basis how stable the signals are and whether the signal stability is sufficient for the desired use case.

6 Security model and analysis

We consider a user and an adversary. The user conducts WiFi measurements and executes either the generate or reproduce function to construct the vicinity key K. We assume that the user is trusted, that the user's hardware is not compromised, and that the user does not deliberately provide any information to the adversary.

We consider two different scenarios, the first in which the attacker is assumed to have no knowledge of the device component, the helper data P and the second in which the attacker has knowledge of P. We assume for both scenarios that the adversary's goal is to retrieve the vicinity key K.

No access to P Without access to the device component, the adversary is required to guess the correct key K or the WiFi measurements $\mathbf{F}$ the user used during the generation phase and the random coefficients of the strong extractor. Due to our choice of key derivation, the key K is either of size 48-bit, 128-bit, or 256-bit and is uniformly distributed. This implies that an attacker has a chance of $\frac{1}{|K|}$ for $|K| \in \{2^{48}, 2^{128}, 2^{256}\}$ to guess a key correctly.

An adversary could compromise a WiFi capable device near the user, or be on-site to conduct WiFi measurements. However, without knowing the information in the helper data P the adversary has no means to know whether the WiFi measurement conducted is the WiFi measurement used during the generation phase. Without knowing the random coefficients a and b from the universal hash function, it is infeasible to extract the correct vicinity key K. Therefore, we conclude that an adversary is required to brute force at least the number of bits of the freshly generated vicinity key and that doing so is infeasible.

With access to P This entails that the adversary got access to the user's device and circumvented security mechanisms in place. Traditional standalone solutions (e.g., password or key) would be broken at this point. Therefore, this is a worst-case analysis, which focuses on the question: if the adversary has access to the device component, the helper data P, how likely is it for an adversary to also derive the measurement component? From P the adversary knows a, b, t and $|\mathbf{F}|$. As PinSketch is capable of correcting up to t APs the task at hand for an adversary becomes to sample the correct set of APs such that together with P the vicinity key K can be reconstructed. The minimum set size to have for a correct reconstruction has size $k = |\mathbf{F}| - t$. The chance for an adversary to choose the correct k APRs is $\binom{n}{k}$. The question becomes what the knowledge of the AP composition of the adversary is. We express the adversary's domain knowledge by n. If the adversary has no information, then $n = 2^{48}$, which is the theoretical upper bound from the specification. An adversary with access to a service like Wigle, which lists one billion APs, reduces this number already to $n = 2^{30}$. However, if an adversary is capable of constructing a scan on-site the size of n could be decreased significantly. If the adversary misses only one AP, having observed only $k-1$ correct APs, the missing AP can be from the whole domain (at least 2^{30}). However, if the adversary happens to measure at least k correct APs the chance becomes smaller. As soon as the adversary scans at least one AP not in the measurement component of size k, the adversary has to try at least k combinations. Therefore, this can be mitigated by asking the service to allow only a strict number of authentication attempts. The only chance the adversary has is to scan k correct APs as $\binom{n}{k} = 1$ for $n = k$. Considering the fluctuation of APs, this is highly improbable and works only if the adversary has access to P and the location's WiFi composition.

7 Discussion

In this section, we discuss our results and its limitation as well as the privacy considerations that come with it.

Results We deliberately chose the bare minimum of information per APR and showed that all locations provide sufficient entropy in the measurable beacon frames for the scenario of password hardening and that between 46% and 68% percent of the locations provide sufficient entropy to derive a high-entropy key. Further, we showed that we are capable to determine whether LocKey can be applied at a given location and which measurement times we require per specific location.

Limitations The limitations of our approach are twofold. On the one hand, we observe outliers (Section 5), which require long (up to 15 minutes) reconstruction times. These locations with specific key strengths might not yet be suited for the application of LocKey. On the other hand, we consider WiFi measurements, which are inherently mostly out of the control of the user. Therefore, changes in the AP composition could deny key reconstruction. However, we observe two general and one specific mitigation. First, being capable of correcting t errors gives us some flexibility regarding a changing environment. Second, we assume application only when a fallback mechanism is present. In the case of the password and the challenge-response mechanism, a second factor would enable an alternative confirmation of identity and allow for a re-enrollment with the new WiFi environment.

Local Attacks Jamming, flooding or AP pool poisoning describes an adversary who deliberately changes the AP environment of a user. This can happen during the generation phase and during the reconstruction phase. Therefore, we assume a trusted generation phase. The system checks anyways that sufficient APs are present. During the reconstruction phase manipulating the WiFi environment could result in a denial of service. As a countermeasure, we have the assumption that LocKey should only be used in conjunction with a fall-back factor.

Privacy The privacy considerations are relevant as personally identifiable information (PII) is present in the beacon frame. For the features accessible on the Linux OS, the SSID and the MAC address classify as PII. The EU classifies a MAC address belonging to a user, even in its hashed form, as PII [24]. However, it is impossible for an adversary to retrieve the PII because we process the PII only locally and combine it with high entropy randomness. An adversary on-site could conduct a WiFi measurement and obtain the PII, such as every entity conducting WiFi measurements at a specific location. For an adversary, even holding the correct vicinity key K, there is no stable ground truth to derive relevant PII.

8 Related work

In our work, we use WiFi beacon frame features measured for a certain period to derive a vicinity key by fuzzy extraction. To our knowledge, we are the first to promote the idea of using the semi-volatile signals of a user as key storage from a read-only perspective. We do require that a backup mechanism is present, due to the volatile nature of the signals.

The authors of [1] approach the problem from the perspective of spatial role-based access control and the authors of [7] discuss position-based cryptography also in terms of a user's claimed characteristic. Both are examples of plenty solutions which require control of the sensor and the APs. With these, the authors prove that in the Bounded Retrieval Model position-based secret communication and position-based authentication and signatures are possible. In contrast, we offer the user an increase in security and the reduced burden of a second authentication factor only by access to the sensor. However, we most likely require occasional re-enrollments of the user.

The authors of [3] use error correction to create a shared key separately on two WiFi nodes, focusing on reducing the computation and communication overhead. They achieve this by elevating signal interference. Explicitly, the authors use deep fades, a strong and destructive interference. They measure deep fades occurring in the signal transmitted to the node for a particular duration and encode their occurrence in a bit string. If two nodes are sufficiently close to one another, the bit string is similar enough that both nodes can reconstruct the same key by communicating verification information. The approach presented differs from ours, as two nodes are required to compute the same key, and they create the key from signal interference. Instead, we want to reproduce a key at the same device and location later from information sent in a beacon frame. However, we consider the work sufficiently relevant as might offer a path for future work to look into the information transmitted by the WiFi standard and in interference or error behavior.

The authors of [25] also use WiFi signals to create a shared key between two devices. In their case, two devices are in a body area network. What makes their publication interesting is that it focuses on the RSSI only. They create a fixed-length binary vector between two WiFi devices based on the signal strength of the communication between these two devices. The authors of [2] go one step further and use next to the RSSI, the link quality indicator. Another metric produced by the sensor not by the AP. They show that the method is suited to 'verify the location of an IoT node within a small area with high probability of success'. Using other sensor metrics might help to extend LocKey in the future or use a similar concept on small IoT devices.

Unrelated to fuzzy extractors is the concept of [18] in which the authors fingerprint a location only in terms of WiFi signals. They draw a line between capability and PII aspects of a WiFi beacon frame and recognize a location in terms of only the previously mentioned six capability features available in the Linux OSs. Their work shows that the surrounding WiFi signals accurately define a location. However, they focus on the average entropy by applying the Shannon entropy as they are not required to harvest the entropy for key generation. Hence, they could leave out the higher entropy sources of MAC address and SSID to opt for more privacy, as they intend to share a template of the WiFi location with a service to allow for authentication which requires a heightened focus on PII.

9 Conclusion

The signals in a user's environment provide sufficient reconstructable information to derive a key even when having access only to the sensor and focusing only on particular WiFi features. The density of the electromagnetic signals determines the amount of information that is extractable at a location. We showed that it is possible to extract the information once and also to some certainty *reconstruct* the extracted information. However, it is possible that the required composition is not achievable. Therefore, we focused only on use cases with a fallback mechanism in place. We further showed that combining the measured component with the device component in a secure extraction step generates uniformly random key material. The extent to which we can successfully extract a key is dependent on the location. We were able to extract a low-entropy key of 48-bit for all locations we considered. Extracting a 128-bit key succeeded for nearly 68% of the locations while extracting a 256-bit key succeeded for nearly 46% of the locations we considered.

Future work includes applying LocKey in practice, to increase the entropy available and to decrease the measurement time. More WiFi data could be captured to increase the entropy available and decrease the measurement time. This can either be achieved by including more features like the RSSI or by deriving different stable features from the beacon frame. Additionally, different data sources can be used to extract the required entropy. Different aspects of WiFi could be gathered, e.g., interference or error behavior as done by [3] or WiFi could be used to read other signals in the user's vicinity as done by [19]. Next to WiFi, different wireless data sources like Bluetooth, Lora, or alike can be aggregated. Finally, we expect a 'natural' increase in electromagnetic signals available over time due to the adaptation of more wireless devices, which increases the applicability of authentication-factor extensions like LocKey.

Bibliography

[1] Isaac Agudo, Ruben Rios, and Javier Lopez. A privacy-aware continuous authentication scheme for proximity-based access control. In *39th Journal of Computers & Security*, pages 117–126, 2013.

[2] Muhammad Naveed Aman, Mohamed Haroon Basheer, and Biplab Sikdar. Two-factor authentication for IoT with location information. In *2nd IEEE Internet of Things Journal*, volume 6, pages 3335–3351, 2018.

[3] Babak Azimi-Sadjadi, Aggelos Kiayias, Alejandra Mercado, and Bulent Yener. Robust key generation from signal envelopes in wireless networks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 401–410, 2007.

[4] Elaine Barker and Quynh Dang. Nist special publication 800-57 part 1, revision 5: Recommendation for key management: Part 1–general, may 2020. *Cited on*, page 58, 2020.

[5] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The keccak sponge function family: Specifications summary. *Ref: http://keccak. noekeon. org/specs_summary. html*, 2011.

[6] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, 1977.

[7] Nishanth Chandran, Vipul Goyal, Ryan Moriarty, and Rafail Ostrovsky. Position based cryptography. In *CRYPTO*, volume 9, pages 391–407. Springer, 2009.

[8] YounSun Cho, Lichun Bao, and Michael T. Goodrich. Laac: A location-aware access control protocol. In *2006 3rd Annual International Conference on Mobile and Ubiquitous Systems-Workshops*, pages 1–7, 2006.

[9] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *International conference on the theory and applications of cryptographic techniques*, pages 523–540. Springer, 2004.

[10] Aymen Fakhreddine, Nils Ole Tippenhauer, and Domenico Giustiniano. Design and large-scale evaluation of wifi proximity metrics. In *European Wireless 2018; 24th European Wireless Conference*, pages 1–6. VDE, 2018.

[11] James Foti. Entity authentication using public key cryptography. 1997.

[12] GNOME. org.freedesktop.networkmanager.accesspoint, 2021. `https://developer.gnome.org/NetworkManager/1.2/gdbus-org.freedesktop.NetworkManager.AccessPoint.html`.

[13] Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffers*, 2:147–156, 1959.

[14] IEEE Standard. Wireless lan medium access control (mac)and physical layer (phy) specifications, 2007. `https://www.iith.ac.in/~tbr/teaching/docs/802.11-2007.pdf`.

[15] Intel. Dual band wireless-ac 8265, 2021. `https://ark.intel.com/content/www/us/en/ark/products/94150/intel-dual-band-wireless-ac-8265.html`.

[16] ISO 27000. Information technology, security techniques, information security management systems, overview andvocabulary, 2018.

[17] A Jagadeesan, T Thillaikkarasi, and K Duraiswamy. Cryptographic key generation from multiple biometric modalities: Fusing minutiae with iris feature. *International Journal of Computer Applications*, 2(6):16–26, 2010.

[18] Philipp Jakubeit, Andreas Peter, and Maarten van Steen. The measurable environment as nonintrusive au-thentication factor on the example of wifi beacon frames. *International Workshop on Emerging Technologies for Authorization and Authentication*, 5, 2022.

[19] Woojae Jeong, Jinhwan Jung, Yuanda Wang, Shuai Wang, Seokwon Yang, Qiben Yan, Yung Yi, and Song Min Kim. Sdr receiver using commodity wifi via physical-layer signal reconstruction. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[20] Ari Juels and Madhu Sudan. A fuzzy vault scheme. *Designs, Codes and Cryptography*, 38(2):237–257, 2006.

[21] Guoqiang Li and Patrick Bours. Studying wifi and accelerometer data based authentication method on mobile phones. In *Proceedings of the 2018 2nd international conference on biometric engineering and applications*, pages 18–23, 2018.

[22] Charles Lohr. A WebGL-based raytraced voxel engine with transparency of wifi signal over a 360mm x 360mm x 180mm area., 2016. `https://cnlohr.github.io/voxeltastic/`.

[23] James M Turner. The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, 198(1):1–13, 2008.

[24] WP29. Opinion 01/2017 on the proposed regulation for the eprivacy regulation (2002/58/ec), December 2017. `http://ec.europa.eu/newsroom/document.cfm?doc_id=44103`.

[25] Yang Wu, Yongmei Sun, Lei Zhan, and Yuefeng Ji. Low mismatch key agreement based on wavelet-transform trend and fuzzy vault in body area network. *International Journal of Distributed Sensor Networks*, 9(6):912873, 2013.

[26] Chouchang Yang and Huai-Rong Shao. Wifi-based indoor positioning. *IEEE Communications Magazine*, 53(3):150–157, 2015.