Wireless Networks (2024) 30:2081–2098
https://doi.org/10.1007/s11276-023-03645-6

ORIGINAL PAPER

Quest: instant questionnaire collection from handshake messages using WLAN

Xiaojun Zhu[1] · Hao Han[1]

Accepted: 22 December 2023 / Published online: 25 January 2024

Abstract
A common approach for questionnaire collection is to set up a WLAN and respondents submit answers via specific apps or a web browser. However, much unnecessary background traffic is incurred once a device connects to Wi-Fi, consuming limited airtime and constraining the number of simultaneous connections. In this paper, we show that connection is not necessary, since the handshake messages themselves can carry answers. We propose Quest, an alternative questionnaire collection system that requires neither Internet access nor a stable connection. Quest sets up a password-protected Wi-Fi network to which respondents connect using any smart device with a Wi-Fi module and type their answers as passwords. Quest then retrieves the answers from the handshake messages. However, the answers are sent in hashed form instead of plain text; thus, recovery of answers from handshake messages, i.e., password cracking, is time-consuming, leading to a long delay. Quest employs three techniques to address the challenge. First, Quest restricts the types of questions to closed-ended ones, reducing the set of possible answers. Second, Quest precomputes an offline dictionary to speed up the cracking process. Third, we prove that, in our problem, it is sufficient to consider the first 128 bits of a key, instead of the entire 384 bits in the standard. Thus, Quest only checks the first 128 bits of a key. These techniques reduce the worst-case per-user cracking time from several minutes to a few seconds on a laptop computer. We implement Quest in commodity-off-the-shelf hardware and evaluate it in a real-world environment.

Keywords Questionnaire collection service · WPA2 · Wi-Fi handshake · password cracking

1 Introduction

Onsite questionnaire can be an effective means of soliciting instant feedback, intentions, preferences, and opinions from the audience more quickly than other methods. For example, during lectures, the instructor may want to know the background of the students and whether they learn the course materials at the desired pace. This situation also occurs in a public lecture or an academic conference where the speaker needs feedback from the audience to adjust the pace or content. A common practice of the onsite questionnaire is to let the audience raise their hands, such as *"Hands up if you know..."*. However, this approach is inefficient in collecting multiple answers.

Paper-based questionnaires are another common method. The major disadvantage of this approach is that it is labor-intensive and incurs a certain delay in processing the collected answers. For example, the instructor may find, after the lecture, that many students have already lost their minds at the beginning of the lecture. The web-based questionnaire avoids the downside of the paper-based solution by building a website and could allow audiences to enter their responses in the browser. The downside is that, a web server in the cloud requires Internet access and it is a challenge itself to provide reliable Internet service to all participants in a packed area. Furthermore, real-world measurements [1] show that the Wi-Fi connection time of a smartphone could be up to 30 seconds. A local server can be used, e.g., one can set up a WLAN that provides questionnaire pages (detailed in Sect. 8.1). The problem is

✉ Xiaojun Zhu
xzhu@nuaa.edu.cn

Hao Han
hhan@nuaa.edu.cn

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, 29 Jiangjun Road, Nanjing 211106, Jiangsu, China

that the number of clients supported by (inexpensive) wireless routers is rather limited. Our experiments suggest that the number is roughly 10-15 for a reliable connection. To support more users, a mesh network consisting of multiple routers is required. However, this is not a portable solution, which also requires additional financial investments.

The reason why an AP can only support a limited number of clients depends on many factors, including the number of radios, the Wi-Fi chipset, the CPU speed, and the memory size. For every connected client, the AP needs to manage the client status. For example, when WPA or WPA2 is enabled, the AP has to keep track of individual encryption keys per client. A major factor we find during experiments is that many clients send background traffic periodically once the client is connected to Wi-Fi. Since all clients share the PHY wireless channel, allowing too many concurrent clients will inevitably lead to reduced bandwidth for each client. Thus, most AP implementations have a hard limitation on the number of clients.

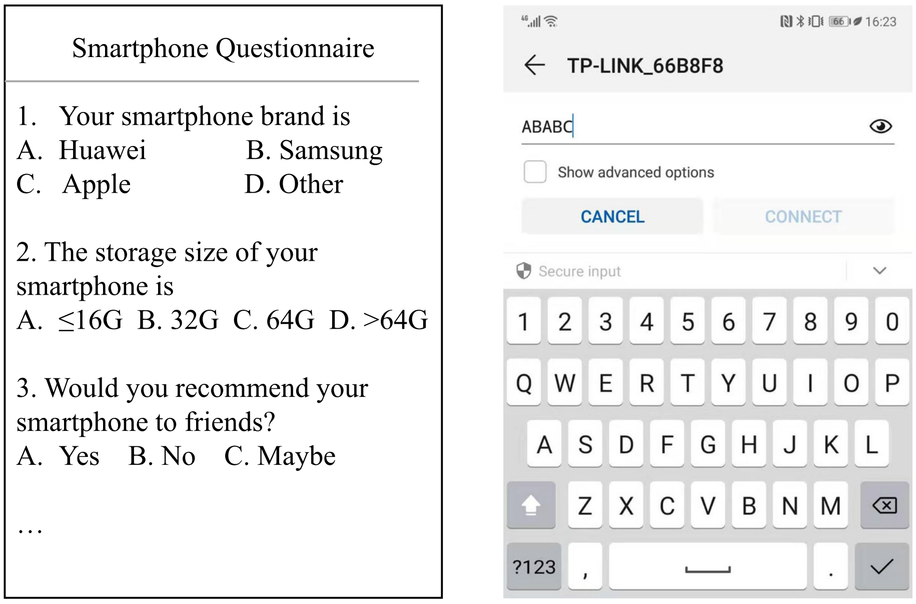
In this paper, we propose an alternative approach, Quest, a questionnaire collection system using WLAN. Quest configures a password-protected WLAN without disclosing the password. When joining the network, the respondents type their responses to the questionnaire in the password box, as in Fig. 1. Since the answers are not designed to match the password, the connection request is automatically rejected. However, the typed password is hidden in the handshake process and can be recovered as answers. Since no clients will be successfully associated with the AP, the AP does not maintain any connection with the clients. In addition, there is no overhead from TCP/IP or the upper layers in the connection process. Therefore, Quest can support more clients than the web-based approach.

Quest has several advantages over other approaches, as follows: (1) It avoids the burden of providing Internet access to crowded participants and can support a large number of clients. (2) Quest does not require any software or modifications on the client side, which is suitable for onsite questionnaires with strangers. For example, in a public lecture or an academic conference, the audience may be different from time to time. It is not convenient to require the audience to install software before joining the lecture or conference. (3) Quest is easy to deploy and can support a large number of clients. It only needs a laptop or smartphone with a built-in wireless card.

However, it is not trivial to design and implement a working Quest system in practice. The typed password is sent in hashed form instead of plain text; thus, recovery of answers from handshake messages, i.e., password cracking, is time-consuming, leading to a long delay. In our experiments, a brute-force cracking process takes several minutes for cracking the submitted answer of a single user on a questionnaire consisting of eight questions. Therefore, the common practice can take hours for 20 users, which is not suitable for instant questionnaire collection.

Our major contribution can be enumerated as follows.

1. We propose the design of Quest, which, to the best of our knowledge, is the first system using the Wi-Fi password interface to collect answers to questionnaires. It is able to support a large number of clients.
2. We propose several techniques to speed up the password-cracking process, reducing the time of recovering questionnaire responses from minutes to seconds. It should be noticed that our techniques can be



Smartphone Questionnaire

1. Your smartphone brand is
A. Huawei B. Samsung
C. Apple D. Other

2. The storage size of your smartphone is
A. ≤16G B. 32G C. 64G D. >64G

3. Would you recommend your smartphone to friends?
A. Yes B. No C. Maybe

…

TP-LINK_66B8F8

ABABC

Fig. 1 In Quest, a user can enter the answer to a questionnaire as the Wi-Fi password. Though the connection request will be rejected, the AP can infer the typed response from the handshake frames

used to crack Wi-Fi passwords in general, but given a strong password the state-of-the-art Wi-Fi security is still unbreakable even with our techniques.

3. Quest has already been used in field tests during the class in our university. Extensive real-world experiments demonstrate its efficacy and effectiveness.

2 Related work

2.1 Research on questionnaire design

How to design a questionnaire is a traditional research topic in social science [2]. In addition to printed questionnaires, the web-based questionnaire or the online survey also attracts much research attention [3, 4]. Various types of question are designed and analyzed. Quest targets at one type of question, that is, closed-ended questions whose answers are fixed to a limited number of options. The research in questionnaire design is orthogonal to this paper.

2.2 Wi-Fi handshake processes

The handshake process incurs extra delay when a user switches APs, which attracts much research attention [5]. Instead of optimizing the handshake process, we use it to transmit information. A similar idea has been proposed in the literature to use beacon messages to deliver information. Observing that a client cannot communicate with un-associated APs, Beacon-Stuffing [6] reuses several fields in beacon messages to transmit information between a client and APs without requiring the client to associate to any AP. However, Beacon-Stuffing requires driver modification at both the client and the AP. In dense WLANs with many APs, periodic beacon transmissions waste a lot of airtime, reducing the air time for data transmission. BeconRider [7] can coordinate APs so that data transmission occurs during the beacon period of neighboring APs. Note that recently, Wi-Fi has been used in the context of sensing and localization [8].

2.3 Alternative short range wireless communication

Besides Wi-Fi, alternative technologies can be applied to collecting data from nearby devices, including acoustic communication, Bluetooth, and visible light communication. Acoustic communication uses a speaker to send out signals and a microphone to receive signals, which are available on most smartphones. Thus, it is attractive for the development of novel applications without Internet access [9]. However, acoustic communication requires specially designed software on both the client and the server side, which is not convenient for clients. Bluetooth is mainly used to provide point-to-point connection between two devices. Up to 8 devices can form a piconet. In case of questionnaire collection where more than 7 clients are present, multiple piconets may form a multi-hop network [10]. Using Bluetooth to collect questionnaires also requires specialized software for clients. Recently, researchers have proposed methods to support more NB-IoT devices with coordinated APs [11].

In visible light communication [12], LED light bulbs are used to transmit information to clients, mainly used for broadcasting. LED bulbs can be replaced by a screen to form a screen-camera communication link. For example, RainBar+ [13] facilitates screen-camera communication by means of a speaker-microphone link to provide feedback to the sender, and ChromaCode [14] hides information in video. These works generally provide either broadcast or unicast service. It is challenging to apply them to a convergecast service, e.g., collecting information from respondents in the questionnaire collection problem.

2.4 Password cracking techniques and Wi-Fi security

There are generally three techniques for breaking passwords [15], brute-force attack, dictionary attack, and Rainbow table attack. Brute-force attack tries all possible combinations, dictionary attack facilitates the search process with a dictionary, and Rainbow table attack uses a Rainbow table to store precomputed hashes to reduce cracking time. These techniques are for breaking a general-purpose password and are not specific to the Wi-Fi password. Countermeasures such as using longer passwords and salted hashing are proposed to defend against these attacks. In particular, the hash function used in WPA2 is particularly designed to defend against Rainbow table attack.

Wi-Fi security has also attracted much attention. Wired Equivalent Privacy (WEP) has been shown to have serious security flaws and was deprecated [16]. However, it is still supported by most current devices. Wi-Fi Protected Access (WPA 2) relies on a 4-way handshake process and is vulnerable to a key reinstallation attack [17]. For stealing passwords from WPA 2 protected Wi-Fi, the most common attack is the de-authentication attack [18], where the adversary sends a spoofed deauthentication frame to force a legitimate user to disconnect and re-connect, exchanging handshake frames that are sniffed by the adversary. The adversary then employs brute-force search to find the password.

In this work, we are not interested in designing general-purpose password-cracking techniques or revealing the vulnerabilities of related systems and protocols [19]. Instead, we jointly constrain the set of possible passwords and propose specific password-cracking techniques. We discuss the potential security issues of our system in Sect. 8.2.

3 Quest design

We first show the workflow of Quest, and then explain how to constrain question types and how to choose Wi-Fi versions.

3.1 System overview and challenges

As shown in Fig. 2, Quest involves two parties, a questionnaire designer who distributes questionnaires and collects responses, and many respondents who submit answers. The designer should have a computer connected (via cable) to a wireless AP, and each respondent should have a wireless device (e.g., a smartphone) capable of connecting to Wi-Fi. Quest involves the following steps.

1. Before meeting the respondents, the designer prepares the questionnaire (Sect. 3.2) and builds an offline dictionary containing all possible answers (Sects. 4 and 5).
2. The designer meets the respondents and shows the questions to the respondents. This can be done via slides, whiteboard, or printed lecture notes. The method for questionnaire distribution is orthogonal to the paper.
3. The designer sets up a WLAN through the AP, listening for connections from respondents.
4. Each respondent figures out the answer, and uses the answer as the WiFi password to connect to the WLAN.
5. The designer sniffs all handshake frames and cracks out the respondents' answers (Sects. 4 and 5).

The major challenge of Quest is the long cracking time in step 5, which may be years if arbitrary answers are allowed in step 1. We need to carefully design questions to reduce the number of candidate responses. Nevertheless, we find in our experiments that constraining question choices is insufficient, and we should devise fast cracking algorithms.

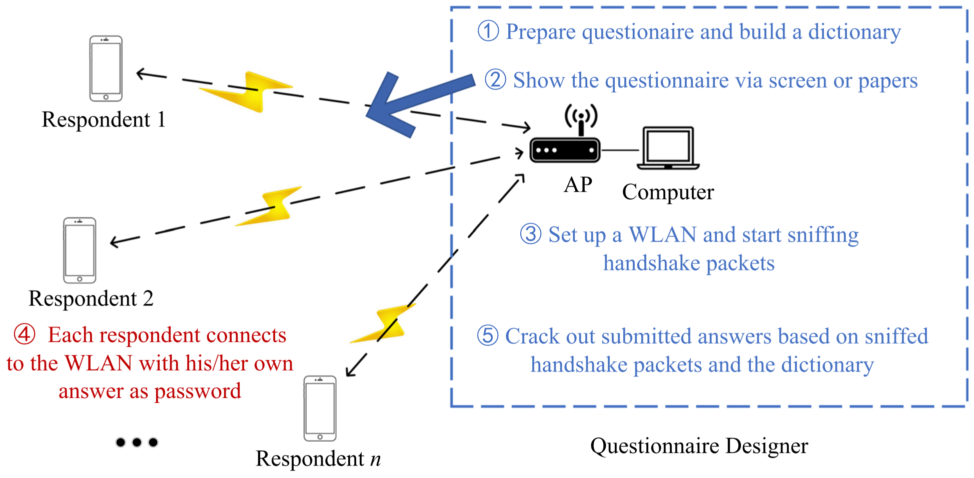
3.2 Constraining question types

Quest can support the following questions, which are referred to as closed-ended questions: (1) True or false question, where the answer is either 'T' for true or 'F' for false. (2) Single-choice questions, where the answer can be represented by a letter such as A, B, C, or D. (3) Multiple choice questions with a fixed number of options. For example, a respondent is required to select exactly two options. Then the question is similar to two single-choice questions. (4) Multiple choice questions with a variable number of options. In this case, a delimiter is used to separate answers. For example, a user may need to type '0ABC0' to indicate that the answer is 'ABC'.

Note that other *objective* questions, where possible answers can be easily enumerated (e.g., student IDs) can also be used in Quest. In contrast, it is not recommended to use Quest for open-ended questions or free-text questions, where the answer to a question is a general sentence, due to a long cracking time.

There are a few remarks on the constraints. Our main motivation is to reduce the cracking time, but from another point of view, we do not consider *subjective* questions either. Open-ended questions require respondents to write their opinion, and answers cannot be easily analyzed by a program. In fact, analyzing user opinion is still an active research topic in sentiment analysis [20]. The answers are usually processed by a human, which has a large delay no matter how the answers are collected. As Quest targets at instant feedback from the audience, we exclude open-ended questions from consideration. Additionally, it is

Fig. 2 Flowchart of Quest


① Prepare questionaire and build a dictionary
② Show the questionnaire via screen or papers
Respondent 1
AP
Computer
③ Set up a WLAN and start sniffing handshake packets
Respondent 2
④ Each respondent connects to the WLAN with his/her own answer as password
⑤ Crack out submitted answers based on sniffed handshake packets and the dictionary
Respondent n
Questionnaire Designer

inconvenient for respondents to type a general sentence in the small password interface.

3.3 Wi-Fi security protocol selection

To implement Quest, we need to select a Wi-Fi security protocol, which specifies the handshake process and how the entered password is hashed in the handshake frames. Currently, there are four available protocols, including Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA), WPA2 and WPA3. Theoretically, all of them can be used to implement Quest, but there are a few differences.

WEP has serious security vulnerabilities [16], and was deprecated in 2004. However, it is still supported by many Wi-Fi devices. When applied to Quest, a major limitation of WEP is that the password is restricted to 5 (WEP-40) or 13 ASCII characters (WEP-104). This would pose additional constraints on the number of questions, either 5 or 13 questions. In contrast, WPA, WPA2, and WPA3 all allow the use of passwords with no less than 8 ASCII characters. Thus, we did not choose WEP for Quest.

We finally choose WPA2, because it may be supported for a longer time than WPA, and currently, there are more devices supporting WPA2 than WPA3. We emphasize that both WPA and WPA3 can be used to implement Quest.

4 Password cracking: background and straightforward solution

4.1 Handshake process of WPA2

WPA2 uses a 4-way handshake between the client and the AP to prove to each other that they know the password. The handshake involves four EAPoL (Extensible Authentication Protocol over LAN) frames and is initiated by the AP after the association process.

The process has four steps. (1) The AP sends the first EAPoL frame to the client, including a random number, ANonce. (2) The client generates a random number, SNonce. It then constructs the PTK (Pairwise Transient Key) according to the password, ANonce, SNonce, and MAC addresses of both sides. It sends the second EAPoL frame to the AP, including SNonce and MIC (Message Integrity Check). (3) The AP constructs PTK with the password, ANonce, SNonce, and MAC addresses of both sides, with the same algorithm as the client. Then, the AP uses the PTK to verify the MIC of the second frame, and, upon verification success, sends the third EAPoL frame to the client, which contains a GTK (Group Temporary Key) and another MIC with the PTK. (4) The client uses its own PTK to verify the MIC of the third frame. If the verification succeeds, the client sends the fourth EAPoL frame to the AP, confirming the success of the handshake process.

We only need to capture the first two EAPoL frames. The third and fourth frames will not appear in Quest due to wrong password. The basic idea of password cracking is to find a string that can generate a MIC that matches the one in the frame.

4.2 From password to MIC

The complete process from password to MIC is shown in Fig. 3. According to IEEE Std. 802.11-2016, the PTK in Step 2 of the 4-way handshake is computed as follows.

$$\begin{aligned}\text{PTK} = \text{PRF-384}(&\text{PSK}, \text{``Pairwise key expansion''},\\ &S_{\min}||S_{\max}||N_{\min}||N_{\max}),\end{aligned}$$

where PRF-384 is a pseudo random number generation function using HMAC-SHA-1 as a subroutine, PSK is

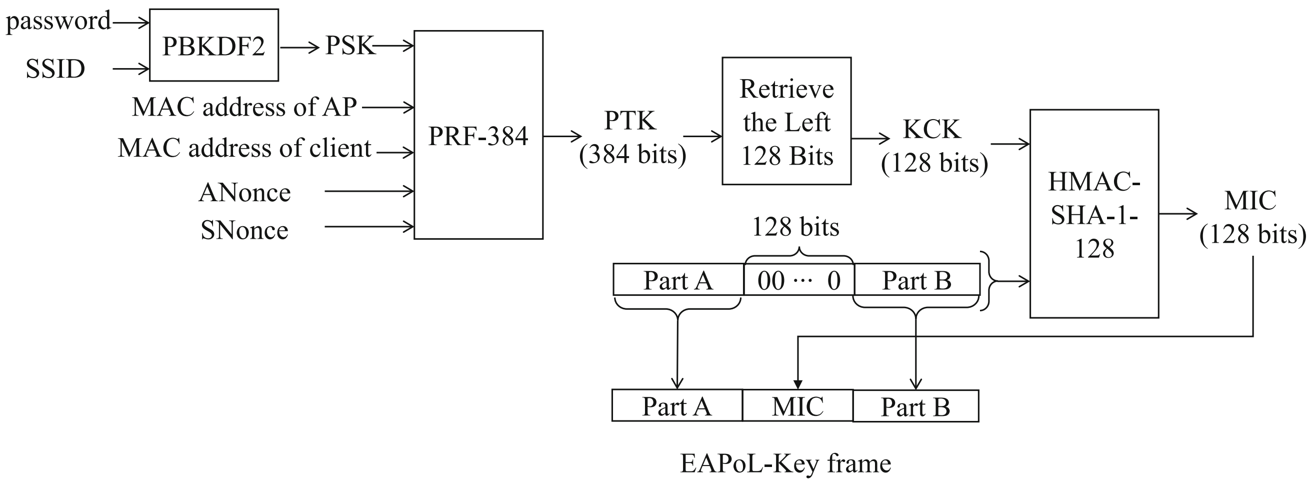


Fig. 3 The process from password to MIC in WPA2 protocol

constructed from the password, operator "||" concatenates the left bit string with the right bit string, $S_{\min}$ ($S_{\max}$) is the smaller (larger) MAC address among the client and the AP, and $N_{\min}$ ($N_{\max}$) is the smaller (larger) one of ANonce and SNonce. In the standard, the PSK is constructed from the password as $\mathrm{PSK} = \mathrm{PBKDF2}(password, ssid, 4096, 256/8)$, where PBKDF2 is a key derivation function defined in RFC 2898 Section 5.2, which takes four parameters as input and outputs a random binary string. Among the parameters, *password* and *ssid* are the password and SSID of the Wi-Fi represented as sequences of ASCII encoded characters, 4096 is the number of times the password hashed, and 256/8 is the number of bytes of the output. PBKDF2 relies on HMAC-SHA-1 for hashing, and calls HMAC-SHA-1 4096 times.

After the PTK is computed, the MIC is computed as follows:

$$\mathrm{MIC} = \text{HMAC-SHA-1-128}(\mathrm{KCK}, \mathrm{data}),$$

where HMAC-SHA-1-128 returns the first 128 bits of HMAC-SHA-1, KCK is the first 128 bits of PTK, and data is an EAPoL-Key frame with a 128-bit MIC field filled with zeros.

4.3 A Strawman cracking solution

The main task of cracking password is to find a password consistent with the following known variables: (1) MAC addresses of the AP and the client, which are included in the handshake frame header; (2) ANonce, which is generated by the AP and included in the first handshake frame from the AP to the client; (3) SNonce, which is included in the second handshake frame; (4) the second EAPoL handshake frame and (5) MIC, which is included in the second handshake frame.

Algorithm 1 specifies the cracking process. The AP continuously monitors the wireless interface to capture all handshake frames. Among the four handshake frames, it only deals with the first and second frames. The first handshake frame carries ANonce (lines 5-6). Upon receiving a second handshake frame, it records relevant information and calls the subroutine Find-Password to find the password entered by the respondent (lines 7-9).

Algorithm 1 Cracking password entered by the respondent

```
1: let the SSID of the network be $ssid$
2: construct a set $set\_of\_psw$ of candidate passwords
3: **while true do**
4:    capture a 4-way handshake frame $pkt$
5:    **if** $pkt$ is the first handshake frame **then**
6:       $anonce \leftarrow$ ANonce field in $pkt$
7:    **else if** $pkt$ is the second handshake frame **then**
8:       $psw \leftarrow$ Find-Password($set\_of\_psw$, $anonce$, $pkt$)
9:       record $psw$ as the answer of a respondent

Find-Password($set\_of\_psw$, $anonce$, $pkt$)
1: $(cpkt, v) \leftarrow$ Extract-Fields($anonce$, $pkt$)
2: **for** $c \in set\_of\_psw$ **do**
3:    $psk \leftarrow$ PBKDF2($c$, $ssid$, 4096, 256/8)
4:    $ptk \leftarrow$PRF-384($psk$, "Pairwise key expansion", $v$)
5:    $kck \leftarrow$ leftmost 128 bits of $ptk$
6:    $cmic \leftarrow$ leftmost 128 bits of HMAC-SHA-1($kck$, $cpkt$)
7:    **if** $cmic = mic$ **then**
8:       break the loop, and return $c$

Extract-Fields($anonce$, $pkt$)
1: retrieve from $pkt$ the MIC field as $mic$, the SNonce field as $snonce$, the MAC address of
   the client as $mac\_clt$, and the MAC address of the AP as $mac\_ap$
2: compare $mac\_clt$ and $mac\_ap$, and let $S_{\min}$ be the smaller one, and $S_{\max}$ be the greater
   one
3: compare $anonce$ and $snonce$, and let $N_{\min}$ be the smaller one, and $N_{\max}$ be the greater
   one
4: $v \leftarrow S_{\min}||S_{\max}||N_{\min}||N_{\max}$
5: copy $pkt$ to a new variable $cpkt$, and set the 128-bit MIC field of $cpkt$ to 0
6: **return** $(cpkt, v)$

PRF-$n$ ($K$, $A$, $B$)
1: **for** $i \leftarrow 0$ **to** $\lceil n/160 \rceil$ **do**
2:    $R \leftarrow R||$HMAC-SHA-1($K$, $A||0||B||i$)
3: **return** the leftmost $n$ bits of R
```

Find-Password takes as input three parameters, a set of all possible passwords, ANonce, and the second handshake frame. It repeatedly fetches a password from the set (line 2) and tests whether the fetched password would generate a matching MIC. In the case of a matching MIC, the fetched password is the one entered by the respondent, and the loop breaks (line 8). If no password yields a matching MIC, Find-Password terminates without returning anything. Extract-Fields preprocesses frames, and PRF-*n* is defined in the 802.11 standard. Note that PBKDF2 and HMAC-SHA-1 are standard functions, and their implementations are omitted. In the algorithm, "||" means bit string concatenation.

4.4 Long running time problem

A key problem in brute-force search strategy is how long it takes to break a password. The basic operation in Algorithm 1 is the checking step of Find-Password, that is, verifying whether a candidate password is correct (lines 3-6). We measure the computation time of PBKDF2, PRF-384, and HMAC-SHA-1 in lines 3, 4, and 6, respectively, on frames from a handshake process. We use a laptop computer with an I5-3230M CPU and 8G memory running Ubuntu 18.04.4. We implement the algorithm with Python 3.7.4, and use the built-in hashlib library to compute PBKDF2 and the hmac library to compute HMAC-SHA-1. We repeat the experiment 10,000 times and report the average computation time and standard deviation in Table 1.

Observe that in Find-Password, verifying a candidate string takes 3.83 ms. For a questionnaire with 8 four-choice questions, there are 4^8 candidate strings, which requires $4^8 * 3.8314ms = 4.18$ minutes to crack out the answer of a respondent. For 20 respondents, the time is 1.4 hours, which is not acceptable.

Breaking down the time consumption of the functions, we can see that PBKDF2 dominates the computation time. In general, the computation time of PBKDF2 is 84 times that of PRF-384 and 363 times that of HMAC-SHA-1. Note that PBKDF2 has already been optimized by Python since it makes 4096 calls to HMAC-SHA-1.

Table 1 Computation time (average ± standard deviation) of functions in Algorithm 1 on a laptop computer

PBKDF2	PRF-384	HMAC-SHA-1	Verification
3.78 ± 0.25ms	0.04 ± 0.01ms	0.01 ± 0.00ms	3.83 ± 0.26ms

5 Speeding up the cracking process

5.1 Ignoring redundant requests

We find that, in the event of a wrong password, the 4-way handshake process will be repeated for 4 times by the AP. Some clients will also retry the connection several times. It is a waste of time to crack the password on every 4-way handshake.

To address this problem, we mark a client's MAC address for each cracked 4-way handshake and ignore subsequent frames from clients that are marked. This can reduce the amount of computation to 1/4 of the original method.

5.2 Precomputing an offline password-PSK dictionary

In Fig. 3, there are three time-consuming steps: PBKDF2, PRF-384 and HMAC-SHA-1-128. We show that we can eliminate the PBKDF2 step by precomputing a PSK dictionary.

In particular, we fix the SSID of the network in advance. Then, the output of PBKDF2 depends only on the password. For each candidate password, we compute the PSK for the fixed SSID, and store the password-PSK combination as a key-value pair in a dictionary. To find the password, we enumerate all key-value pairs in the dictionary and take the value as PSK directly instead of computing it via the PBKDF2 function.

There are some remarks. First, the precomputed password-PSK dictionary can be used multiple times, as long as the SSID is the same. To reduce run-time cracking overhead, the computation can be done offline using GPU, supercomputers, or on many computers in parallel, before Quest is deployed. Second, Quest goes for exact matches, and if flexibility is allowed at the respondent's side, for example, 'a' and 'A' are equivalent, then both versions should be included in the dictionary. The reason is that PBKDF2 in Fig. 3 hashes similar inputs (case sensitive) into completely different outputs, and there is no known efficient algorithm that can use the similarity of inputs to facilitate cracking.

5.3 Reducing key length from 384 Bits to 128 Bits

We show that in the last step of computing the MIC, only 128 bits (i.e., KCK) of PTK are used. Thus, there is no need to compute the entire 384-bit PTK. It is sufficient to calculate the KCK directly instead of PTK. Then we prove that PRF-384 in Figure 3 can be simplified to PRF-128.

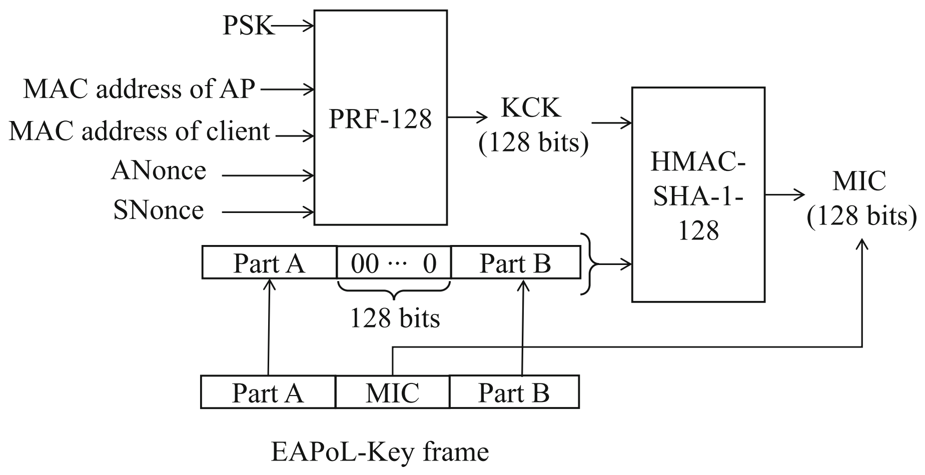


Fig. 4 A simplified process to find the password matching a given MIC

Lemma 1 *For any parameters psk and v, let*

$$ptk = \text{PRF-384}(psk, \text{``Pairwise key expansion''}, \text{v})$$

and kck be the leftmost 128 bits of ptk. Then

$$kck = \text{PRF-128}(psk, \text{``Pairwise key expansion''}, \text{v}).$$

Proof Let s be the string "Pairwise key expansion". By running PRF-384(psk, s, v) in Algorithm 1, we can see that ptk is the leftmost 384 bits of the following 480-bit string:

$$\text{HMAC-SHA-1}(psk, s||0||v||0)||\text{HMAC-SHA-1}(psk, s||0||v||1)$$
$$||\text{HMAC-SHA-1}(psk, s||0||v||2),$$

where the output of each HMAC-SHA-1 is a 160-bit string.

Because kck is the leftmost 128 bits of ptk, it is also the leftmost 128 bits of the 480-bit string, and is the leftmost 128 bits of $\text{HMAC-SHA-1}(psk, s||0||v||0)$, which is PRF-128.

The benefit of key length reduction is that the number of calls to HMAC-SHA-1 is reduced from 3 to 1.

Lemma 2 *PRF*-384 *makes* 3 *calls to HMAC-SHA*-1, *and PRF*-128 *makes* 1 *call to HMAC-SHA*-1.

Proof Observe that in PRF-n, the number of calls to HMAC-SHA-1 is equal to $\lceil n/160 \rceil$. Further note that $\lceil 384/160 \rceil = 3$ and $\lceil 128/160 \rceil = 1$.

5.4 Putting it altogether

By combining the above three techniques, we get Algorithm 2, which produces the same result as Algorithm 1 with significantly lower computation cost. Fig. 4 shows the simplified process to generate the MIC. The differences of Algorithm 2 from Algorithm 1 are in three aspects (commented in the algorithm).

First, a new subroutine Construct-PSKDict is used to precompute a password-PSK dictionary, before the AP begins to capture frames. It is straightforward to construct such a dictionary.

Second, a new handshake frame will not incur a password cracking process if the client has been marked, i.e., its answer has been cracked. To do this, handshake frames are recorded once it has been cracked. Note that the mark of a client can be easily cleared if a new questionnaire is distributed.

Third, when searching the password, PRF-128 is called instead of PRF-384. We have the following theorem.

Theorem 1 *Algorithm* 2 *produces the same set of answers as Algorithm* 1. *To verify whether a candidate password is correct*, *Algorithm* 2 *makes* 2 *calls to HMAC-SHA*-1, *and Algorithm* 1 *makes* 4100 *calls to HMAC-SHA*-1.

Proof In Algorithm 2, filtering out duplicate computations in line 6 does not change the set of cracked answers. According to Lemma 1, Find-Pass2 in Algorithm 2 is equivalent to Find-Password in Algorithm 1. Thus, the two algorithms produce the same set of answers. For the number of calls to HMAC-SHA-1, observe that Find-Pass2 makes 1 call in PRF-128 in line 3 (according to Lemma 2) and one call in line 4, giving 2 calls in total. On the contrary, Find-Password makes 4096 calls in PBKDF2 in line 3, 3 calls in PRF-384 in line 4 (according to Lemma 2) and 1 call in line 6, giving 4100 calls in total.

Algorithm 2 Speeding up the cracking process

```
1: let the SSID of the network be $ssid$
2: construct a set $set\_of\_psw$ of candidate passwords
3: $psk\_dict$ ←Construct-PSKDict($set\_of\_psw$, $ssid$) //Improvement: precompute an offline dictionary
4: while true do
5:     capture a 4-way handshake frame $pkt$
6:     skip to the next iteration if the client MAC of $pkt$ has been marked //Improvement: ignore redundant requests
7:     if $pkt$ is the first handshake frame then
8:         $anonce$ ← ANonce field in $pkt$
9:     else if $pkt$ is the second handshake frame then
10:        $psw$ ← Find-Pass2($psk\_dict$, $anonce$, $pkt$)
11:        record $psw$ as the answer of a respondent
12:        mark the client MAC of $pkt$

Construct-PSKDict($set\_of\_psw$, $ssid$)
1: build an empty dictionary $psk\_dict$
2: for $c \in set\_of\_psw$ do
3:     $psk$ ← PBKDF2($c$, $ssid$, 4096, 256/8)
4:     insert $(c, psk)$ as a key-value pair into $psk\_dict$
5: return $psk\_dict$

Find-Pass2($psk\_dict$, $anonce$, $pkt$)
1: $(cpkt, v)$ ← Extract-Fields($anonce$, $pkt$) //defined in Algorithm 1
2: for $(psw, psk) \in psk\_dict$ do
3:     $kck$ ←PRF-128($psk$, "Pairwise key expansion", $v$) //Improvement: reduce key length
4:     $cmic$ ← leftmost 128 bits of HMAC-SHA-1($kck$, $cpkt$)
5:     if $cmic = mic$ then
6:         break the loop, and return $psw$
```

To illustrate performance improvement, we implement Algorithm 2 in Python 3.7.4 and run it on the same laptop computer as in Sect. 4.4. Table 2 shows that PRF-128 needs 0.02 ms, half that of PRF-384 in Table 1. HMAC-SHA-1 needs the same computation time as Table 1. However, Algorithm 2 does not compute PBKDF2, so the verification of one password takes 0.03 ms, less than 1% of Algorithm 1. For a questionnaire with 8 four-choice questions, cracking the answer of a respondent takes $4^8 * 0.03ms = 1.97s$. For 20 respondents, the algorithm can finish in one minute. Note that this time only occurs in the worst case, that is, the whole dictionary is searched before an answer is found. In practice, the computation time is much shorter.

6 Theoretical limitation due to shared wireless media

Questionnaire collection speed (QCS) is the number of respondents whose answers are successfully collected by Quest per second. There are two factors that limit QCS, the computation time for cracking out the answers and the collection speed of handshake frames.

Observe that the cracking time can be made arbitrarily small by upgrading the hardware. In contrast, the second factor cannot be made arbitrarily fast. Handshake frames share a wireless communication channel and may collide with each other. In this section, we analyze a typical message exchange sequence of Quest, and calculate an upper bound on the speed of questionnaire collection due to the shared wireless channel. This upper bound can help to understand the fundamental limitations of Quest.

6.1 Standard message exchange sequence

Fig. 5 shows a typical message exchange sequence for a respondent with the AP in Quest, after the respondent clicks the 'connect' button. Note that each frame must be acknowledged by a short ACK frame; otherwise, the frame will be retransmitted (except broadcast frame such as probe request). In Quest, only the first two steps of the 4-way handshake occur. However, because the handshake fails (due to a wrong password), existing wireless drivers will repeat the first two steps multiple times before showing an incorrect password prompt to the user. The number of repetitions, m, is vendor dependent and is generally not configurable for commercial hardware APs. We have observed values ranging from 4 to 9 on commercial

Table 2 Computation time (average±standard deviation) of functions in Algorithm 2 on a laptop computer

PRF-128	HMAC-SHA-1	Password Verification
0.02 ± 0.01 ms	0.01 ± 0.00 ms	0.03 ± 0.01 ms

hardware APs, but theoretically it can be any positive integer. In software APs, such as hostapd, its value can be set with parameter 'wpa_pairwise_update_count', whose default value is 4. For consistency with hardware APs, we did not change this default value in the experiment.

In addition to the above frames, the AP broadcasts beacon messages periodically. These beacon messages will also occupy a valuable channel resource and should be considered when estimating QCS.

6.2 Estimating QCS with air time of frames

Directly computing QCS is complex. A frame may be lost due to noise or collision, so that a client may retransmit it multiple times. Each client may have a back-off timer before sensing the channel. To ease the analysis, we assume that (1) there is no propagation delay, and (2) there is no frame loss, that is, all frames sent by any party are decoded correctly at the receiver. Violation of these assumptions will decrease QCS. Thus, our analysis puts an upper bound on QCS.

The basic idea is to use the air time [21], the time of signals in the air, of the frames to estimate QCS. All handshake frames and beacons should occupy the wireless channel sequentially. In fact, to be correctly decoded, no two frames can appear simultaneously in the air.

We first give a formula for the air time of frames in a typical message exchange sequence as in Fig. 5. Denote by $P = \{0, 1, \ldots, 9\}$ the set of 10 possible frames[1], and the length of frame $i \in P$ be l_i. In addition, let the data rate for sending a frame be R.

Lemma 3 *For a message exchange sequence in Fig. 5, the air time it takes is* $T = \frac{(5+2m)l_9 + m(l_7+l_8) + \sum_{i=1}^{6} l_i}{R}$, *where m is the number of repetitions in ?*tic=?>Fig. 5.

Proof Simply examine the exchanged frames.

Theorem 2 *For a Quest system that sends n beacon messages per second, the number of questionnaires it can collect per second is at most* $\frac{L - nl_0}{TR}$, *where* l_0 *is the length of the beacon message, T is defined in Lemma 3, and L is the number of bits in the air per second with data rate R, i.e.,* $L = R \cdot (1 \text{ second})$.

Proof Observe that the n beacon messages occupy nl_0/R time. For the rest time, each questionnaire occupies T time due to Lemma 3. Therefore, the number of questionnaires per second is $\frac{(1 \text{ second}) - nl_0/R}{T} = \frac{L - nl_0}{TR}$.

Note that L may not be equal to R, since they have different units, that is, bits or bytes versus bits per second. For example, when $R = 1Mbps$, we have $L = 10^6$ bits or $L = 10^6/8$ bytes.

6.3 Empirical values

To understand QCS in practical systems, we sniff handshake frames and record typical length values. For parameter m, we find that it is 4 for hostapd by default [22], the software that implements SoftAP in Quest. Beacon frequency n is set to 10 in hostapd.

Among the frames, the frames sent by AP are of fixed size once the parameters, e.g., protocol version and SSID, are set. On the contrary, different clients may have variable lengths of their frames.

Experiments show that Probe Request, Authentication Seq 1, and Association Request frames vary with different clients, while the other frames are mostly the same. Therefore, we set $l_v = l_1 + l_3 + l_5$ and show its impact on QCS in Fig. 6. We consider data rates 1, 2 and 5.5 Mbps. Although a higher data rate (e.g., 54 Mbps) can be supported, we find that control frames and management frames are sent at a low data rate in the real world. One possible reason is that these frames are either broadcast traffic or are sent before the basic rate set has been negotiated. Thus, wireless card manufacturers choose to send these frames at the lowest basic rate. The most common data rate we observe is 1 Mbps.

Fig. 6 shows two phenomena. First, QCS decreases with increasing l_v. Thus, to increase QCS, we prefer shorter frames. Second, QCS decreases with the data rate. However, even for 1Mbps, we can collect about 50 questionnaires per second, which is sufficient for most applications. In practice, we may not achieve the QCS speed reported in Fig. 6. Factors including propagation delay, signal collision, back-off timer, DIFS and SIFS will contribute to lower QCS. We will conduct an experiment to study QCS in a controlled environment.

7 Implementation

Quest can be implemented on any computer with a wireless card that supports AP mode. We use hostapd [22] to create a WPA2 protected Wi-Fi network. Because handshake

[1] Frames labeled from 0 to 9 are Beacon, Probe Request, Probe Response, Authentication Seq 1, Authentication Seq 2, Association Request, Association Response, EAPoL Key 1, EAPoL Key 2, and ACK, respectively.

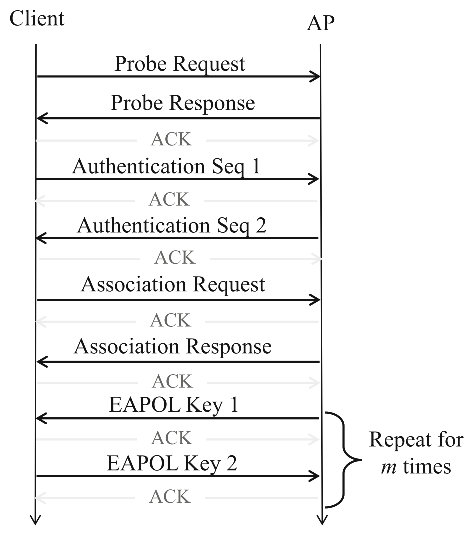


Fig. 5 Typical message exchange between a user and AP

frames are usually bursty, password cracking may block frame capture if the two operations are performed sequentially. We thus implement Algorithm 2 as two threads, a frame capture thread (the main thread), and a password cracking thread. Both are written in Python 3, forming a typical producer-consumer paradigm. The pseudocode is shown in Algorithm 3, and the source code is available at https://github.com/Xiaojun-Zhu/Quest.

Algorithm 3 One implementation of Quest in the AP

```
Main-Thread:
1: create a global queue for storing cracking tasks
2: run a cracking thread concurrently
3: construct a dictionary and start sniffing packets according to Algorithm 2
4: whenever Find-Pass2 should be invoked in line 10 of Algorithm 2, submit a crack-
   ing task containing all inputs to the global queue instead, and continue to sniff
   subsequent packets

Cracking-Thread:
1: while true do
2:     wait until there is a cracking task in the global queue
3:     retrieve a cracking task
4:     call Find-Pass2 in Algorithm 2
5:     store the cracked password
```

Specifically, the frame capture thread relies on Scapy [23], a Python program that supports frame sniffing. The thread monitors the 4-way handshake process and creates a cracking task named by the MAC address of the client upon detection of a matching pair of EAPoL 1 and EAPoL 2 frames. The task is sent to a task queue, and the client's MAC address is marked to avoid duplicate tasks. The password cracking thread repeatedly retrieves a task from the task queue and performs password cracking according to Algorithm 2. Before Quest deployment, an offline password-PSK dictionary is built and stored locally. It is loaded into memory before Quest runs.

Our hardware equipment is shown in Fig. 7, a laptop computer with an I5-3230M CPU (2.6GHz) and 8G memory running Ubuntu 18.04.4, which uses an external USB Wi-Fi adapter with RT3070L chip that supports AP mode. The hardware and software are summarized in Table 3. In particular, we use 802.11g (or Wi-Fi 3) for compatibility with more devices. It is fine to use Wi-Fi 4 (802.11n) or 5 (802.11ac) with suitable hardware, as these standards also rely on WPA 2.

8 Discussions

In this section, we discuss several related issues of Quest.

8.1 Local-web server approach and its downside

An alternative solution mentioned in Section 1 is to set up a WLAN and run a local web server. The server hosts a web application displaying questions to respondents and collects answers. In particular, compared to Quest, this approach involves the following steps:

1. Before meeting the respondents, the designer prepares the questionnaire as a web page.
2. The designer meets the respondents, sets up a WLAN through the AP, and runs a local web server on the computer.

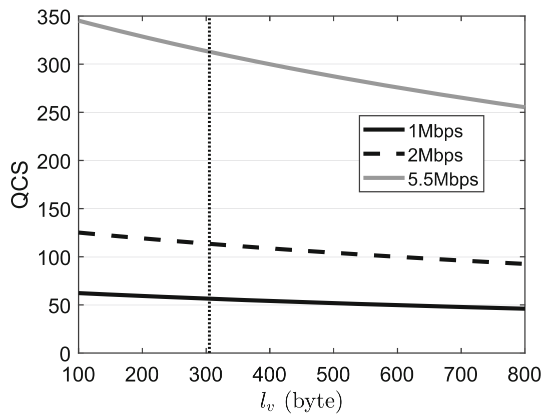


Fig. 6 Theoretical upper bound on QCS with respect to frame length l_v. The vertical dashed line corresponds to $l_v = 305$

3. Each respondent connects to the WLAN, opens a web browser, types in the IP address of the web server, fetches the questions and submits the answer.
4. The designer retrieves answers from respondents.

We implement one such prototype system with the same hardware as Quest, but we find that only a few respondents (10-15) can connect to the WLAN, i.e., the major downside of this approach lies in step 3. To find out the reason, we sniff the exchanged packets, and find many (unnecessary) frames including the four-way handshake frames, arp traffic, DHCP traffic, TCP retransmissions, and much unrelated traffic from background applications. The amount of possibly useful packets (TCP packets with the server IP address as either source or destination) is roughly 20% of the total captured packets. Thus, much precious air time is wasted by unrelated traffic. In addition, once a client connects to the Wi-Fi, it remains connected and occupies resources until the respondent manually disconnects. In contrast, Quest generates only a few packets and automatically disconnects respondents to allow others submitting answers.

8.2 Privacy concern and user identification

There may be two complementary concerns for Quest, privacy concern and user identification concern.

For privacy concern, a user might worry his/her answers being sniffed by a third party. Currently, we consider non-private questionnaires, like those that can be answered by raising a hand. When a user raises his hand, everyone else can see. Compared to the hand-raising method, Quest has an additional benefit. In the hand-raising method, if a respondent feels offended, he/she can choose not to raise a hand. In this case, the instructor may be misled because not raising a hand is a negative answer. This misleading phenomenon does not occur in Quest, because a 'no' response should be typed in manually.

For user identification concern, sometimes the instructor may want to connect an answer with a respondent. Though Quest does not provide such service, it can be easily augmented with a registration procedure, where a respondent submits the identity using Quest, thus relating the device's MAC address with the user. Note that the registration procedure is itself an independent Quest task, and is performed only once, e.g., at the beginning of a semester. It is an easy Quest task because the set of possible answers is just the set of user IDs. We do not consider ill-behaving clients who change MAC addresses.

8.3 Long cracking time for complex input

A limitation of Quest is that it does not support the collection of long sentences commonly seen as responses to subjective questions. Respondents can submit them and Quest can record the hashed values. However, we are unable to crack out the answers within reasonable time (e.g., within days or years). It is itself a challenging problem, because successful password cracking over long sentences will mean that the current Wi-Fi standards are insecure.

For long-sentence answers, we suggest alternative techniques such as the traditional handwriting method. In addition, for such long sentences, their analysis is challenging and is still an active research topic in sentiment analysis [20].

Quest targets at *instant* questionnaire collection and analysis. These scenarios usually use closed-ended questions defined in Sect. 3.2 and do not involve free text questions, which gives a relatively small dictionary size for cracking. Quest is suitable for such questions and has an acceptable delay in processing the answers.

8.4 Dealing with short answers and connection attempt restriction

For situations where the answers are even shorter than 8 characters, the respondents are not able to submit them via the password interface. IEEE 802.11i-2004 suggests that passwords should be 8 to 63 characters, and many Wi-Fi device manufacturers follow this suggestion. Among all mobile phones we can find, including Android phones and iPhones, if the respondent types in less than 8 characters, the 'connect' button in Fig. 1 is not clickable.

There are two workarounds. One approach is to employ zero padding by filling zeros to satisfy the 8-character requirement. There should be agreement on the padding style, that is, before or after the answer. The other approach is to write a special application running on smartphones to automatically pad zeros. This approach requires installation of applications on respondents' smartphones and is suitable for situations when Quest is repeatedly used, e.g., in offline classes.

A special application is also required in another possible scenario, where either the AP or the smartphone hinders a user from entering password after a few failed attempts. We do not observe such phenomenon in current devices, mainly because there is a more convenient attack for obtaining Wi-Fi password, the deauthentication attack [18]. If such attempt restriction is imposed in the future, Quest should change Wi-Fi SSIDs after several questionnaires and it is better to write an application for smartphones to automatically re-join WLAN.

8.5 Application-specific cracking time speed up

It is possible to devise application-specific algorithms to crack the answers of the respondents faster than Quest. In a take-attendance application, a respondent should type in an ID, e.g., a student ID. In this case, the collection of possible IDs is known in advance. The dictionary is relatively small, only up to a few hundred. The cracking time is about a dozen milliseconds using Algorithm 2. In a pop-up quiz application, respondents should type in the answers to a few questions whose correct answers are fixed. Then it is efficient to first try correct answers instead of arbitrarily exploring the dictionary. That is, the order of candidate responses that are explored matters. In summary, it is better to first guess the most probable answers and try them first during cracking. This is a natural solution for most hackers. In this work, we leave application-specific optimizations to application developers.

Fig. 7 A laptop computer with an external USB Wi-Fi adapter

8.6 Lost questionnaires

In Quest, even though we use the same wireless adapter for AP and sniffing, the response of a respondent may be lost due to frame collision, that is, the AP itself does not receive the handshake frames. In our implementation of Quest, the user should observe a timeout prompt from the device, instead of a wrong password prompt.

Generally, a respondent may observe two kinds of prompts after clicking ‘connect’ button, either incorrect password or timeout. If the respondent observes an incorrect password prompt, then the typed answer has been collected successfully by Quest. In this case, the AP has received handshake frames and will crack the typed answer. Unfortunately, if the respondent observes a timeout prompt, the typed answer may not be received by Quest. For example, if multiple respondents send out handshake frames simultaneously and the frames collide, the AP simply cannot decode anything. It is also possible that a respondent’s signals are weak and cannot be decoded in most time. In these cases, the typed answer gets lost.

Table 3 Hardware and software in implementation

Parameter	Value
Computer	Thinkpad T430
AP	External USB Wi-Fi adapter with RT3070L chip
CPU	I5-3230M, 2.6GHz
Memory	8 GB
Operating system	Ubuntu 18.04.4 (64 bit) with Linux kernel 5.4.0
Python	3.6.9
Scapy	2.4.4
hostapd	2.6
Wi-Fi operation mode	IEEE 802.11g
SSID	QuestAP
Authentication algorithm	Open System Authentication
Wi-Fi security protocol	WPA 2
Key management algorithm	WPA-PSK
Pairwise cipher	CCMP-128

Therefore, a respondent is suggested to reconnect to Wi-Fi upon observing a timeout prompt. This solves the problem of signal colliding. If the respondent has weak signals, then reconnection may not succeed either. Other measures should be taken.

Consequently, Quest may lose answers from some respondents. In fact, for respondents at distant locations, their signals may easily be corrupted by noise or signals from other respondents. The problem is similar to that of traditional printed questionnaires, since it is also difficult to guarantee the collection of all printed questionnaires. We will perform experiments to quantify the number of lost answers.

8.7 Cracking failure

Cracking may sometimes fail. Quest relies on a dictionary for cracking, so the answers not in the dictionary will not be cracked. Particularly, in iPhones, the password textbox is masked and the masking behavior cannot be turned off like Android phones.[2] In this case, a user has a higher chance to submit an invalid answer (a response not in the dictionary).

Generally, invalid answers cannot be cracked. Note that Quest targets at short answers, thus we believe that the chance of a user submitting an invalid answer is not high, in general. Upon cracking failure, Quest should notify the organizer that an invalid response is received and clear the mark of the corresponding client to wait for the next 4-way handshake. If the identity of the client is known (e.g., in a lecture), the organizer can inform the client that the typed answer is not valid.

9 Experiments

We show the feasibility of Quest, implemented in Sect. 7, by conducting real-world experiments. We are mainly interested in the following metrics.

- Time delay, the duration from the time a respondent clicks the 'connect' button to the time that Quest cracks out the input of the respondent, reflecting how timely the organizer can see the result;
- Success ratio, the ratio of successfully collected questionnaires to all submitted questionnaires;
- Questionnaire collection speed (QCS), the number of questionnaires collected by Quest per second.

The time delay includes the time due to message exchange in Fig. 5 and the cracking time due to Algorithm 2. The success ratio is determined by wireless interference. QCS is jointly influenced by time delay, success ratio, and the time in which respondents submit their responses.

9.1 Delay of quest with different answers

Measuring time delay is challenging. The start event occurs at the client, but the end event occurs at the AP. The clocks of the client and AP are usually not synchronized. Observing that the first frame sent by the client is a probe request frame, we use the timestamp of receiving the probe request frame as an approximation of the start time. This poses a second challenge, i.e., we need to capture probe frames, but when running in AP mode, a wireless adapter does not report 802.11 frames to any upper layer software, and we cannot sniff probe frames. To solve this problem, we installed a second wireless adapter, turned it into monitor mode, and sniffed 802.11 frames with this new adapter. Note that, the second adapter is only for evaluating Quest. It is not needed in Quest itself.

We use a smartphone as a client and consider a questionnaire with 8 four-choice questions, i.e., the answers range from 8 'A's to 8 'D's, giving a total of $4^8 = 65536$ possible answers. We built the password-PSK dictionary beforehand. In the experiment, we try submitting 4 answers, 8 'A's, 8 'B's, 8 'C's and 8 'D's. For each answer, we repeat the submission operation 10 times. We report the average time delay and its 95% interval in Fig. 8, where two major components, delay due to frame transmission and delay due to password cracking, are differentiated by color. Frame delay is the time until a cracking task is created (i.e., line 10 of Algorithm 2), and cracking delay is the time to find the password (i.e., the running time of Find-Pass2 in Algorithm 2).

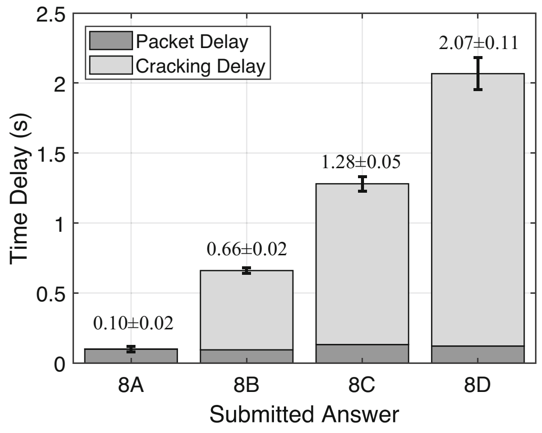


Fig. 8 Average time delay (± 95% confidence interval) of Quest for different answers submitted. Answer 8A is located at the head of the dictionary, representing the best case, and 8D is at the end of the dictionary, representing the worst case

[2] We believe the turn-off option may become available in the future given that iOS 16, released in 2022, can already show saved Wi-Fi passwords in clear text.

We can see that the frame delay is relatively stable, consisting of transmission of the probe response, authentication frames, association frames, and EAPoL frames. The frame delay is roughly 0.1 seconds across the whole experiment. In contrast, different answers have different cracking delays. In the best case, i.e., 8 'A's located at the head of the dictionary, the cracking delay is negligible. In the worst case where the whole dictionary is exhausted, i.e., 8 'D's, the cracking delay can be up to 2 seconds.

To understand the impact of computer hardware on the delay, we try two other computers to crack the password in the worst-case scenario, i.e., the 8D case. For a fair comparison, we use the captured packets in the first experiment as input to the cracking algorithm, which runs on a new computer, and add up the packet delay to produce the total delay of the computer.

- Computer 1: a laptop computer with Windows 11, CPU i7-10510U (2.3GHz), and 16G memory;
- Computer 2: a desktop computer with Windows 11, CPU i9-9900K (3.6GHz), and 64G memory.

Table 4 shows the results. We can see that computer 1 also requires 2 seconds, while computer 2 requires 1 second. This shows the potential of hardware upgrade on the performance of Quest.

Another interesting phenomenon that we find during this experiment is that a connection failure notice on the smartphone appears *after* the AP has received the password. The reason is that, for the 4 repetitions of EAPoL 1 and 2 frames (see Fig. 5), the AP waits for about 1 second before the next repetition; thus, the client only knows the connection failure after about 3 seconds.

9.2 Take-attendance in classroom

We use Quest in teaching practice. There are 29 students in a classroom in Fig. 9. Each student has a smart device, including iPhone and Android phones. Quest is deployed in the front of the classroom.

In the experiment, each student submits the student ID as an answer. In our university, a student ID is a 9-digit number. Because the set of possible answers is not large, the cracking time is negligible. To pose a challenge to Quest, we intentionally let students press 'connect' button at the same time, upon a voice command being issued.

Fig. 10 shows the results, where the detection of the first student is set as the start of time. Quest receives 23 student IDs within the first 4 seconds and receives the last 6 student IDs in 22 seconds. The main reason is that for the last 6 students, their handshake frames are lost in the first 4 seconds, and they automatically back off and retry later, which leads to extra delay. It is worth mentioning that if Quest is replaced by a Wi-Fi AP providing Internet access, then all these handshake frames will appear during the Wi-Fi connection phase, which means the delay of an Internet-based solution will be longer than Quest, even if the AP can support all these clients.

Table 4 Time delay (average ± 95% confidence interval in seconds) of three computers for 8D answer

Default	Computer 1	Computer 2
2.07 ± 0.11 s	2.03 ± 0.07 s	1.00 ± 0.04 s

9.3 Large-scale emulation

To understand the performance of Quest in a highly loaded environment, we design experiments to emulate large-scale clients who submit their answers, which should cause frame loss. By reducing the inter-packet delay of a Wi-Fi interface, we can emulate multiple clients from a Raspberry Pi 4B (RPi) computer.

To this end, we use 20 RPi computers with 1GB RAM, with each RPi computer emulating 20 clients. Each RPi computer is programmed to submit numbers from 1 to 20 through the Wi-Fi connection interface, representing 20 clients. There is no waiting time between submitting two numbers; that is, a client immediately tries the next number upon termination of the previous number.

The RPi computers are placed in a table shown in Fig. 11, simulating a noisy environment with heavy interference. The RPi computers begin submitting numbers upon power up, which share a power strip and are controlled by a common switch. We vary the number of RPi computers from 4 to 20 with increments of 4, thus emulating 80 to 400 clients with increments of 80. For each number, we repeat the experiment 10 times.

Fig. 12(a) shows the average success ratio with respect to the number of clients. We can see that with more clients, the success ratio decreases slightly. However, in the most crowded case, more than 97% questionnaires are still successfully collected.

For the same experiment, we compute the average QCS, the number of questionnaires successfully received divided by the total elapsed time. Fig. 12(b) shows that QCS increases with the increase of clients. This is because, with fewer clients, the number of questionnaires submitted per second is small, posing a limitation to QCS. In the experiment, QCS is 0.77 for 80 clients and 3.17 for 400 clients.

10 Conclusion

This paper proposes Quest, a questionnaire system that collects responses from respondents via the Wi-Fi password interface. Respondents type in their responses to the

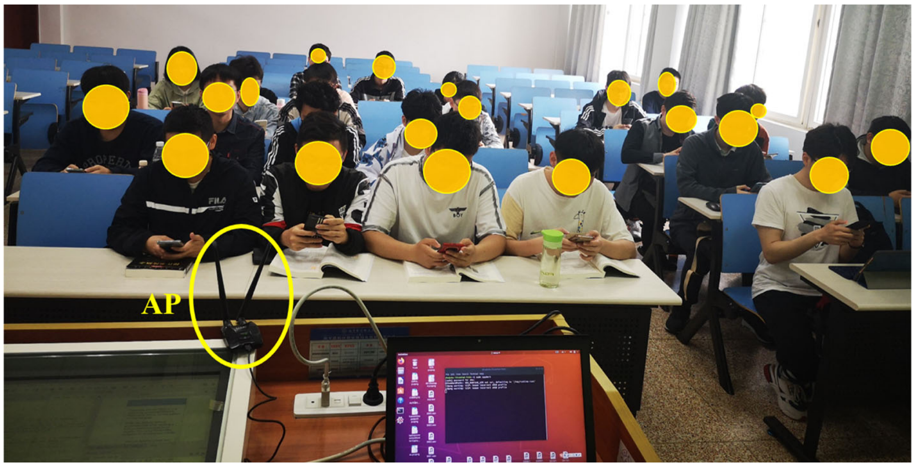


Fig. 9 Classroom experiment with AP in the front. Some students are out of the picture

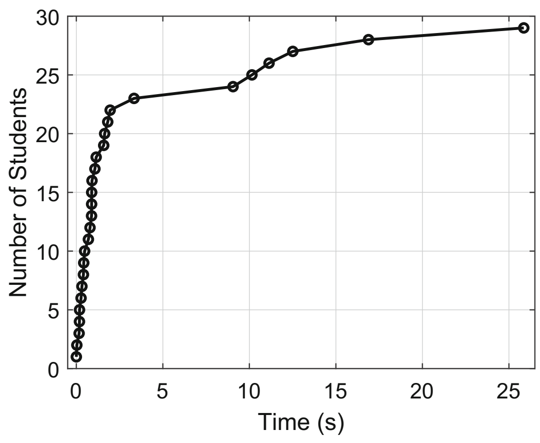


Fig. 10 Number of detected students with respect to time

questionnaire as passwords and connect to a specified AP, which then cracks out the passwords to retrieve the answers. The novelty of Quest over traditional password-cracking techniques is that it uses a questionnaire-cracking co-design. From the questionnaire side, it restricts the set of questions, reducing the cracking time from hours to minutes. From the cracking algorithm side, it proposes a specialized cracking algorithm that only considers the first 128 bits of a key, rather than 384 bits in traditional cracking, reducing cracking time from minutes to seconds. In real-world experiments, we show that for a questionnaire consisting of 65, 536 possible answers, the delay of answer collection per questionnaire is within 2 seconds on a laptop computer, and within 1 second on an upgraded desktop computer, including both packet delay and cracking delay. In a take-attendance application with 29 students, Quest collects 80% student IDs within 4 seconds, and over 90% within 20 seconds. These experiments verify the feasibility of Quest.

There are some potential extensions over Quest. First, Quest targets at planned questionnaires where the set of possible answers is known, and thus a dictionary containing all possible inputs can be built in advance. For open-ended questions, other mechanisms should be designed. Second, Quest can be implemented as a parallel algorithm, and it can offload the cracking task to a cloud. Extension in this direction can further reduce the cracking delay.

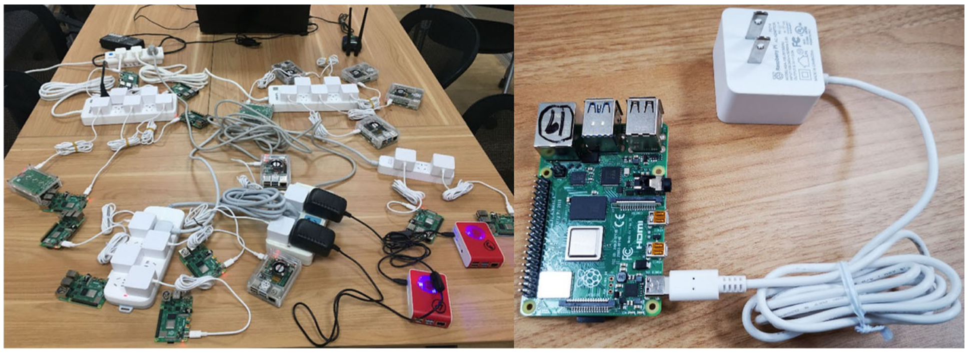


Fig. 11 Twenty Raspberry Pi computers are in a conference table (left) and a single Raspberry Pi computer (right)

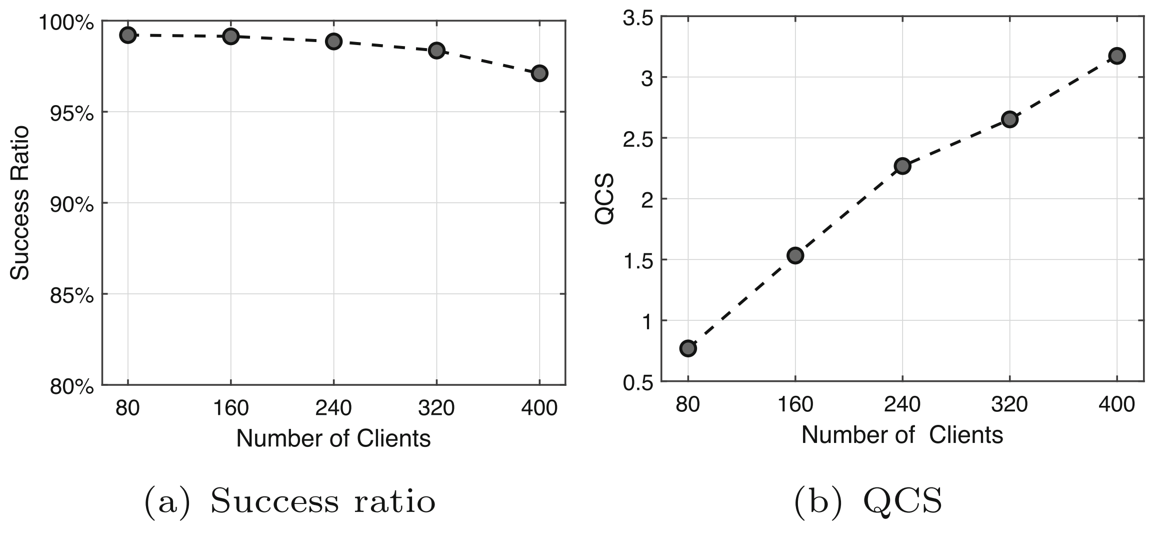


Fig. 12 Impact of client number on success ratio and QCS

Funding This work was supported by the National Natural Science Foundation of China (62372230, 61972199, 61972200).

Declarations

Competing interests The authors declare that they have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.

Ethics approval: Not applicable.

Data and materials: Available at https://github.com/Xiaojun-Zhu/Quest.

References

1. Zhang, C., Hei, X., Bensaou, B. (2019). A measurement study of campus wifi networks using wifitracer. In S. Guo, D. Zeng (eds.) Cyber-Physical Systems: Architecture, Security and Application, pp. 19–42
2. Krosnick, J. A. (2018). 2018. In D. L. Vannette & J. A. Krosnick (Eds.), *Questionnaire Design* (pp. 439–455). Cham: Springer.
3. Manfreda, K. L., Batagelj, Z., & Vehovar, V. (2002). Design of web survey questionnaires: Three basic experiments. *Journal of Computer-Mediated Communication, 7*(3), 731.
4. Vicente, P., & Reis, E. (2010). Using questionnaire design to fight nonresponse bias in web surveys. *Social Science Computer Review, 28*(2), 251–267.
5. Shao, S., Zheng, J., Zhong, C., Lu, P., Guo, S., & Bu, X. (2023). IEEE 802.11ax meet edge computing: AP seamless handover for multi-service communications in industrial WLAN. *IEEE Transactions on Network and Service Management, 20*(3), 3396–3412. https://doi.org/10.1109/TNSM.2023.3239404
6. Chandra, R., Padhye, J., Ravindranath, L., & Wolman, A. (2007) Beacon-stuffing: Wi-fi without associations. In *Proceedings of HotMobile*
7. Lee, H., Kim, J., Joo, C., Bahk, S. (2019) BeaconRider: Opportunistic sharing of beacon air-time in densely deployed WLANs. In Proceedings of ICNP
8. Ma, C., Wu, B., Poslad, S., & Selviah, D. R. (2022). Wi-fi rtt ranging performance characterization and positioning system design. *IEEE Transactions on Mobile Computing, 21*(2), 740–756. https://doi.org/10.1109/TMC.2020.3012563
9. Zhang, L., Zhu, X., Wu, X. (2019) No more free riders: Sharing wifi secrets with acoustic signals. In Proceedings of ICCCN, pp. 1–8
10. Todtenberg, N., & Kraemer, R. (2019). A survey on bluetooth multi-hop networks. *Ad Hoc Networks, 93*, 101922.
11. Yan, Y., Yang, P., Xiong, J., & Li, X. (2022). Opencarrier: Breaking the user limit for uplink MU-MIMO transmissions with coordinated aps. *ACM Trans. Sens. Networks, 18*(2), 19–11921.
12. Matheus, L. E. M., Vieira, A. B., Vieira, L. F. M., Vieira, M. A. M., & Gnawali, O. (2019). Visible light communication: Concepts, applications and challenges. *IEEE Communications Surveys Tutorials, 21*(4), 3204–3237.
13. Zhou, M., Wang, Q., Lei, T., Wang, Z., & Ren, K. (2018). Enabling online robust barcode-based visible light communication with realtime feedback. *IEEE Transactions on Wireless Communications, 17*(12), 8063–8076. https://doi.org/10.1109/TWC.2018.2873731
14. Zhang, K., Zhao, Y., Wu, C., Yang, C., Huang, K., Peng, C., Liu, Y., & Yang, Z. (2021). Chromacode: A fully imperceptible screen-camera communication system. *IEEE Transactions on Mobile Computing, 20*(3), 861–876. https://doi.org/10.1109/TMC.2019.2956493
15. Kanta, A., Coisel, I., & Scanlon, M. (2020). A survey exploring open source intelligence for smarter password cracking. *Forensic Science International: Digital Investigation, 35*, 301075.
16. Bittau, A., Handley, M., & Lackey, J. (2006). The final nail in WEP's coffin. In *Proceedings of S &P*.
17. Vanhoef, M., & Piessens, F. (2017). Key reinstallation attacks: Forcing nonce reuse in wpa2. In *Proceedings of CCS*.
18. Schepers, D., Ranganathan, A., & Vanhoef, M. (2022). On the robustness of wi-fi deauthentication countermeasures. In *Proceedings of WiSec*. https://doi.org/10.1145/3507657.3528548
19. Stute, M., Heinrich, A., Lorenz, J., Hollick, M. (2021) Disrupting continuity of apple's wireless ecosystem security: New tracking, DoS, and MitM attacks on iOS and macOS through bluetooth low energy, AWDL, and Wi-Fi. In 30th USENIX Security Symposium (USENIX Security 21), pp. 3917–3934.
20. Feldman, R. (2013). Techniques and applications for sentiment analysis. *Communications of the ACM, 56*(4), 82–89.

21. Wang, W.-J., Yang, H.-C., & Alouini, M.-S. (2018). Wireless transmission of big data: A transmission time analysis over fading channel. *IEEE Transactions on Wireless Communications, 17*(7), 4315–4325. https://doi.org/10.1109/TWC.2018.2822801
22. hostapd https://w1.fi/hostapd/ (2023)
23. Scapy https://scapy.net/ (2023)

Xiaojun Zhu is an associate professor in the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. He obtained his B.S. degree and Ph.D. degree in computer science from Nanjing University in 2008 and 2014, respectively. From August 2011 to August 2012, he was a visiting scholar at the College of William and Mary. He is a senior member of IEEE and CCF, and an associate editor for Computer Communications. His research interests include wireless networks, UAV networks, and smartphone systems.

Hao Han received the B.S. degree in computer science and technology from Nanjing University, Nanjing, China, in 2005, and the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2014. He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include system and software security against various types of threats.