FEATURE ARTICLE

Matrix Barcode Based Secure Authentication without Trusting Third Party

Yongge Wang
University of North Carolina, Charlotte

Khaled M. Khan
Qatar University

The user creates a JavaScript-enabled bookmark file once, and opens this file to generate QR code every time to use secure servers. The mobile device generates a secret to encrypt her credentials, saves it so that the browser could access to, thus enabling her to log in to secure servers.

The QR codes are two-dimensional bar codes storing certain information. Users normally use QR codes based authentication schemes to defeat phishing, spyware, key loggers, and other attacks. Most of the existing QR authentication schemes use various techniques such as trusting third-party online password storage servers, using embedded challenge nonce, deploying tamper resistant IC chips, modification of web servers, etc. This article proposes a QR codes based secure web server authentication technique without a trusted third party. A user logs in to a web site by scanning a QR code to verify the server without entering a user ID and password. The protocol uses a shared secret between the user's device and the server.

A hardware based authentication approach uses USB and NFC dual-interface token with embedded tamper resistant IC chips to facilitate users to log in to web servers[1]. A browser app is required to communicate with the IC-chips using USB or NFC channel. This approach has been later extended to indirect NFC authentications by using QR codes to simulate the USB or NFC interface[2-3]. There are also recommendations to use QR codes for password storage. For example, TiQR[4] and MyDigiPass[5] market their password storage servers. In their products, a user registers an account within the marketed servers and stores in the registered account her user name and passwords for different web services. Each time when a user wants to log in to a web

server, the user retrieves the password from the servers by scanning the corresponding QR image. In these schemes, the user has to trust the password storage servers such as TiQR or MyDigiPass, which is unacceptable for most users. In the TruWalletM solution, the password wallet is stored in the trusted component of a phone[6]. It is only applicable to surf web pages via mobile browsers.

In the contemporary cyber world, the general sentiment of security-aware users is not to trust anyone except those on which they have sufficient control. Control influences them significantly on what they are going to use, and how much trust they should place on them[7]. To address these, we introduce a matrix barcode based secure authentication technique that does not require the user to trust any third party. From a security perspective, we claim that our protocol achieves the following security features:

- Malicious software is unable to steal user credential.
- Adversary cannot retrieve the user credentials even if the PIN or fingerprint protection of mobile phone is compromised.
- Reduce the risk of insecure weak randomness generators deployed in the mobile device or browser.

Furthermore, our approach ensures the following characteristics.

- It does not require users to trust any third party.
- The user holds reasonable control over the protocol.
- Web servers need no modification to support the protocol.

How can we achieve all these? Let's find out.

SECURE AUTHENTICATION WITH MOBILE DEVICES

We assume that users trust their mobile devices, and lock their mobile phones using PIN or fingerprints. If the phone is lost, the user can remotely destroy the data on the phone or disable the phone before an attacker breaks the PIN or fingerprint protection of the phone. Thus it is reasonable to use mobile devices as password vaults to facilitate the authentication process. Furthermore, mobile devices are beginning to integrate tamper resistant Secure Elements within them. For example, Apple has included Secure Element components in their iPhone 6 for their Apple Pay service. We expect that other mobile devices will also include tamper resistant Secure Element components for user's applications in future. These tamper resistant components are perfect choices for user password vaults. Does it sound similar to existing password vault solutions such as TiQR[4]? Not really.

In order for mobile devices to serve as password vaults, a two way communication channel between the Desktop browser and the mobile device is required. Though NFC/RFID, USB, WiFi, Bluetooth, Infrared Port, local area network, and other techniques could be used as a two way communication channel between the Desktop browser and the mobile device, our protocol is not limited to these channels. The protocol works as follows:

1. The user stores passwords for different web sites within the mobile device password vault. If possible, the password vault should be located within a tamper resistant component of the mobile device. A user PIN or fingerprint is required to access the password vault. The password vault also holds a private key a together with the elliptic curve public key cryptography parameters. The corresponding public key for this private key is aG.
2. The user creates a JavaScript enabled bookmark file that is stored in a local computer or a public location such as the Dropbox Public directory. The only requirement is that the user computer has access to this bookmark file regardless of the user's location. Note that the adversary has also access to this bookmark file. The bookmark file contains a list of web servers that the user wants to visit. The bookmark file also contains

the public key aG of the password vault and JavaScript for the following functionalities:

- QR code generation for each web server account
- Public key agreement protocol negotiation
- Automatic check of messages from the mobile devices (e.g., from a public cloud storage server, or from other channels such as USB, NCF/RFID, Bluetooth, etc.), and
- Automatic connection to the web server after user credentials are retrieved from the mobile device.

Each time when a user wants to log in to a server, the user opens the bookmark file within the Desktop computer browser. The user clicks the server link (e.g., hotmail.com) that (s)he wants to log in. The JavaScript within the bookmark file generates a QR code and displays the image within the Desktop browser.

The user scans the QR code in the browser using the mobile device that contains the password vault. This initiates a screen on the mobile device for the user to authenticate herself to the password fault. The authentication depends on the mobile device capabilities. For example, it could be fingerprint only authentication, PIN only authentication, or both. The QR code image contains the URL that the user wants to log in, the account name, and a Diffie-Hellman key agreement message from the browser which is generally in the format of z_2G. It may also further contain an optional message confirmation tag to show the knowledge of the private ephemeral exponent z_2 by the browser. For example, this could be achieved by authenticating some public message using a key derived from (aG, z_2). The details will be presented in the next section.

The mobile device will then choose the user's own Diffie-Hellman(DH) key agreement message which is generally in the format of z_1G, compute the session key K using a key agreement protocol for DH messages (z_1, z_2G, aG), retrieve the credentials (e.g., user-name and password) for the desired web server, encrypt the credentials using the session key K, authenticate the entire message by creating an authentication tag and deliver this encrypted and authenticated message to the Desktop computer browser using the chosen communication channel. In our prototype, we use the Dropbox Public directory with a specific file name that the browser knows. That is, the mobile device just writes the encrypted and authenticated credentials to a file in the Dropbox Public directory. After the Desktop computer browser displays the QR code in the browser, it keeps checking the communication channel from the mobile device. In our proof of concept implementation, it keeps checking the existence of a file with a specific name within the Dropbox Public directory. After it receives the message from the mobile device, it checks the authentication tag and decrypts the credentials using the session key computed from (z_1G,z_2,aG). The browser then logs the user to the remote server by submitting the decrypted credentials.

Security models and assumption

In our protocol, the adversary is allowed to control the communication links, the storage device that contains the JavaScript enabled bookmark file, and the communication channel between the mobile device and the browser (in our case, the Dropbox Public directory). However, the adversary is not allowed to control the web server as well as the browser. On the other hand, the security of our protocol is dependent on the security of the authenticated Diffie-Hellman key agreement protocol (in our case, the MQV key agreement protocol), and the security of the session key encryption scheme (in our case, the AES encryption scheme).

Other implementation choices

In this protocol, the password vaults are located within the protected mobile device storage. The user may choose to store the encrypted password vaults in any public location such as a cloud storage. Of course, it is required that the password vault be encrypted using a strong key that the user needs to memorize or be stored in a protected mobile device. Since the password vault is encrypted using a strong cipher with a strong key, the user does not need to trust the server that

hosts the encrypted password vault. Each time, when the cell phone accesses the password vault, the user needs to enter a strong key to the mobile device if the mobile device does not have the strong key already, or the user only needs to authenticate to the mobile device to unlock the strong key in its protected storage.

PROOF OF CONCEPTS

We developed a working prototype implementing the protocol as a proof of concept. In our proof of concept implementation, we use QR codes as the communication channel from a Desktop computer to a mobile device and we use public cloud storage services (e.g., Dropbox, iCloud, etc.) as the communication channel from a mobile device to a Desktop computer. Specifically, the mobile device installs the Dropbox app so that it has *write* access permission to the Public directory of the cloud service. Meanwhile, the Desktop browser could access the Dropbox Public directory using the HTTP protocol. As an alternative, we've also implemented a small web server on the Desktop computer to accept messages from a mobile device in case that the Desktop computer and the mobile device are on the same local area network, such as both of them are connected to the same WiFi access point. We choose to use the Elliptic curve parameters for the curve `P-256` (where the NIST name is `nistp256` and SECG name is `secp256r1`[8]). The `secp256r1` are specified by a six-tuple $T = (p, a, b, G, n, h)$ with the finite field F_p is given by

$p =$ FFFFFFFF 00000001 00000000 00000000
00000000 FFFFFFFF FFFFFFFF FFFFFFFF

$$= 2^{224}(2^{32} - 1) + 2^{192} + 2^{96} - 1$$

The elliptic curve is defined by $y^2 = x^3 + ax + b$ over F_p with

$a =$ FFFFFFFF 00000001 00000000 00000000
00000000 FFFFFFFF FFFFFFFF FFFFFFFC
$b =$ 5AC635D8 AA3A93E7 B3EBBD55 769886BC
651D06B0 CC53B0F6 3BCE3C3E 27D2604B

It should be noted that point compression techniques can be used to reduce the information that needs to be included in the QR codes. The point compression techniques were originally patented by Certicom though the patent has expired. Thus one can use it freely. The base point G in uncompressed form is as follows:

$x_G =$ 6B17D1F2 E12C4247 F8BCE6E5 63A440F2
77037D81 2DEB33A0 F4A13945 D898C296
$y_G =$ 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16
2BCE3357 6B315ECE CBB64068 37BF51F5

The order n for the point G is

$n =$ FFFFFFFF 00000000 FFFFFFFF FFFFFFFF
BCE6FAAD A7179E84 F3B9CAC2 FC632551

The cofactor $h = 1$.

In our prototype, we used a variant of the HMQV[9] key agreement protocol. We first describe the original HMQV protocol. In order for Alice (password vault) to establish a shared secret with Bob (Desktop computer browser) using the P-256 curve that we have just described, the protocol proceeds as follows, where we assume that Alice's private/public keys are a and $P_A = aG$ respectively, and Bob's private/public keys are b and $P_B = bG$ respectively.

1. `Bob` chooses a random z_B and sends the point $Z_B = z_B G$ to `Alice`.
2. `Alice` chooses a random z_A and sends the point $Z_A = z_A G$ to `Bob`.
3. `Alice` and `Bob` compute d= SHA-256(Z_A,`Bob`) and e = SHA-256(Z_B, `Alice`).

4. `Alice` computes the shared key $K = (z_A + da)(Z_B + eP_B)$.
5. `Bob` computes the shared key $K = (z_B + eb)(Z_A + dP_A)$.

For the protocol, the bookmark file is stored in a public location. In our experiment, it is accessed using the address: https://dl.dropboxusercontent.com/u/7499393/js/QRmk.html. Thus the private key b cannot be stored in the bookmark file. There are three approaches to address this challenge.

- In the first approach, we use (z_B, Z_B) both as the ephemeral session key pair and the permanent key pair. In other words, we set $P_B = Z_B$ and $b=z_B$.
- In the second approach, each time when the user opens the bookmark file in the Desktop computer browser, the browser asks the user to enter a random string which is used as a seed to generate a random private key b for the browser on the fly (alternatively, the browser generates a random private key b using randomness from the computer random source). In this approach, the QR code needs to include the public key $P_B = bG$ so that the mobile device could carry out the key agreement protocol calculations.
- In the third approach, the browser long term private key b is chosen at the system set up phase, and the mobile device password vault holds the public key bG in its protected storage. For this approach, it is not safe to store the long term private key b in the bookmark file. In the system set up phase, the user provides a secret seed α to generate the private key b. Each time when the user opens the bookmark file, the user needs to input this secret value α to the browser so that the browser can calculate the private key b. In other words, this value α could be considered as a lock to the bookmark file, and it adds an additional layer of security for lost phones as we will discuss in the next section.

We implemented the password vault on Android platforms and tested our prototype on a Galaxy S7 phone. The account user-names and passwords are stored in a text file, and the file is encrypted using AES-128 with a user chosen PIN. In the prototype test, we used the following private key for the mobile device password vault.

$a =$ `66A7BFA1 312CC13F 9C365E6D 05CA1142`
`03CF1C4A E9A1BCAE A4850FB4 8EDCDC3A`

The corresponding uncompressed public key $P_A = aG$ is

$^{x}P_A =$ `F7118AED 2DF5226D FC9A937A 09F1619D`
`A36343E1 7ED754F6 853508B8 6534C4C6`
$^{y}P_A =$ `B91A9D17 15C1676F 8864C8D3 4B2E01BB`
`0C6EB404 4493D102 8BBB6E97 C6E67A11`

As an example, the bookmark file is located at the address:

https://dl.dropboxusercontent.com/u/7499393/js/QRmk.html

where 7499393 is the Yongge.wang's DropBox account ID. Please note that after prototype testing, we disabled this page. Generally this kind of page is user specific and not made public.

We opened it in the browser and entered a seed first-test for the browser to generate the private key b. Then we clicked the gmail button to log into the gmail server and a QR code is displayed in the browser. Figure 1(a) and (b) show two screen shots of the browser QR codes. Figure 1(a) uses HMQV without point compression, and Figure 1(b) uses HMQV with point compression.

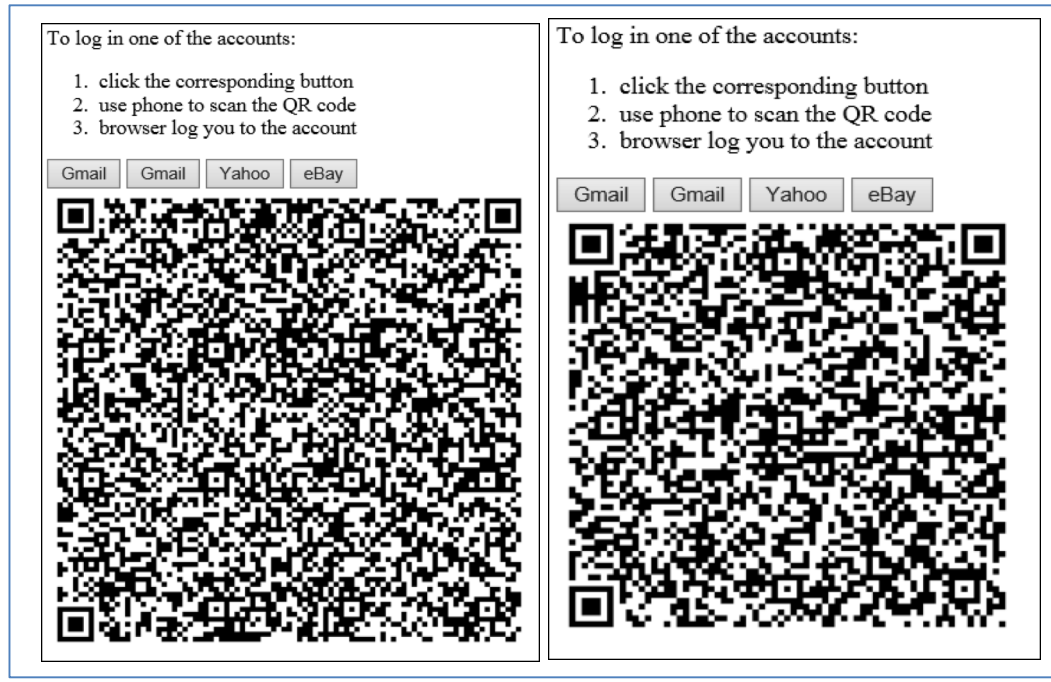

Figure 1: (a) The QR codes generated in the Browser without EC point compressions (left-hand image); (b) The QR codes generated in the Browser with EC point compression (right-hand image).

The QR code in Figure 1(a) contains the following string
`http://www.gmail.com/,yongge.wang,36959C42CF9299526E941FC347A1C9B061DC23420289B7202EBCCC833D5C329E#D0ED2FDD12854279467417BD6E5D7D969E65F54B9823DDC8AA801631E919BC5B,8C26456063E21A728280870D2D0CDE2E16C3A39C70E65629C8FFDE05AACB772B#6AAA1F467C5B6E0B3741E3D6E66F4277DCBCBAC56225EC11ECFF5B082C9EF43C` and the QR code in Figure 1(b) contains the following string
`https://www.gmail.com/,yongge.wang,0336959C42CF9299526E941FC347A1C9B061DC23420289B7202EBCCC833D5C329E,038C26456063E21A728280870D2D0CDE2E16C3A39C70E65629C8FFDE05AACB772B`

We used an Android phone that contains the password vault to scan the QR codes in Figure 1(a) and Figure 1(b). After the App scans the QR code, it will retrieve the URL address `http://www.gmail.com/`, the user name `yongge.wang`, and the two public keys: Z_B and P_B. Note that the QQ code in Figure 1(b) used compressed version of the public key point, thus the strings for the public keys are short.

By debugging the JavaScript, we obtained the ephemeral and permanent private/public key pairs. The private key b is

$b =$ `936AFA09 474CC76A 6AA42182 D5DC982F 38607317 B2FBF80D EB48E552 F1A895FF`

The corresponding uncompressed public key $P_B = bG$ (this is contained in the QR code image) is

$x_{P_B} =$ `36959C42 CF929952 6E941FC3 47A1C9B0 61DC2342 0289B720 2EBCCC83 3D5C329E`

$y_{P_B} =$ `D0ED2FDD 12854279 467417BD 6E5D7D96 9E65F54B 9823DDC8 AA801631 E919BC5B`

The ephemeral private key z_B is

$z_B =$ 505DEA00 167379FB C938CFCB 7B5EAAEA
81DD23A5 61956702 CC36792B 6A84BC5D

The corresponding uncompressed ephemeral public key $Z_B = z_B G$ (this is contained in the QR code image) is

$x_{Z_B} =$ 8C264560 63E21A72 8280870D 2D0CDE2E
16C3A39C 70E65629 C8FFDE05 AACB772B

$y_{Z_B} =$ 6AAA1F46 7C5B6E0B 3741E3D6 E66F4277
DCBCBAC5 6225EC11 ECFF5B08 2C9EF43C

The Android phone calculates the HMQV session key K. Using this session key, the phone calculates an encrypted string $C = \text{AES-128}_K(\texttt{yongge.wang}, \texttt{passwd})$ where `passwd` is the password for the account `yongge.wang` at `gmail.com`. The mobile phone then creates a file `D0ED2FDD` in the Dropbox Public directory. The file `D0ED2FDD` contains both the ciphertext C and the HMQV key agreement ephemeral public key $Z_A = z_A G$ in uncompressed format, where $Z_A = (x_{Z_A}, y_{Z_A})$ is

$x_{Z_A} =$ 1ED7829B 6A844809 D4E94A91 CE665678
F0EA0A9F 66D61E10 342902D0 B86FD478

$y_{Z_A} =$ B0325499 72DABC8C 2A349FBF 640E0175
86A2602C 300E3EFC EC790091 C8788B3B

The corresponding ephemeral private key z_A for the password vault is

$z_A =$ 9058F57C 0B8C243B 55BEF822 58466E58
7B76332A BE5CACA3 012E9871 BFE422E2

Note that the file name `D0ED2FDD` is a prefix of y_{P_B}.

After the browser displays the QR codes in Figure 1(a), it keeps checking whether there is a file `D0ED2FDD` in the Dropbox Public directory. Alternatively, we may also let the user to click a button in the browser to initiate the process of fetching the file `D0ED2FDD` in the Dropbox Public directory after the user finishes scanning the QR code and authenticating to the mobile device. Shortly after the file `D0ED2FDD` is written to the Dropbox Public directory, we observed that the browser automatically logs to the gmail server with the user-name: `yongge.wang`. It should be noted that the mobile phone is required to download the Android Dropbox app to the phone so that it can have write permission to the Dropbox Public directory. On the other hand, the Desktop computer needs not to install the Dropbox client since the browser can access the file `D0ED2FDD` using HTTPS protocol from the location: https://dl.dropboxusercontent.com/u/7499393/D0ED2FDD.

KEY CONTRIBUTIONS TO SECURITY

The protocol is expected to contribute significantly to the following security concerns regarding the QR code-based authentication. These are briefly outlined in the following sections.

Keystroke logger and malicious software

In our protocol, no user name or password is entered via the keyboard. Thus a malicious software such as a key logger will not be able to steal user credentials. Furthermore, only JavaScript within the browser has access to the user credentials. Thus unless the malware has access to the browser execution environment, it is not able to steal the user credentials either.

Trusting third party

In most recommended single sign-on protocols (e.g., existing password vaults etc.), the user has to trust a third party to manage his passwords. This is normally un-acceptable in practice. In our protocol, the only trusted entity is the protected storage of user's mobile devices.

Lost or theft phone

Most mobile devices provide the capability of remotely deleting the content on the devices if they are lost, assuming devices are connected to the Internet. If an adversary manages to break the protection of the mobile device, it cannot recover the user credentials because they are stored in the tamper resistant storage such as the Secure Element on mobile phones. However, the adversary may try to open the bookmark file in a browser and scan the QR codes to retrieve credentials to a specific web site. This attack could be further defeated by requiring the user to input a secret seed to generate the browser long term private key b each time when the bookmark file is opened. Without the knowledge of the secret seed for b, the adversary cannot calculate the session key for the HMQV protocol, and the browser cannot decrypt the communication from the mobile device to the browser.

Offline PIN/fingerprint attacks

Though it is generally hard for the adversary to break the PIN/fingerprint protection on a stolen phone, we may still use other protection mechanisms to defeat off-line attacks. For example, Wang[10] has designed several smart card or memory stick authentication protocols for defeating offline dictionary attacks. These protocols could be used to protect the password vault in the mobile devices also.

HMQV protocol

In our protocol, we used HMQV key agreement protocol to generate the session key for the user credential encryption. In particular, both the browser and the mobile device need to contribute to the key agreement protocol. This helps to reduce the risk of insecure weak randomness generators implemented either in the mobile device or in the browser.

CONCLUSION

QR code-based authentication is to stay for quite a while, but its dependency on third-party entities is a concern for users. A zero-trust on third-party coupled with more users control could definitely establish more trust on the QR based authentication protocol. We conclude by pointing out some future research to address the limitations of our protocol. First, the user credentials can be stolen if the browser is controlled by malicious scripts. Since our approach cannot withstand man-in-the-browser attacks in this situation, further research is needed to address this limitation. Secondly, our protocol is applied on Desktop computers, more research is required in order to know how to extend this to other platforms such as mobile browsers.

REFERENCES

1. E. Grosse and M. Upadhyay, "Authentication at Scale," IEEE Security & Privacy, 11(1):15-22, 2013.
2. B. Borchert and M. Gunther, "Indirect NFC-login," http://www.ekaay.com/press/Pressekopien/NFC-Login.pdf, ICITST, 2013, pp. 204-209.

3. K. Reinhardt and B. Borchert, “Method and Computer Program Product for Providing Authorized Access to Online Accounts,” 2011, WO/2011/069492.
4. TiQR. https://tiqr.org.
5. MyDigiPass. https://www.mydigipass.com.
6. S. Bugiel, A. Dmitrienko, K. Kostiainen, A. Sadeghi, and M. Winandy, “TruWalletM: Secure Web Authentication on Mobile Platforms,” Trusted Systems, Springer 2011, pp. 219-236.
7. K. Khan, and Q. Malluhi, “Trust in Cloud Services: Providing more controls to clients,” IEEE Computer, Vol. 46(7):94-96, 2013.
8. SECG SEC. Sec2: Recommended elliptic curve domain parameters. http://www.secg.org, 2000.
9. H. Krawczyk. HMQV: A High-Performance Secure Diffie-Hellman Protocol. Advances in Cryptology–CRYPTO 2005, LNCS 3621, pp. 546–566.
10. Y. Wang, “Password Protected Smart Card and Memory Stick Authentication against Off-line Dictionary Attacks,” In SEC IFIP AICT 376, 2012, pp. 489–500.

Yongge Wang is a Professor in the department of Software and Information Systems at the University of North Carolina at Charlotte. He is the inventor of IEEE P1363 standards SRP5 and WANG-KE, and has contributed significantly to the mathematical randomness. He has also invented a distance based statistical testing technique to improve NIST SP800-22 testing in randomness tests. He is known for the invention of the quantum resistant random linear code based encryption scheme RLCE.

Khaled M. Khan is an Associate Professor in the department of Computer Science and Engineering at Qatar University. He received his B.S and M.S in computer science from the Norwegian University of Science and Technology. His Ph.D. is from Monash University. He has published more than 100 refereed papers. He is the Editor-in-Chief Emeritus of the International Journal of Secure Software Engineering.