University of Glasgow | School of Computing Science

Using mobile devices' hardware-backed keystore for universal authentication

Akos Szente (2094613s)

April 22, 2018

ABSTRACT

Mobile phones have gone through significant advancements in the past ten years. Many current smartphones are capable of advanced asymmetrical cryptography operations, assisted by their security co-processor's hardware-backed key store.

Despite the advancements, there is currently no straighforward way for developers to integrate with native electronic signature capabilities, and to use device possession as an authentication factor. Services often resort to using methods that are outdated and do not offer an adequate balance of security and usability.

This paper introduces a lightweight authentication scheme that uses hardware-backed asymmetrical cryptography capabilities of iOS devices for electronic signatures. These signatures serve as evidence of authenticated device possession, and effectively confirm legitimate user identity.

The scheme was designed to couple strong security with high usability. We measure the scheme's usability via a user study, and confirm that it is more usable than current popular two-factor authentication solutions while providing increased security.

1. INTRODUCTION

As **computer usage shifts** from traditional workstations to portable devices and ubiquitous computing, passwords are becoming less convenient and start to make less sense. Frequently entering something complex that the user *knows* is no longer required on mobile devices — yet we are still using them at online services.

An anecdote illustrates the point very well. A few months ago, my grandfather called me to ask me what his Netflix password was, because his Apple TV was asking for it. My grandfather didn't even know he had a Netflix password. I told him the password, which he first tried to spell using the remote's microphone. It didn't work because voice recognition systems are not trained for such tasks. He then tried to enter the password using the remote's clumsy trackpad, but the system did not accept the password. He probably missed a letter. As a consequence, he could not access the service he had paid for.

2. BACKGROUND

2.1 Weaknesses of Passwords

Not only is the concept of passwords inconvenient, but it is also fairly insecure. A major issue with passwords is that they are **not ephemeral**. Instead of being changed with every transaction, the same password is capable of signing **every future request** besides the current request.

Attackers can also exploit leaked passwords to gain access to other accounts of the same user. While studying cross-site password security, Das et al. observed [13] that over 40% of users directly re-use passwords between sites. When participants were asked how they choose the password for a new e-mail account, 51% said that they would reuse an existing password and 26% said that they would modify an existing password.

Moreover, there is no automatic way to inform users about password leaks and universally revoke the leaked password. Such features are commonplace in the public key infrastructure, effectively preventing leaked private keys from being used once the leak has been discovered and reported [26].

In *Users Are Not The Enemy* [1], Adams and Sasse discuss how **password requirements** form users' password habits. More importantly, they observe that users have a very poor understanding of **what makes passwords secure**, and how passwords are cracked. They note that fifty percent of respondents **wrote down** their password somewhere to avoid forgetting. Many of them were forced to periodically change their password, or had to abide by so many complex rules that they would not have been able to remember it otherwise. A number of users linked their passwords to some **common element** to make sure they could remember them all. If an attacker were to figure out one of these passwords, figuring out the common root would be easy. Moreover, even though participants usually used this method to avoid forgetting passwords, it actually had the opposite effect: similarity lead them to **forget** which password belonged to which service.

Passwords are also vulnerable to social engineering attacks, such as phishing. In a 2006 user study, Dhamija and Tygar [15] found that popup warnings and browsing cues were not effective enough: good phishing websites fooled 90% of participants. While browsers have advanced since then, so have phishing techniques. Advanced methods, such as the homograph attack against internationalised domain names [30], may even fool experts.

2.2 Digital Signatures

The concept of digital signature was first proposed by Diffie and Hellmann in 1976 [16]. The FIPS Digital Signature Standard (DSS) [20] defines digital signatures as a technology to detect unauthorised modifications to data and to authenticate the identity of a signatory. The recipient of signed data can use a digital signature as evidence in demonstrating to a third party that the signature was, in fact, generated by the claimed signatory.

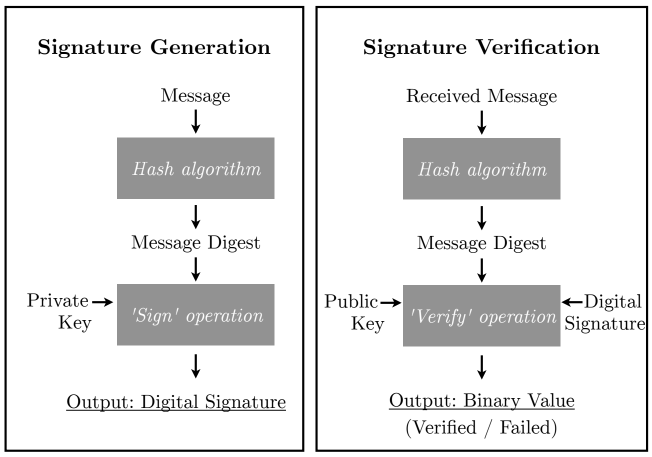


Figure 1: The two main workflows of the Elliptic Curve Digital Signature Algorithm [20]

The DSS outlines the following requirements for digital signatures:

- Each signatory owns a key pair consisting of a private and a public key. These are required for signature generation and verification. While public keys may be known by the public, private keys are kept secret.
- A hash function is used in the signature generation process to obtain a condensed, fixed-length version of the data to be signed. This is called a message digest.
- Signature generation is based on mathematical trapdoor functions, where it is easy to perform the function in one direction, but difficult to find the inverse without having access to special information.
- The digital signature is provided to the intended verifier along with the signed data. The verifying entity verifies the signature by using the public key and the same hash function that was used to generate the signature.

Figure 1 depicts how these components connect to generate and verify signatures.

Digital signatures provide the cryptographic security of many everyday services — from contract management and timestamping to software distribution and EMV (Chip & PIN) card transactions. Workflows where end-users meet digital signatures often operate without their being aware of the underlying technology. One of the reasons behind this is that users find it hard to understand the concept of public and private keys — as Whitten and Tygar conclude in *Why Johnny Can't Encrypt* via a case study of PGP [29].

2.2.1 PGP and its Usability

A popular implementation of public key authentication is PGP, where it is the user's responsibility to handle the key pair. The private and public keys are stored on the user's computer, and the mail client uses those keys to generate a signature or encrypt e-mails. Users are expected to publish their public keys so that other users can verify their signature and decrypt their e-mails.

PGP's usability has been extensively studied. In *Why Johnny Can't Encrypt* [29], the authors conclude that usability requirements of security differ from standard user interface requirements, and define the following priorities for usable security:

Security software is usable if the people who are expected to use it:

- "Are reliably made aware of the security tasks they need to perform."
- "Are able to figure out how to successfully perform those tasks. "
- "Don't make dangerous errors."
- "Are sufficiently comfortable with the interface to continue using it."

Authors also describe five problematic properties of security.

The **unmotivated user property** suggests that security is usually a secondary goal: users do not use devices for security's sake, but to perform a set of other tasks [29]. Security is expected to protect them while performing these tasks, without getting in the way and requiring extra steps and management. Users are not motivated in security, and usually just assume that security works. Users must not be expected to learn about security and look for security controls to enable security.

The **abstraction property** describes that computer programmers should take into account that the concept of regulating access and managing rules may be alien to users, and many may find it unintuitive [29]. PGP also used a variety of bad metaphors, such as a quill to represent signatures, but after clicking on the quill the software requested keys, confusing users.

The **lack of feedback property** highlights that providing good feedback is hard, because security configuration states are often complex and cannot be adequately summarised [29]. Furthermore, the study notes that the only good configuration is the one that the user actually wants, and since the software only knows what the user tells them, it is hard to prevent configuration errors. For example, copying their private key onto another computer may be intentional, or it may also be a fatal accident — there is no way for the software to tell.

The **barn door property** explains that once a secret has been left unprotected, there is no way of telling whether it has been compromised or not [29]. Security interfaces should prevent users from making mistakes and ensure that they understand processes well enough. The user study found that around a quarter of users have accidentally sent their PGP private key because of the confusing process and the unrelatable concept of differing private and public keys. Hence PGP did not adequately ensure that the barn door was always locked.

Finally, the **weakest link property** says that since fatal mistakes can happen in any stage, a security process is only as usable as its least usable component [29]. Users must be guided through the process and not be left exploring the client's security intuitively, as they would explore a word processor where actions can always be undone.

The study concludes that PGP does not even come close to achieving the authors' usability standard, and that it does not make public key encryption of electronic mails manageable for average computer users.

PGP still has not been widely replaced with a better solution for e-mail encryption and signature. The study was repeated by Ruoti et al. [25] in 2015, and concluded that modern PGP tools are still unusable for the masses.

The scheme proposed in this paper was evaluated based on the five problematic properties in Section 5.1.

2.2.2 *Compartmentalising the private key*

While the authors of "Why Johnny Can't Encrypt" only considered a scenario when both the public and private keys were residing on the user's computer, it is a common practice to move the private key to a different device, such as a smart card or a USB dongle. This has a number of advantages:

- It makes the concept of private and public keys easier to relate to, because the private key becomes an actual key instead of being a file.
- Since the user does not have direct access to the private key, it is significantly harder to disclose the key by mistake.
- External keys can be manufactured in a way that the private key never leaves the device. This ensures that the key cannot be replicated, and this makes it a better candidate for authentication by possession.

There are several standards that use digital signature technology for authentication. A popular standard is the Universal 2nd Factor (U2F) Specification by the FIDO Alliance [28]. Originally created by Google and Yubico, it was designed to log in to online services via an external secondary factor device, with native support from web browsers. After initial enrolment, the online service passes a challenge to the user's web browser at each login. The user may answer the challenge by pressing a button on a USB key or by tapping a device over NFC.

A newer standard by the FIDO Alliance is the Universal Authenticator Framework (UAF) [23]. It was designed to optionally replace passwords by saving a private key on a mobile device, then authenticate at each future login with said key.

One concern with the UAF standard is that it aims to be universally compatible with as many devices as possible, instead of limiting this facility to devices that provide a stringent enough level of security. The scheme's universality adds significant complexity, and brings several security drawbacks. For example, implementations are allowed to store the private key in a retrievable manner. Furthermore, there is no free reference implementation available.

2.3 Asynchronous Cryptography on Mobile Devices

Leading modern smartphones are equipped with a trusted execution environment [18]. This is an area dedicated to store, process and verify sensitive data, and to ensure that data integrity and confidentiality is maintained even if the rest of the device is compromised.

Secure Enclave is the proprietary security co-processor of iOS devices and Mac computers [7]. It works in tight integration with the main operating system, and performs real-time encryption, authentication and other security-related tasks. It runs a variant of L4 microkernel called SEPOS, has its own encrypted memory space, and communicates via a highly limited, messaging-based interface with the rest of the device.

iOS itself uses Secure Enclave extensively to ensure security. For example, disk encryption keys are derived from the passcode and stored within Secure Enclave [7]. Whenever the user enters their password, the raw entry is forwarded to Secure Enclave, and the validation happens within. Secure Enclave checks passcodes, and only unlocks encrypted files when the correct passcode is received. Furthermore, it disposes of encryption keys after a number of attempts, essentially wiping the disk.

The reduced attack surface of Secure Enclave, and its decoupled operation ensure that data remains protected and inaccessible even if an attacker gains access to iOS, and that the data on iPhones cannot be easily cracked by brute force.

One feature of Secure Enclave is the generation of pairs of public and private keys within. The public key is returned to the main OS, but the private key is retained within. The main OS may only pass message digests to Secure Enclave along with instructions to sign, without having the ability to extract the key. Secure Enclave can optionally be configured to require various levels of local authentication at signing, such as a passcode or biometric [5].

2.4 Biometrics and Passcodes

Apple introduced support for fingerprint reading in 2013. While the iPhone 5s was not the first mobile phone with a fingerprint scanner [11], it was arguably the first phone that made mobile biometric authentication widely popular and passcode security convenient. Apple have revealed that while 49% of users used a passcode before the introduction of Touch ID, this number has since increased to 89% [3]. They attribute this to the seamless integration of secure hardware and software that has turned authentication from a chore to an invisible step that works reliably and immediately makes sense. The reason Touch ID is so successful, and why it was able to bring biometric authentication to the masses, is arguably its usability and accessibility.

The technology operates alongside the passcode. Whenever authentication is necessary, users may either enter the passcode or touch the home button with a previously enrolled finger [7]. This lets users overcome the inconvenience of entering the passcode, and also makes using a longer, more complex passcode more practical, because users do not have to enter the passcode as frequently.

DeLuca et al. [14] argued that a well-designed biometric authentication system can have a positive effect on the protection of the users' smartphone data, and found that Touch ID was considered more usable than the alternatives (e.g. regular passcodes). It seemed more smoothly integrated into the interaction process and did not add much overhead (or even none) to the overall unlocking process. When asked to elaborate, some participants called it 'fun', 'joyful' and 'interesting', and they hardly ever mentioned trust issues or privacy.

3. PROPOSED SCHEME

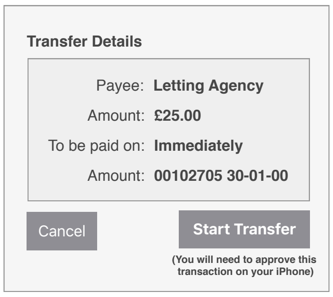
3.1 Overview

The proposed scheme offers remote identity verification on iOS via asynchronous cryptography by utilising native security features of iOS and Secure Enclave.

The application generates a keypair in Secure Enclave, and retains the public key and a reference to the private key. The public key is shared with third party services during an enrolment procedure. Such services may then send authentication challenges to the device, which are processed and presented by the application. Challenges can be digitally signed with the private key through an intuitive user interface after local identity verification. The signature is returned to the third-party service, where it is verified with the public key.

3.2 Sample User Journey for Signing a Challenge

Online money transfers often require authentication with a secondary factor. To give a better understanding of the scheme, we introduce its capabilities with wireframes for this scenario. On Figures 2-5, a user initiates an online money transfer, and signs the transaction with their mobile phone.

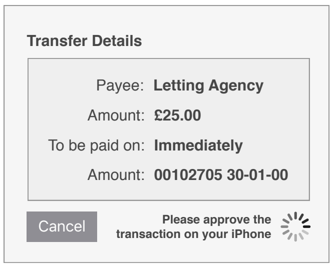


Transfer Details

Payee: Letting Agency
Amount: £25.00
To be paid on: Immediately
Amount: 00102705 30-01-00

Cancel
Start Transfer
(You will need to approve this transaction on your iPhone)

Figure 2: Preparations. The user enters transaction details, such as amount and recipient, on the bank's website, and initiates the transaction.



Transfer Details

Payee: Letting Agency
Amount: £25.00
To be paid on: Immediately
Amount: 00102705 30-01-00

Cancel
Please approve the transaction on your iPhone

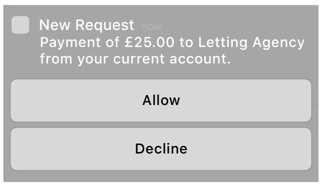
Figure 3: The bank's website instructs the user to confirm the transaction on their mobile phone.

3.3 Messaging

The scheme uses textual messaging to receive and send challenges and responses. Messages contain all information in key-value pairs, and are cryptographically signed to ensure authenticity and prevent alteration in transit.

3.3.1 Authentication challenges

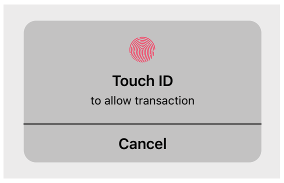
To request signature from a device, an authentication challenge is sent by third-party services. Such messages have the following parameters:



New Request now
Payment of £25.00 to Letting Agency from your current account.

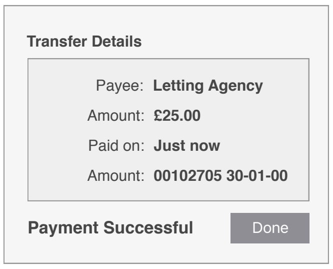
Allow
Decline

Figure 4: Meanwhile, a confirmation message is delivered to the user's phone, along with relevant transaction details. 'Allow' prompts the user to authenticate before telling the bank to let the transaction go through.



Touch ID
to allow transaction
Cancel

Figure 5: The user is prompted to authenticate themselves on the phone.



Transfer Details

Payee: Letting Agency
Amount: £25.00
Paid on: Just now
Amount: 00102705 30-01-00

Payment Successful
Done

Figure 6: After successful authentication, a reply is sent to the bank and the transaction goes through.

- **message_id** — Identifier of the authentication challenge, provided by the third-party service. Internally unique within service.
  E.g. "523452"
- **subtitle** — Designed to contain the name of the third-party service, and remain static among all messages from the same service.
  E.g. "Purple Online Banking"
- **short_title** — A short summary of *body.*
  E.g. "Login Attempt"
- **body** — An explanation of the message, potentially including transaction details.
  E.g. "Someone is trying to log in to your Purple Online Banking account 'push' from Glasgow, United Kingdom at 23/02/2018 07:02:23. Is this you?"
- **expiry** — Unix Epoch timestamp of when the authentication challenge expires.
  E.g. "1519387343"
- **nonce** — To prevent replaying signatures, there is a cryptographic nonce added to each authentication

challenge. This is a blob of random data.
E.g. "b5bc3bb46e940ce73591b2f180cc28c b29e22cbd211895a22e0aef615bf71c1212"

- **category** — Represents message type. Messages are currently either in the *challengecategory* or the *enrolmentcategory*.
E.g. "challengecategory"

- **response_url** — The URL where the client is expected to send the signature.
E.g. "https://s02.szente.info/authreq-srv/callback.php"

3.3.2 *Reply*

These messages are sent by the client to respond to authentication challenges with signatures. A reply without a prior authentication challenge is invalid.

Replies are only sent when the user allows the challenge to proceed. Third-party services are not notified about denials.

- **message_id** — Internally unique identifier of the authentication challenge, as previously provided by the third-party service.
E.g. "523452"

- **signature** — Signature in hexadecimal encoding, generated by the client using the private key. Signatures are discussed in a later section.
E.g. "30460221009c53817d9222017713f340c5e539213f 9add2bb0784928a866c410eaa9fd4b46022100ed9ff9f796f 89f61a7bbf8ef2b64b8079dca203fb4be0a15165e39602c4a 3d39"

- **publickey** — Public key of the client's keypair. (The corresponding private key was used to create the signature.) X.509 DER-encoded certificate, in PEM format.
*E.g. "-----BEGIN PUBLIC KEY----- MFkwEwYHKoZ Izj0CAQYIKoZIzj0DAQcDQgAEeXGgRqcDZBCXQ96 LWVVq2cxg0+Hx\nP8NxGIllbp24Txxa62b/dXBc++i I3JWQOZbUa2lxyxHTwcxvjzRm4eblcQ==
----- END PUBLIC KEY----- "*

3.4 Enrolment

To perform enrolment, the server component generates a special authentication challenge that the user must sign, and sets the category to `enrolmentcategory`.

As presented in 3.3.2, replies always contain the corresponding device's public key. This provides an opportunity for the third-party service to save the public key.

When the server component receives the reply to the enrolment message, it saves the device's public key instead of validating the signature, and validates all future replies' signatures using the acquired public key.

3.5 Signatures

Clients create the signature by hashing certain fields of the original authentication challenge, then signing the hash with their private key. Third-party services validate the signature with the client's public key.

3.5.1 *Signature algorithm and keys*

The scheme uses the Elliptic Curve Digital Signature Algorithm (X9.62) to sign authentication challenges. The ECDSA is based on the computational intractability of the discrete logarithm problem (DLP) in prime-order subgroups of $\mathbb{Z}_p^*$ [21]. It is an ANSI, ISO, IEEE and NIST standard, recommended by the United States National Security Agency to protect data up to `TOP SECRET` classification [19]. ECDSA is widely used in digital signature based technologies, such as in blockchain-based cryptocurrencies [2].

The scheme utilises 256-bit ECDSA keys (`secp256r1`), as defined in Section 5.1.1 of RFC 4492[9]. Although the NSA now recommends to moving to larger key sizes (e.g. `secp384r1`) while awaiting quantum-safe protocols [19], P-256 is still widely used — and it is the only cipher suite supported by Secure Enclave as of iOS 11.2.6 [7].

The public key is encoded in standard X.509 DER for interoperability.

3.5.2 *Representing challenges before creating message digest*

Authentication challenges contain several data fields — as introduced in 3.3.1. These data fields carry all information that the third-party service can pass to the user via the scheme. As we will later demonstrate in Section 3.5.4 efficient signature must be based on all of these fields.

Bencode is a simple textual data representation format introduced by the peer-to-peer distribution technology BitTorrent. It was designed to provide bijection between values and their encodings — making sure that for each data set there is only one valid Bencode representation, and that two different Bencode strings can never mean the same data set.

To create a canonical summarised representation of the key-value pairs, the scheme uses Bencode. The sample authentication challenge of 3.3.1 is converted to the following representation:

```
d4:body138:Someone is trying to log in to your Pu
rple Online Banking account 'push' from Glasgow, Un
ited Kingdom at 23/02/2018 07:02:23. Is this you?8:
category17:challengecategory6:expiryi1519387343e10:
message_idi523452e5:nonce64:b5bc3bb46e940ce73591b2f
180cc28cb29e22cbd211895a22e0aef615bf71c1212:respons
e_url48:https://s02.szente.info/authreq-srv/callbac
k.php11:short_title13:Login Attempt8:subtitle21:Pur
ple Online Banking5:title26:New requeste
```

3.5.3 *Message digest*

According to the FIPS DSS, "an approved hash function, as specified in FIPS 180, shall be used during the generation of digital signatures" [20]. One of such hash functions is SHA-384, which is also recommended by the United States National Security Agency to protect data up to `TOP SECRET` classification [19].

Message digest is created by calculating the SHA-384 hash of the authentication challenge, using its canonical representation.

The message digest in hexadecimal representation for the authentication challenge in Section 3.5.2 is `71f182c099317c 4f86ecae7edc315881bb8f47a45f682d8a1f7d6cc51531573f4 295d984756ae7e92dbb7d220bbdc932`.

3.5.4 *What you see is what you sign*

The term "What You See Is What You Sign" was proposed by Landrock and Pedersen [22]. While traditional signatures generally certify printed sentences that are directly legible and understandable by humans, digital signatures certify binary data that cannot be interpreted by humans in the original format. Binary data must go through a high level semantic interpretation to convert them into a form that humans understand so that they can review and sign such documents. However, the same string of bits can be interpreted in multiple ways, and data that one has signed could potentially mean something else in a different presentation. The authors propose that digital signature schemes must ensure that they leave no way to alter the semantic interpretation of the original message, either by accident or intent, so that humans can be certain that *what they see is what they sign*.

The scheme was designed to provide such traits. For each challenge request there is only one valid presentation, and fields that carry relevant content (subtitle, short_title, body) are always visible when issuing signatures. Users can be reasonably expected to have seen each of these fields when they issue signatures.

The iOS keychain model ensures that the private key remains accessible only for the corresponding app, and that other applications cannot perform operations. The app binary is protected by code signature which is verified by iOS at every startup, ensuring that third-party code cannot hijack the key and sign arbitrary challenges, even with physical access to the device [7].

3.6 Message Delivery

Three messages are passed between client and server during authentication:

1. Authentication challenge (server → client)
2. Reply with signature (client → server)
3. Acknowledgement (server → client)

Acknowledgements are optional, as there is no way to revoke signatures once creating them. The signature's validity does not depend on the delivery of an acknowledgement.

The current implementation utilises three ways for delivery of authentication challenges:

- **Push notifications** — Authentication challenges arrive via push notifications over the air. Users are automatically notified of pending requests.
- **QR codes** — The built-in camera application of iOS is capable of reading QR codes. We encode challenge data in a special QR code that the user may read with their mobile phone.
- **URL schemes** — Special links may be added to websites that open the application and import an authentication challenge automatically. (It is required to open said website on the enrolled device, otherwise the link does not work.)

Replies and acknowledgements happen over HTTP in TLS. Implemented messaging is further discussed in Section 4.

Messages could potentially be passed in any other way between client and server, as long as the correct Bencode representation can be reconstructed after delivery. Possible ways of extension are discussed in Section 6.5.

4. IMPLEMENTATION

The delivered prototype is called Authreq, and consists of the following four main components:

- **iOS client** — where users can receive authentication requests and send replies with signatures.
- **Server** — which receives replies and keeps track of pending requests.
- **Service SDK** — which allows third parties to integrate with the scheme and create authentication challenges in their PHP applications.
- **Reference Site** — which imitates an online banking site where login attempts and transactions must be approved via the iOS client.

Figure 11 depicts potential channels of interaction between implemented components. Figure 16 in the Appendix documents interaction over time for a complete user journey of authenticating for a privileged action. A complete user journey requires all four components to participate: the iOS client, the server, an external service, and the service SDK. Possible channels of communication will be presented in detail in subsections of specific components.

4.1 iOS Client

The iOS Client has the following main responsibilities:

- Generating and managing public and private keys
- Enrolling with service providers
- Receiving and processing authentication challenges
- Signing authentication challenges

The application was entirely written in Swift 4, and requires iOS 11. (All devices that support iOS 11 include Secure Enclave.)

Cryptography operations are only performed via public APIs of iOS. These are covered by Apple's regular security updates [7].

4.1.1 Key generation

Upon startup, the application checks the iOS Keychain for an existing authentication keypair. If no such keys exist, it instructs Secure Enclave to generate a new keypair, and adds the public key to the keychain.

To generate the keypair, the `SecKeyGeneratePair` function is used from the Apple Security Framework [6].

We set the following keys and values in the `parameters` dictionary:

- `kSecAttrTokenID = kSecAttrTokenIDSecureEnclave`
  Specifies an item should be stored in Secure Enclave.
- `kSecAttrKeySizeInBits = 256`
  Indicates the number of bits in the cryptographic key.
- `kSecAttrKeyType = kSecAttrKeyTypeECSECPrime-Random` *Elliptic curve algorithm.*

For the private key, we ensure that a reference remains in the device's keychain by setting `kSecAttrIsPermanent` to `true`, mark the possibility for future authentication prompts by setting `kSecUseAuthenticationUI` to `.allow`, add the access control flag `.privateKeyUsage` to instruct Secure Enclave to let the key be used for signatures, and add the `kSecAttrAccessibleWhenUnlockedThisDeviceOnly` access flag to ensure that the key does not migrate between devices and that the device must be unlocked before access.

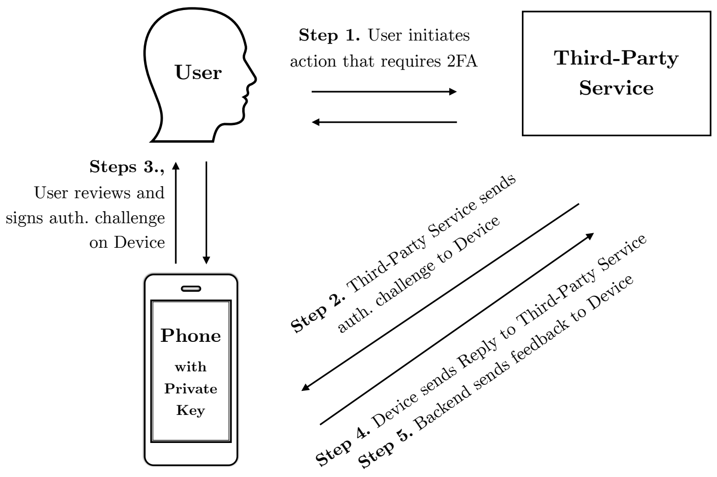
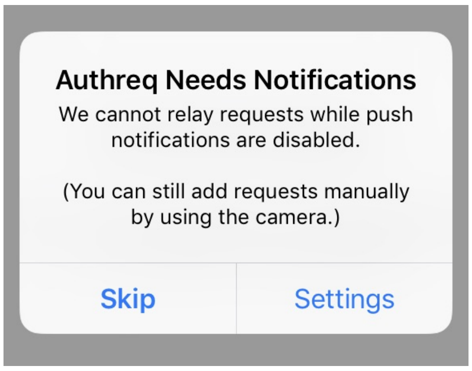

User
Step 1. User initiates action that requires 2FA
Third-Party Service
Steps 3., User reviews and signs auth. challenge on Device
Phone with Private Key
Step 2. Third-Party Service sends auth. challenge to Device
Step 4. Device sends Reply to Third-Party Service
Step 5. Backend sends feedback to Device

Figure 7: Communication from the user's perspective

4.1.2 Communication

The implemented client receives authentication challenges over the air via the Apple Push Notification Service (APNs).



Authreq Needs Notifications

We cannot relay requests while push notifications are disabled.

(You can still add requests manually by using the camera.)

Skip | Settings

Figure 8: iOS client prompting the user to enable push notifications

Upon first startup, the application asks the user to enable push notifications. If push notifications are disabled, the application regularly prompts the user to reconsider their decision — as presented on Figure 8.

APNs uses a JSON-based messaging format for notifications [4]. This is a sample push notification, containing the same data that the request in Section 3.3.1 did.

```
{"aps" : {
 "category": challengecategory,
 "alert": {
    "body" : "Someone is trying to log in
        to your Purple Online Banking
        account 'push' from Glasgow,
        United Kingdom at 23/02/2018
        07:02:23. Is this you?";
    "subtitle" : "Purple Online Banking";
    title : "New Request"
 },
 "badge": 1,
 "sound": default,
 "content-available": 1,
 "additional_data": {
    expiry : 1519387343;
    "message_id" : 14;
    "nonce" : "
        b5bc3bb46e940ce73591b2f180cc28cb29
         e22cbd211895a22e0aef615bf71c12";
    "response_url" : "https://s02.szente.
        info/authreq-srv/callback.php";
    "short_title" : "Login Attempt";
    "signature" : "fyvRhrhuAv25oqs4+Xlo15
        qpZzFymVKQ2gCPbWBNsXgsyuqQAw7pssX
        4KBbMmEV4fAj2Rzj9s4KZ4xkXVyMpfJJst
         bW2sebDlhAzGGvuAdNrbO2KD+
        gPYEgNoqX koKI4UX8BTrRQ8tNKizuzAtQ
        +x9sUiGLsZ
        v3B7RI43UWQ5ViqSXBHXY12PPW2YvCU/eV
         0
        qV428thDugvMBuGOVsRLFE5zWhreQUbEj
        nIvRUGGfOQk7k4wR7mAmXbUiJxZbWdcALK

        Ea8YrPehOnVSr7c0vNPjVZaIkCnzxes1BO
         MjatGPYBmxUTwFzOWSUEMDODAtGTwFMl
        ++ ZsLBLmjE1wYaa+Q==";
 }
}}
```

Push notifications are automatically converted to the client's internal storage format upon arrival. Signature validation and storage are discussed in detail Section 4.1.3.

APNs uses tokens to identify and address devices. Devices forward their token to the server both during enrolment and during every signature reply. This allows the server to keep the device token up-to-date and reliably deliver authentication challenges over the air.

According to Apple, device tokens may change at certain rare system events [4]. This may result in undelivered push notifications. However, by using alternative delivery methods (such as QR code, as we will later discuss in this section) requests can still be signed, and the new device token will automatically be saved on the server at the next successful signature.

Besides push notifications, the application implements a url scheme (`authreq://`) for receiving enrolment requests and authentication challenges. This provides a universal solution for addding requests via external routes.

The application expects incoming URLs to contain regular APNs dictionaries in base64 encoding.

```
authreq://eyJhcHMiOnsiYWxlcnQiOnsidGl(...)
```

The stock camera application of iOS 11 supports QR codes natively. QR codes with `authreq://` URLs inside provide an easy way to add authentication requests from other computers without having to rely on push notifications. Users can simply scan an on-screen QR code and import the encoded request with a tap — as depicted on Figure 9.

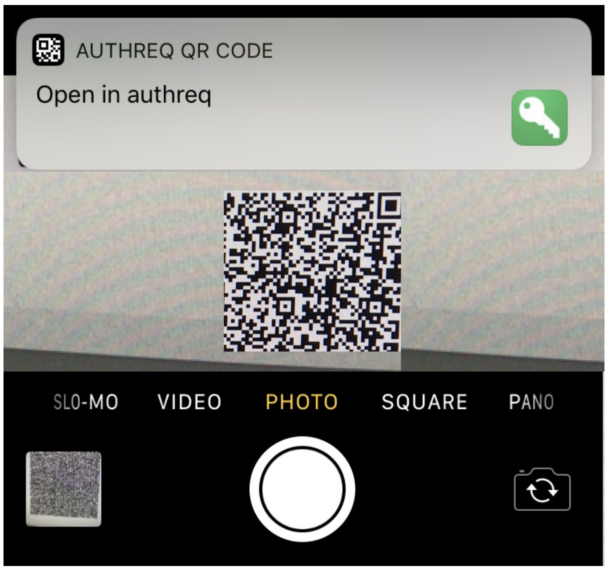


Figure 9: iOS camera reading an Authreq QR code from a desktop computer. (Image height edited)

Furthermore, the URL scheme also provides a way for third-party services to add a special button to their website. The button on Figure 10 is useful when browsing from a device that is enrolled in the scheme: it directly opens Authreq and adds the challenge.

Figure 10: 'Add to authreq' icon that imports the challenge directly when browsing from the mobile device

These methods can be used during enrolment and whenever push notifications are unreliable.

Replies are sent via secure HTTP by `URLSession`. The class can be configured to allow pending requests to continue in the background, and handles dropped connections efficiently. Messages are included within the POST request's body in JSON.

4.1.3 Signature validation and storage

The iOS client utilises the Core Data framework to ensure that authentication requests persist between sessions.

Received authentication request dictionaries are converted to `SignatureRequest` objects, which is a Core Data backed internal representation. The `SignatureRequest` constructor checks the original dictionary's integrity and authenticity by verifying the signature that the server included against the server's known public key and the dictionary's Bencode representation. Authentication requests with invalid signatures are immediately discarded.

The app's main screen displays a list of previously received `SignatureRequests`. We keep track of requests' status, and display each request as either *active*, *expired*, *allowed* or *declined*. Requests can be manually discarded at any time.

4.1.4 Native citizen on iOS

The application was designed to fully take advantage of native features of iOS, and to provide a user experience that closely follows stock applications. Since the solution is tailored to iPhone, we did not have to restrict its capabilities to the subset of features that all platforms and devices support.

The scheme uses rich notifications to deliver information. Users can fully interact with Authreq without actually opening it, because all information is available directly on the lockscreen. Notification actions appear for 3D Touch, and let users sign requests even when the app is not in the foreground. Results are also presented as notifications.

This is especially useful when dealing with challenges that were triggered by using another app. For example, when a user needs to authenticate on a page in Mobile Safari, they can do so without having to leave the browser.

Secure Enclave requires users to authenticate locally before enabling signatures with the private key. The primary method of such authentication is biometrics — i.e. Touch ID or Face ID.

To notify users about successful and unsuccessful transactions, we use stock system sounds that users can connect to their past experiences with iOS. Furthermore, we use the Taptic Engine to provide haptic feedback when authentications happen.

4.1.5 Third-party components

The following third-party components are used by the iOS client:

- **EllipticCurveKeyPair** — Wraps certain low-level C APIs of the Security Framework in Swift. *Available under the MIT license.*
- **SwiftyRSA** — Used to verify authentication challenges' digital signature. *Available under the MIT license.*
- **Piano** A wrapper of the AVFoundation and UIHapticFeedback libraries, providing easy access to system sounds and the Taptic Engine. *Available under the MIT license.*

4.2 Server

The **Server** component has the following responsibilities:

- Receiving and storing replies
- Validating signatures of incoming replies
- Providing an API where other components can check status of authentication requests

The server component was written in PHP. Since cryptography operations are performed via OpenSSL, a PHP build with OpenSSL support must be used.

4.2.1 *Callback API*

As introduced in Section 3.3.1, authentication challenges contain a callback URL. Behind this URL is the server component, which receives and validates replies. The iOS client sends its reply to this URL for the corresponding challenge. Such reply messages were introduced in Section 3.3.2.

Signatures are validated by testing them with the client's public key and the original message's Bencode representation. A `DatabaseSignature` class instance is created for each incoming signature, where `message_id` contains the corresponding authentication request's ID, `pem` contains the client's public key and `signature` contains the received signature. The `validate()` method queries the original authentication request from the databse, recreates its Bencode representation, then checks the signature by performing:

```
  openssl_verify($bencodedOriginalMessage,
hex2bin($this->signature), $this->pem, "sha384");
```

4.2.2 *IsSigned API*

The isSigned API queries the database for a specific message_id to see whether the server has received the corresponding signature from the client. Third-party applications may use this API to periodically check signature status and provide immediate feedback when the signature arrives.

4.3 Service SDK

This component allows third parties to send Authreq enrolment and authentication requests, and to check signature status from their own applications.

The `SignatureRequest` class represents authentication requests. To authorise a user, one must instantiate this class and fill out the fields from Section 3.3.1. Delivery happens via push notifications by `sendPush()`, or elsewhere using the universal URL scheme representation. We utilise `openssl_random_pseudo_bytes` to gather random data for the cryptographic nonce.

The `Signature` class represents signatures, as introduced in Section 4.2.1.

The Service SDK and the Server communicate via a shared SQL database instance. `SignatureRequest`s created via the Service SDK are accessed by the server component directly.

Internally, the server component also utilises the Service SDK.

4.3.1 *Reference Service Provider*

The reference service provider is a PHP application based on the Yii framework [17]. It simulates an online banking site, where users may log in to view account details and transfer money. User accounts are protected by password, and a secondary authentication factor which may be SMS, card reader, TOTP and Authreq.

The component was created to demonstrate capabilities of the scheme, and to provide a platform where Authreq may be compared to common secondary authentication factors.

Registration is not implemented. Accounts may be manually added by inserting new records into the `user` table.

New accounts can be linked to Authreq (i.e. enrolled) via QR code or by opening an `authreq://` hyperlink. Once linked, all logins and transactions must be approved via Authreq by signing authentication challenges.

The reference service provider uses the service SDK to create authentication requests and to check signature status, and runs alongside an Authreq server instance and a shared SQL database instance.

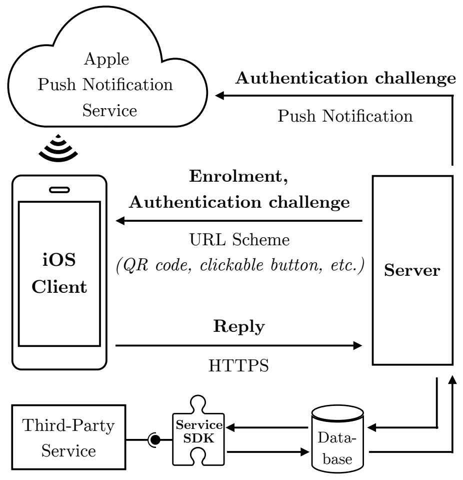


Figure 11: Communication channels

5. EVALUATION

5.1 Comparing to the Five Problematic Properties of Security

Whitten et al. introduced five problematic properties of security [29], as discussed in detail in Section 2.2.1. Providing efficient mitigations to the five properties was an important objective during the scheme's design and implementation.

- **Unmotivated user property** — The system's adoption will arguably depend on its accessibility and usability. Ensuring that users do not feel like Authreq is a burden to use was an important goal during development. We achieve this by utilising hardware that people always have on them: their mobile phone. Interacting with nofications and unlocking the device with Touch ID are both integral parts of the iOS experience, and eligible users should not have trouble adopting the technology.

  Once enabled, the technology quietly protects the user by sending notifications whenever someone tries to log in to their account. The scheme assures users that their account is always safe, without becoming a dominant part of the workflow.

- **Abstraction property** — We wrap powerful digital signature technology with a user journey that completely abstracts technical details. Instead of public and private keys, our model is based on *approving* requests for actions over a different channel. The user interface was designed to be simple and clutter-free.

- **Lack of feedback property** — The scheme's SDK encourages third-party service providers to summarise sensitive operations' details into a single push notification. This provides users with a final overview before committing actions, and also lets users notice and prevent attackers' operations. The software's configuration is practically binary: either it is *on*, or *off.* Once a device is linked to a user account, it is up to the service provider to decide when to request authentication and approval. To lower the number of potential mistakes and misunderstandings, the Authreq client does not offer any user-facing settings.

- **Barn door property** — The scheme delegates the responsibility of protecting the barn door to iOS: only those may issue signatures who are in possession of the device, and are aware of the required passcode or have the correct fingerprint. The security model of Secure Enclave ensures that the private key remains protected and that it cannot be cloned or moved. Furthermore, Apple's operating system offers sophisticated protection for devices that go out of the user's possession, such as remote wiping, automatic wiping after incorrect passcode attempts, and company passcode policies — making iOS-based public key authentication significantly more secure than traditional solutions, such as smart cards.

- **Weakest link property** — The concept of *approvals* also helps reduce the effect of third-party services' weakest links. Users can rely on a final confirmation before sensitive transactions, and gain a soft way to *undo* initiated transactions. While Authreq itself requires one final interaction that cannot be undone, it aims to make sure that the user is aware of all transaction details and that challenges are not signed by accident.

5.2 Quantifying the Quality

In *Quantifying the Quality of Web Authentication Mechanisms — A Usability Perspective*, Renaud proposes a method to measure the quality of web authentication mechanisms by a quality coefficient [24].

The author identifies the following three stages of authentication that one should consider from a convenience perspective:

- **Enrolment** — The first use of the system, when the key is assigned.

- **Authentication** — Subsequent use of the system. Since this task is performed regularly, its price will be paid repeatedly.

- **Replacement** — Necessary when the user's key is no longer valid.

The proposed scheme contains methods for *enrolment* and *authentication*, but not for *key replacement*. Furthermore, the current *enrolment* procedure does not consider *key replacement*, giving it an unfair advantage over other authentication methods. For these reasons, we will concentrate on *authentication* during the evaluation.

The quality coefficient compiles a number of factors regarding a method's accessibility, memorability, security and vulnerability to create a single score. We have measured the quality of Authreq with this method, and compared it to other popular solutions: traditional passwords, SMS one-time passwords, British banks' card reader based one-time passwords, and one-time passwords from the Google Authenticator application. The assessed quality of Authreq is depicted on Table 1, while other methods' quality are in Tables 2-5 in the Appendix.

Dim & Aspect	*Coeff.*	*Reasons*
Accessibility	*ad=0.41*	
Special Requir.	0.33	Application
Convenience	0.25	Fairly quick, but still adds another step
Inclusivity	0	Accessibility features of iOS work with app
Memorability	*md=0*	
Retrieval	0	Nothing to recall
Meaningfulness	0	N/A
Depth of proc.	0	N/A
Security	*sd=0*	
Predictability	0	Signature prediction requires private key
Abundance	0	High entropy
Disclosure	0	The private key is adequately protected from disclosure
Vulnerability	*vd=0*	
Confidentiality	0	Signature differs every time. Cannot be reused
Privacy	0	Leaks have no consequences to privacy
Breakability	0	Not prone to social engineering and research based attacks; ECDSA widely considered safe

Table 1: Quality of Authreq, measured with the scheme of Renaud

Following the author's method, we have calculated the deficiency for each of the mechanisms as follows (accounting for the specialities of web authentication with modifying factors): $\overline{eq_{web}} = 13 - (ad * 1.5 + md + sd * 1.5 + vd)$

In *Convenience*, instead of considering time taken as a binary per each stage, we considered the time that a single authentication takes (0 for very short, 1.0 for very long). For authentication methods that require a mobile phone, we have assumed that users own the required devices, and that they know how to operate the device and the accessibility options that they possibly need.

Results were as follows:

- **Passwords:** 8.71
- **SMS one-time passwords:** 10.71
- **Card reader:** 9.575
- **Google Authenticator:** 10.76
- **Authreq:** 12.385

5.3 User Study

5.3.1 Background

Cristofaro et al. compared the usability of two-factor authentication mechanisms in 2014 [12].

First, the authors conducted interviews to identify popular technologies, contexts and motivations in which these technologies are used. Users were from a wide range of ages, genders and educational background; and were interviewed in one-on-one meetings.

Authors found that the most used 2FA technologies were codes generated by a security token, codes received via SMS or e-mail, and codes generated by an app. They identified three primary contexts: work, personal and finance. Participants used 2FA either because they were forced to, they wanted to, or because they received an incentive.

They then conducted a quantitative study to measure the usability of three popular 2FA solutions: security tokens, one-time passwords via e-mail or SMS, and dedicated smartphone apps like Google Authenticator. An online survey has been completed by 219 users, where participants had to answer a set of questions on a Likert-scale, and a few other open-ended questions.

The paper's authors conclude that user perception of the usability of two-factor authentication mechanisms is often correlated with their individual characteristics (e.g. age, gender, background), rather than with the actual technology or the context/motivation in which the technology is used. They find that the four mechanisms were all perceived as highly usable, with little difference among them. Users' perception of trustworthiness was not negatively correlated with ease of use and required cognitive efforts, and deduct that technologies that are perceived as more trustworthy are not necessarily less usable.

5.3.2 *Methodology*

Building on the methods of Cristofaro et al., we have conducted a user study to compare and measure the usability of the following authentication methods:

- SMS one-time passwords
- Card reader based one-time passwords
- One-time passwords from Google Authenticator
- Authreq

We recruited 30 participants and organised one-on-one meetings that took approximately 30 minutes. The survey consisted of the following three stages:

Stage 1: We gave participants a short questionnaire about demographics and their prior experience with computers in general and with multi-factor authentication. Participants were asked to explain the concept of multi-factor authentication with their own words, to list mechanisms that they have used before (along with corresponding scenarios), and to explain reasons why they enabled the technology.

Stage 2: We gave participants a laptop and a mobile phone, and introduced them to the reference service provider for Authreq (introduced in Section 4.3.1). Participants had to perform a set of simple tasks under each of the studied 2FA mechanisms: **Task 1.** *Log in to Purple Online Banking with the given user name and password.* **Task 2.** *Make a payment of 30 GBP to David Gray from your Current Account.* Participants were asked to indicate the perceived difficulty of the tasks under each scenario immediately after.

Stage 3: Participants were asked to fill out a 1-page questionnaire for each studied 2FA technology. Besides 13 Likert-scale questions, the questionnaire contained two open-ended questions where participants were asked to share what they liked and disliked about each technology. Finally, participants asked to select the authentication method that they perceived as the most convenient, most secure, and most intuitive.

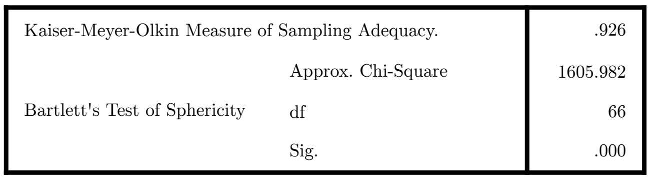
Five participants were asked to mark Authreq on the 10-item questionnaire of the System Usability Scale [10], which is a quick and fairly accurate measure of usability according to previous research [8]. Cristofaro et al. have argued that two-factor authentication technologies rely on a unique combination of hardware and software that SUS may fail to capture — which is why we did not extend this test to all authentication methods and participants.

We recorded screen contents and other inputs from participating devices, and measured the time of each authentication. Before each scenario, participants were introduced to the evaluated 2FA mechanism, and the conductor demonstrated its operation.

Demographics: The demographics of our participants is reported in Appendix C in detail. 64.5% of participants were male, and 19.4% of participants reported having a background in computing science. Participants were recruited from all age categories above 18, with their educational background ranging from less than high school to postgraduate. Our population's diversity is similar to Cristofaro et al.'s.

5.3.3 *Measuring usability via questionnaire*

We performed factor-analysis to convert the variables from twelve of Stage 2's Likert-scale questions into a small set of linearly uncorrelated variables. First, we have examined the presence of correlation between original variables by calculating the Kaiser-Meyer-Olkin Measure of Sampling Adequacy and performing Bartlett's Test of Sphericity. With a KMO of 0.926, we have deemed the data suitable for analysis.



Kaiser-Meyer-Olkin Measure of Sampling Adequacy.		.926
	Approx. Chi-Square	1605.982
Bartlett's Test of Sphericity	df	66
	Sig.	.000

Figure 12: KMO and Bartlett's Test

We continued by performing Principal Component Analysis (PCA) with Varimax (orthogonal) rotation. The analysis yielded two independent factors.

Communalities range from 0.418 to 0.911 — exact values are visible on Figure 21 in the Appendix. The two principal components maintain 77.6% of original information. See Figure 22 in the Appendix for details.

We labelled **Factor 1 Ease of use and trust** due to the high loadings of the following components: *happy to use again, trustworthy, convenient, helpful, enjoyable, quick, user-friendly.* **Factor 2** is labelled **Cognitive efforts** due to the high loadings of the following items: *needed instructions, had to concentrate, did not match expectations, stressful, did not enjoy using.*

We have calculated the two factors' mean values for each authentication technology, then tested the values with ANOVA to determine whether the values differed by technology. See Figure 23 and 24 in the Appendix for details.

Figure 13 presents the two factors over the four examined authentication technologies. We have found significant

Technology	*Ease of use and trust*	*Cogn. effort*
Card Reader	4.09	2.09
SMS	2.26	4.30
Google Auth.	1.90	4.07
Authreq	1.19	4.42

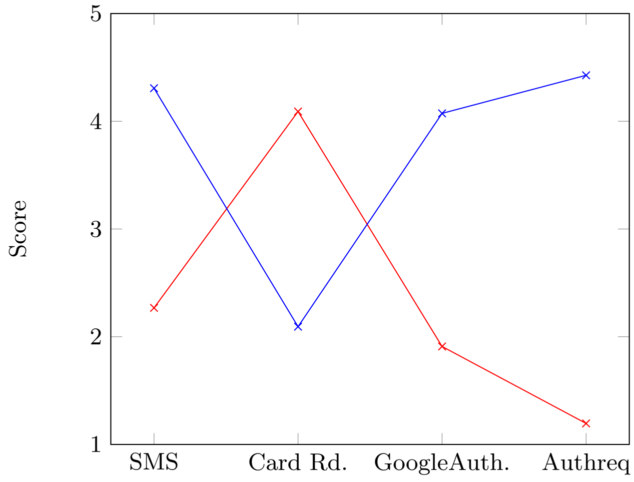


Figure 13: Ease of use and trust (lower values are better) and cognitive effort (higher values are better) for each authentication technology

Technology	*18-34*	*35-54*	*55+*	*Overall*
SMS	21.28	21.25	36.6	23.83
Card Reader	71.28	97.50	117.00	82.40
Google Authentic.	19.85	21.5	38.20	23.13
Authreq	11.23	12.5	18.60	12.63

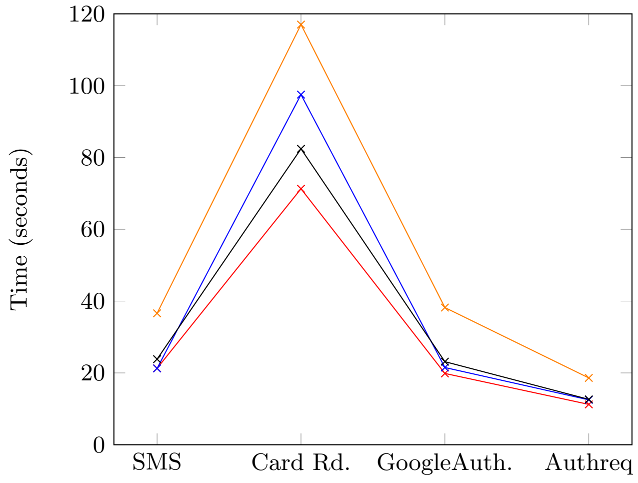


Figure 14: Average time to authenticate, by age — 18-34, 35-54, 55+, overall.

differences between the technologies' usability.

The worst scores were received by card reader, performing very poorly both in cognitive efforts and ease of use and trust. All other methods were perceived as highly usable, with scores above 4.0 (on a scale of 1 to 5) for cognitive effort (higher values are better), and under 2.5 for ease of use and trust (lower values are better).

However, Authreq was found to be significantly easier to use than other methods with an 1.19 mean score for ease of use and trust (on a scale of 1 to 5), and 4.42 for cognitive effort. Open-ended questions provide more insight in Section 5.3.5.

Converting the SUS score to a percentage scale, Authreq's overall SUS is more than 80%, which is considered "Grade A" usability. [12]

5.3.4 *Evaluating time taken to complete tasks*

Quoting the words of Whitten et al., *"people do not generally sit down at their computers wanting to manage their security; rather, they want to send e-mail, browse web pages, or download software, and they want security in place to protect them while they do those things."* The evaluated mechanisms inevitably extend the time of authentication by adding another step to the user journey. Arguably, the shorter that users have to interact with the evaluated systems, the better.

While the questionnaire provided a way to measure perceived usability, the time it has taken to complete tasks gives concrete, comparable data on the designs' efficiency and learnability.

We have found that Authreq was the quickest authentication method, taking 45% less time than Google Authenticator and SMS on average. These differences do not depend on age and computing science background.

Participants with computing science background were able to handle complex tasks quicker. For example, card reader authentication has taken an average time of 55 seconds, while non-comp-sci participants required 89 seconds on average. No such differences were observable regarding Authreq. Using Authreq took approximately 40% longer for users of Android than iPhone users, while they performed similarly at other scenarios. This suggests that familiarity with the environment is a significant factor, as Authreq uses native interactions of iOS that iPhone users are habituated to.

5.3.5 *Anaylsis of open-ended questions and conductor's notes*

90% of users found Authreq to be the most convenient solution, and 70% found it to be the most intuitive. However, most participants perceived card reader as the most secure mechanism (40%), making Authreq second (36%). Cristofaro et al. found trustworthiness not to be negatively correlated with ease of use and required cognitive efforts, and that 2FA technologies perceived as more trustworthy are not necessarily less usable. However, they noted that their results were in contrast with prior work. While our study's participants did not perceive card reader especially trustworthy, most still named it the most secure out of the four solutions when asked, some explaining that "it requires a card, a banking PIN, and pounding many buttons", and that "it must be very secure if it is so complex". Our results indicate that ease of use and required cognitive effort may be negatively correlated with perceived security.

Participants mostly praised SMS-based authentication because of its simplicity and ubiquity. Many called it the defacto standard way of 2FA, and used it as basis of comparison in open-ended questions. 85% of respondents have used the technology before, and for most respondents SMS was the only known 2FA technology. Some mentioned having experienced text messages that wouldn't ever arrive. Users disliked having to type a string manually on the computer, that the technology is somewhat slow, and that text message contents usually appear on phones without requiring authentication. On many occasion, when users received the second text message at Task 2, they did not know which text message was first and which one was for the new transaction. Furthermore, after typing the wrong code, participants did not know whether they had to try re-entering the same code, or wait for another one.

> *Authreq uses TCP/IP and APNs for messaging, which results in significantly faster arrivals. The scheme's requests cannot be approved without authenticating locally first, and it mitigates some of the usability issues of SMS by not having to type a code.*

Card reader was the least favourite, but many praised its (perceived) level of security, calling it "100% secure" and "reassuring". Two participants noted that the card requires a PIN, which is another layer of security. Almost all participants mentioned that the scheme is too complicated and time-consuming. Many mentioned that the card reader must be carried around and that it is an additional item to lose. Several participants had issues with using the card reader in the dark, or without having their glasses nearby, bringing up serious accessibility issues.

> *Authreq mitigates concerns about relying on external hardware by using a device that most participants always have on them. We fully support the accessibility features of iOS, such as accessing via screen reader and TTY. Similarly to card reader, Authreq also requires local authentication.*

Regarding Google Authenticator, participants liked that there was no waiting time compared to SMS, and that it was fully offline. However, some participants disliked having to enter a string manually, and many mentioned that they were often confused whether they could use a code once it had expired, or the same code twice within the given time frame. Many found the 30-seconds timeframe too short. We observed that many participants rather waited for the current timeframe to end to ensure that they would not run out of time. Four participants mistyped the code, on two of those occasions because they didn't realise that the code changed mid-entry.

> *Authreq does not use codes, and for each request we send a new challenge which we clearly indicate. However, the current implementation does not support fully offline operation.*

Authreq was described by many as "intuitive" and "really fast". Many liked that it involves fingerprint scanning, and that it can be operated from the lockscreen without having to open. One participant compared the scheme to Apple Pay, and called the similarities reassuring. Two participant were rightfully afraid that the scheme would only work on iPhone, and one was not used to using their fingerprint to authenticate, and called it weird.

6. CONCLUSION AND FUTURE WORK

In this paper, we have introduced a novel authentication mechanism, and provided a reference implementation that focused on usability. We confirmed Authreq's usability through a user study where we compared it to popular two-factor authentication solutions. However, the proposed scheme has the potential to authenticate users under several other scenarios. We will now discuss further scenarios, some of the scheme's limitations, and possible means of extension.

6.1 Other Potential Scenarios

The ECDSA-based digital signature technology provides sound base for replacing passwords altogether. While it is not a scenario that we investigated during our user study, it is within the scheme's capabilities, and it is why we started exploring the technology. Such use will require deeper analysis of user behaviour, and development of a proper key replacement procedure. The scheme will also require a way to protect against mass login attempts (and unwanted notifications). One possibility is coupling the proposed scheme with a simple PIN that services would ask for when logging in from a new location. This would keep the advantages of Authreq, while still mitigating the issues presented in Section 1 and 2.1.

3-D Secure is an online payment security protocol that protects cardholders from unauthorised charges by validating their identity during online transactions [27]. Banks could potentially integrate the scheme with 3-D Secure and send notifications about online card transactions before they happen, prompting users to approve the transaction.

The scheme could also assist users in approving authentication challenges that do not originate from the web browser. For example, it could be integrated with EMV (Chip & Pin) card infrastructure to send authentication challenges to users about suspicious transactions in real time. This could potentially help reduce card fraud.

Another financial example is allowing customers to withdraw cash without being in possession of the card. Users could scan a QR code (i.e. an authentication challenge) from the cash machine's display and sign the challenge on their mobile phone to prove their identity, instead of having to insert the card physically.

6.2 Digital Signature and Fraud Prevention

An important advantage of the proposed scheme's digital signature technology when compared to passwords and other one-time passwords is that the signature effectively verifies that the user has agreed to the transaction in a formally provable manner — as presented in Section 3.5.4.

A simple and accessible digital signature scheme could help service providers prevent fraud where the customer first initiates a transaction, then claims that the transaction was fraudulent and that it was not them who initiated it.

6.3 Advantages of Public Key Infrastructure

The scheme currently operates with simple private and public keys that do not belong to a certificate authority. While this design makes Authreq less centralised and may be a desirable situation from a privacy standpoint, it introduces several disadvantages. For example:

- There is no way to identify who signed an authentication challenge without acquiring their public key first.
- There is a timeframe during enrolment where an attacker could potentially enrol on someone else's behalf by acquiring the QR code and sending back their own signature instead. The server has no way to verify the origin and owner of the public key.
- In the event of a potential leak of private keys, there is no channel of revocation.

The scheme could rely on the concept of Public Key Infrastructure with little modifications. We could modify the scheme to instruct Secure Enclave to generate a Certificate Signing Request, which, along with other data, would be submitted to a certificate authority. This authority would provide the client with a certificate, and offer centralised means of verification and revocation.

It is a privacy concern of the current implementation that we use the same private and public key with all third-party services. This gives third-party services means of connecting accounts of the same user at various services. One may wish to take this into consideration when extending the scheme with PKI.

6.4 Key Replacement

Keys within the Secure Enclave cannot be exported or migrated to other devices. Users can lose their private key under a variety of scenarios (for example, when the device is lost or replaced, or when contents are erased), leading to an important question: what happens when the key can no longer function?

In Section 5.2 we skipped evaluating Authreq based on its key replacement procedure because currently there isn't one. However, key replacement could be solved by introducing a chain of trust via PKI. A certificate authority could add an abstraction layer where it would link devices to *users* via a trust chain. Then, it would be *users* and not *devices* that are linked to accounts at third-party services providers. Users could have an unlimited number of devices — each with their own private and public key — certified by the certificate authority, allowing users to sign authentication challenges with any of their devices. The proposed modification is depicted on Figure 15.

It is the author's opinion that such scheme would be most effective if it were a native feature of the device. The device manufacturer could start acting as a Certificate Authority by automatically enrolling users' devices in a centralised scheme and linking devices with the corresponding user account (i.e. linking device keys to Apple IDs). Users could sign challenges with their devices, and the manufacturer could verify which account the device belonged to (i.e. whose Apple ID was the challenge was signed with).

6.5 Adding Other Means of Communication

As discussed in Section 4.1.2, our implementation uses APNs, QR codes and buttons for authentication challenge delivery, and HTTPS for replies. This is a potential constraint, as there are scenarios when signatures are necessary even though there is no internet connection on the device.

To mitigate this issue, the scheme could be extended with other delivery methods. Due to the scheme's simplicity, we could easily create an SMS gateway that delivered authentication requests and listened for replies. Users could send replies in SMS when both Wi-Fi and mobile data are unavailable.

Furthermore, we could extend Authreq to optionally pass replies to the iOS share sheet as a failsafe. This would allow users to put the reply on the clipboard, or send it via AirDrop to another device, and submit the reply to the server from there.

The scheme's demo is available on the App Store. Visit `https://authreq.szente.info/` for more information.

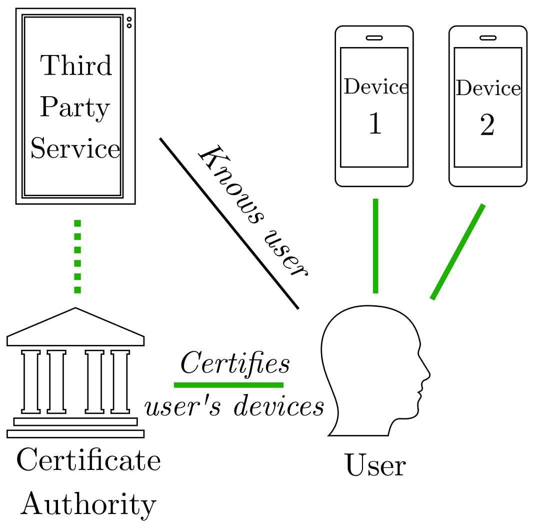


Figure 15: Authreq with PKI

References

[1] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, December 1999. ISSN 0001-0782. doi: 10.1145/322796.322806. URL `http://doi.acm.org/10.1145/322796.322806`.

[2] A.M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media, 2014. ISBN 9781491902646. URL `https://books.google.hu/books?id=IXmrBQAAQBAJ`.

[3] Apple. How iOS security really works, 2016. WWDC 2016 Lecture, `https://developer.apple.com/videos/play/wwdc2016/705/` Accessed on 18/04/2018.

[4] Apple. APNs overview, 2017. iOS Developer Documentation, `https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html` Accessed on 18/04/2018.

[5] Apple. Storing keys in the Secure Enclave, 2017. iOS Developer Documentation, `https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave` Accessed on 18/04/2018.

[6] Apple. Security framework: Seckeygeneratepair, 2018. Apple Developer Documentation, `https://developer.apple.com/documentation/security/1395339-seckeygeneratepair` Accessed on 18/04/2018.

[7] Apple. iOS security, 2018. Official White Paper, `https://www.apple.com/business/docs/iOS_Security_Guide.pdf` Accessed on 18/04/2018.

[8] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008. doi: 10.1080/10447310802205776. URL `https://doi.org/10.1080/10447310802205776`.

[9] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS). RFC 4492, RFC Editor, May 2006. URL http://www.rfc-editor.org/rfc/rfc4492.txt. http://www.rfc-editor.org/rfc/rfc4492.txt.

[10] John Brooke. *"SUS-A quick and dirty usability scale." Usability evaluation in industry.* CRC Press, June 1996. URL https://www.crcpress.com/product/isbn/9780748404605. ISBN: 9780748404605.

[11] Jayaditya Chakrabarty. Fingerprint scanner on phones: History & evolution, but do we really need that?, 2016. iGadgetsworld, https://www.igadgetsworld.com/fingerprint-scanner-history-evolution-but-do-we-really-need-that/ Accessed on 18/04/2018.

[12] Emiliano De Cristofaro, Honglu Du, Julien Freudiger, and Gregory Norcie. Two-factor or not two-factor? A comparative usability study of two-factor authentication. *CoRR*, abs/1309.5344, 2013. URL http://arxiv.org/abs/1309.5344.

[13] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26, 2014.

[14] Alexander De Luca, Alina Hang, Emanuel von Zezschwitz, and Heinrich Hussmann. I feel like I'm taking selfies all day!: Towards understanding biometric authentication on smartphones. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1411–1414, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702141. URL http://doi.acm.org/10.1145/2702123.2702141.

[15] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 581–590, New York, NY, USA, 2006. ACM. ISBN 1-59593-372-7. doi: 10.1145/1124772.1124861. URL http://doi.acm.org/10.1145/1124772.1124861.

[16] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006. ISSN 0018-9448. doi: 10.1109/TIT.1976.1055638. URL http://dx.doi.org/10.1109/TIT.1976.1055638.

[17] Yii PHP Framework. The definitive guide to Yii 2.0, 2018. Website https://www.yiiframework.com/doc/guide/2.0/en/intro-yii Accessed on 18/04/2018.

[18] GlobalPlatform. Trusted execution environment (TEE) guide, 2017. Whitepaper https://www.globalplatform.org/mediaguidetee.asp Accessed on 18/04/2018.

[19] National Security Agency Information Assurance Mission. Commercial national security algorithm suite, 2015. https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm Accessed on 18/04/2018.

[20] National Institute of Standards Information Technology Laboratory and Technology. Digital signature standard (DSS), 2013. Federal Information Processing Standards Publication FIPS PUB 186-4, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[21] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1): 36–63, Aug 2001. ISSN 1615-5262. doi: 10.1007/s102070100002. URL https://doi.org/10.1007/s102070100002.

[22] Peter Landrock and Torben Pedersen. WYSIWYS? - what you see is what you sign? *Information Security Technical Report*, 3(2):55 – 61, 1998. ISSN 1363-4127. doi: https://doi.org/10.1016/S0167-4048(98)80005-8. URL http://www.sciencedirect.com/science/article/pii/S0167404898800058.

[23] Salah Machani, Rob Philpott, Sampath Srinivas, John Kemp, and Jeff Hodges. FIDO UAF architectural overview - FIDO Alliance proposed standard. Technical report, 2017. https://fidoalliance.org/specs/fido-uaf-v1.1-id-20170202/fido-uaf-overview-v1.1-id-20170202.html Accessed on 18/04/2018.

[24] Karen Renaud. Quantifying the quality of web authentication mechanisms: A usability perspective. *J. Web Eng.*, 3(2):95–123, October 2004. ISSN 1540-9589. URL http://dl.acm.org/citation.cfm?id=2011143.2011146.

[25] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent E. Seamons. Why Johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. *CoRR*, abs/1510.08555, 2015. URL http://arxiv.org/abs/1510.08555.

[26] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - OCSP. RFC 6960, RFC Editor, June 2013. URL http://www.rfc-editor.org/rfc/rfc6960.txt.

[27] Mastercard Payment Gateway Services. 3-D Secure, 2018. Integration Guides, https://www.mastercard.com/gateway/integration-guides.html Accessed on 18/04/2018.

[28] Sampath Srinivas, Dirk Balfanz, Eric Tiffany, and Alexei Czeskis. Universal 2nd factor (U2F) overview - FIDO Alliance proposed standard. Technical report, 2017. https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html Accessed on 18/04/2018.

[29] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251421.1251435.

[30] Xudong Zheng. Phishing with unicode domains, 2017. https://www.xudongz.com/blog/2017/idn-phishing/ Accessed on 18/04/2018.

APPENDIX

A. COMMUNICATION OVER TIME

See Figure 16.

B. QUALITY OF 2FA METHODS

Dim & Aspect	*Coeff.*	*Reasons*
Accessibility	*ad=0.33*	
Special Requir.	0	
Convenience	0	No extra step
Inclusivity	0.33	Need to remember password
Memorability	*md=1.45*	
Retrieval	1	Need to recall
Meaningfulness	0.33	Self-assigned
Depth of proc.	1	Effort to remember
Security	*sd=0.61*	
Predictability	0.25	May be predictable
Abundance	0.25	Can choose any key, but need to type
Disclosure	0.50	Possible to observe
Vulnerability	*vd=1.43*	
Confidentiality	1	Full secret always shared
Privacy	0.25	Password may be reused — problem at compromise
Breakability	1	Keyboard tapper

Table 2: Quality of basic password authentication, measured with the scheme of Renaud

Dim & Aspect	*Coeff.*	*Reasons*
Accessibility	*ad=0.50*	
Special Requir.	0	
Convenience	0.50	May take time to arrive. Need to enter code manually.
Inclusivity	0	
Memorability	*md=0.25*	
Retrieval	0	N/A
Meaningfulness	0	N/A
Depth of proc.	0.25	Need to remember code while typing
Security	*sd=0.56*	
Predictability	0	
Abundance	0.50	Low key entropy
Disclosure	0.25	Phone number may be stolen
Vulnerability	*vd=0.41*	
Confidentiality	0	Code cannot be reused
Privacy	0.25	Requires phone number, which is personal
Breakability	0.33	Research-based attack

Table 3: Quality of SMS one-time passwords, measured with the scheme of Renaud

Dim & Aspect	*Coeff.*	*Reasons*
Accessibility	*ad=1.45*	
Special Requir.	0.33	Need to carry around another device
Convenience	1.00	Takes a very long time to operate
Inclusivity	1.0	Potentially excludes users with cognitive, mobility and sensory disabilities
Memorability	*md=0.25*	
Retrieval	0	N/A
Meaningfulness	0	N/A
Depth of proc.	0.25	Need to remember code while typing
Security	*sd=0.50*	
Predictability	0	
Abundance	0.50	Low key entropy
Disclosure	0	Card still requires PIN if stolen
Vulnerability	*vd=0.25*	
Confidentiality	0.25	May be reused in same time frame
Privacy	0	
Breakability	0	

Table 4: Quality of card reader based one-time passwords, measured with the scheme of Renaud

Dim & Aspect	*Coeff.*	*Reasons*
Accessibility	*ad=0.60*	
Special Requir.	0.33	App
Convenience	0.50	Need to open app, then enter code manually.
Inclusivity	0	
Memorability	*md=0.25*	
Retrieval	0	N/A
Meaningfulness	0	N/A
Depth of proc.	0.25	Need to remember code while typing
Security	*sd=0.56*	
Predictability	0	
Abundance	0.50	Low key entropy
Disclosure	0.25	Shared secret can be stolen
Vulnerability	*vd=0.25*	
Confidentiality	0.25	May be reused in same time frame
Privacy	0	
Breakability	0	

Table 5: Quality of one-time passwords from Google Authenticator, measured with the scheme of Renaud

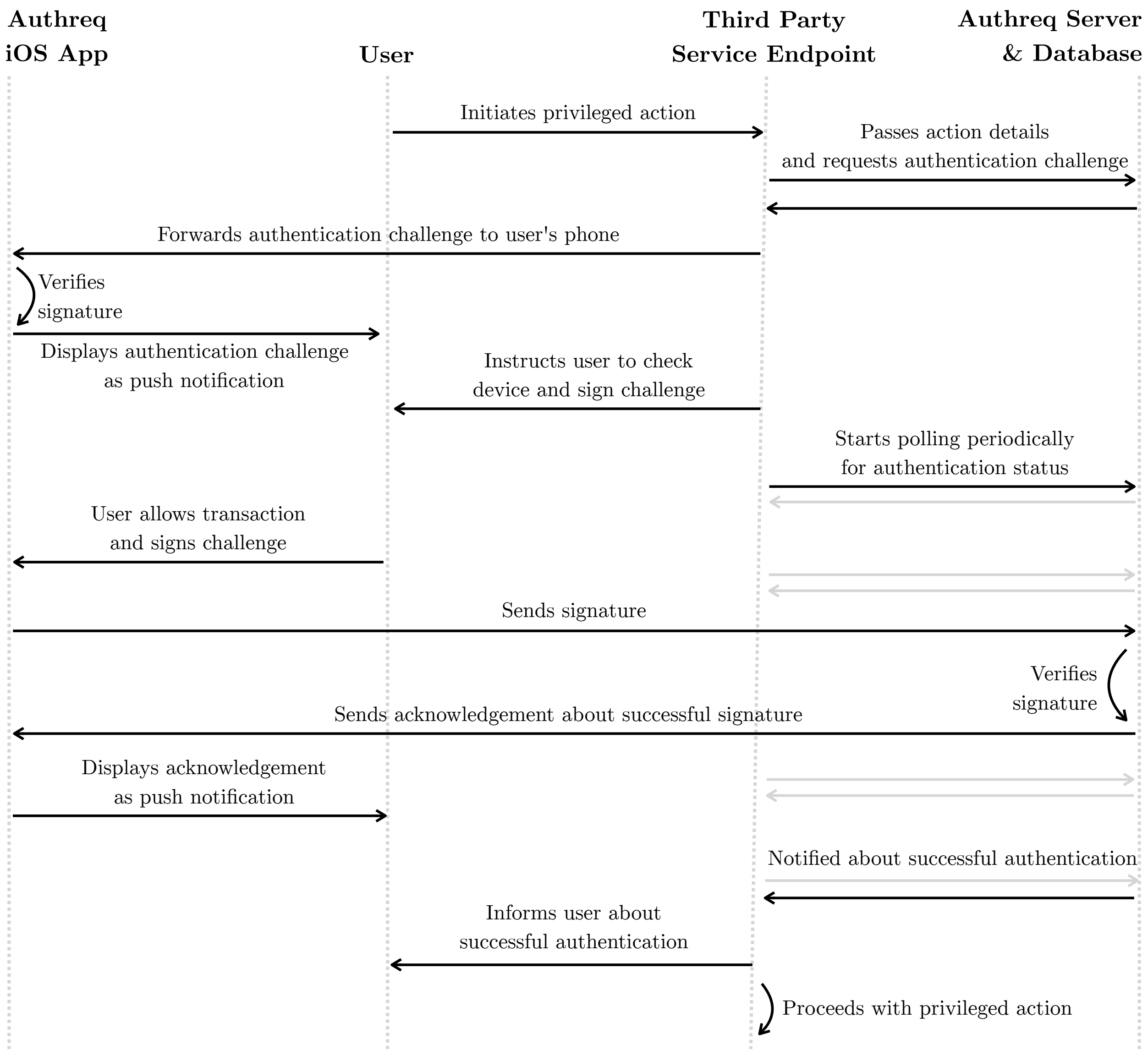

Authreq iOS App
User
Third Party Service Endpoint
Authreq Server & Database
Initiates privileged action
Passes action details and requests authentication challenge
Forwards authentication challenge to user's phone
Verifies signature
Displays authentication challenge as push notification
Instructs user to check device and sign challenge
Starts polling periodically for authentication status
User allows transaction and signs challenge
Sends signature
Verifies signature
Sends acknowledgement about successful signature
Displays acknowledgement as push notification
Notified about successful authentication
Informs user about successful authentication
Proceeds with privileged action

Figure 16: Communication over time

C. USER STUDY - DEMOGRAPHICS

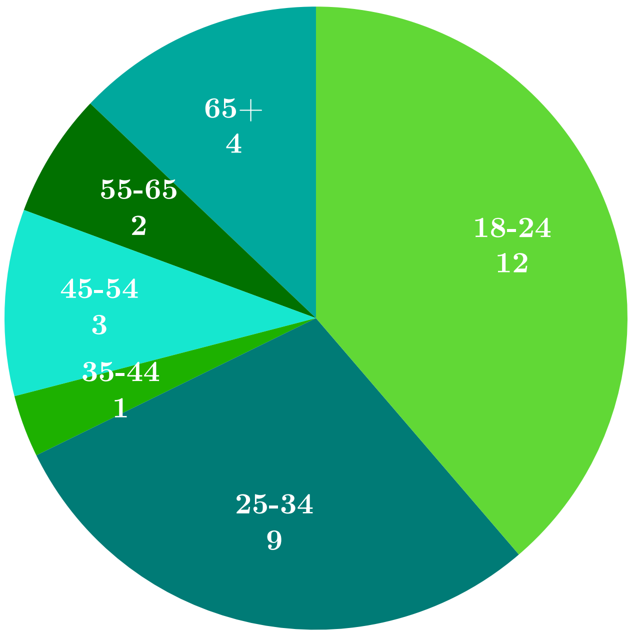


Figure 17: Age of participants

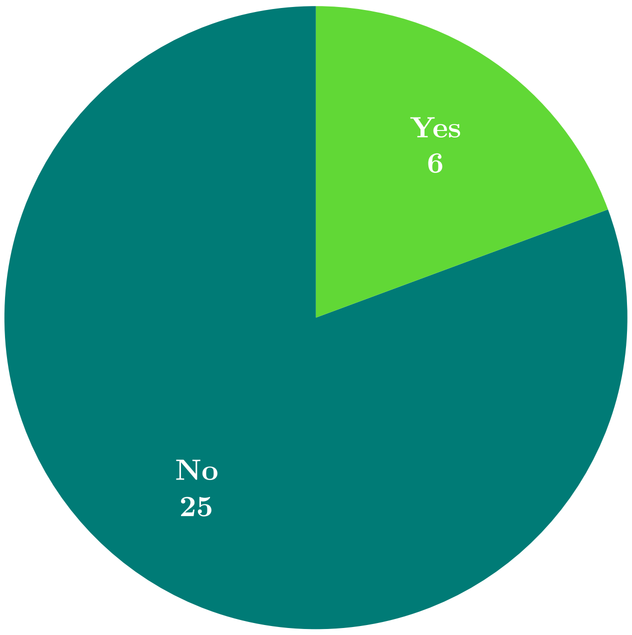


Figure 19: Background in computing science

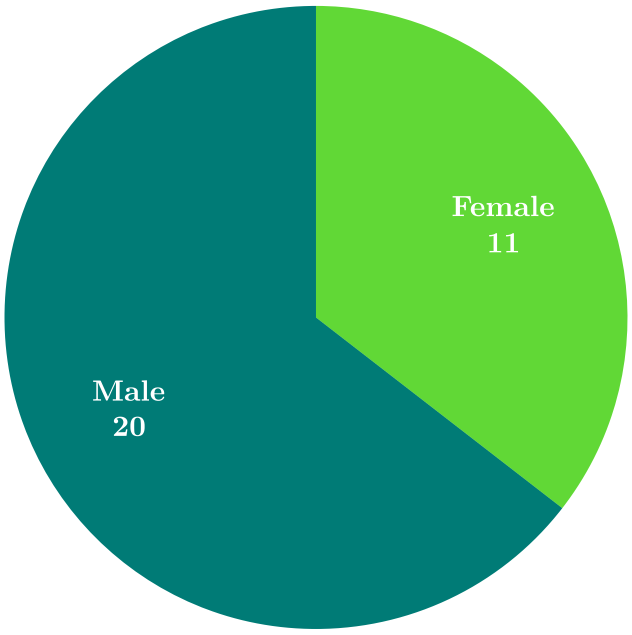


Figure 18: Gender of participants

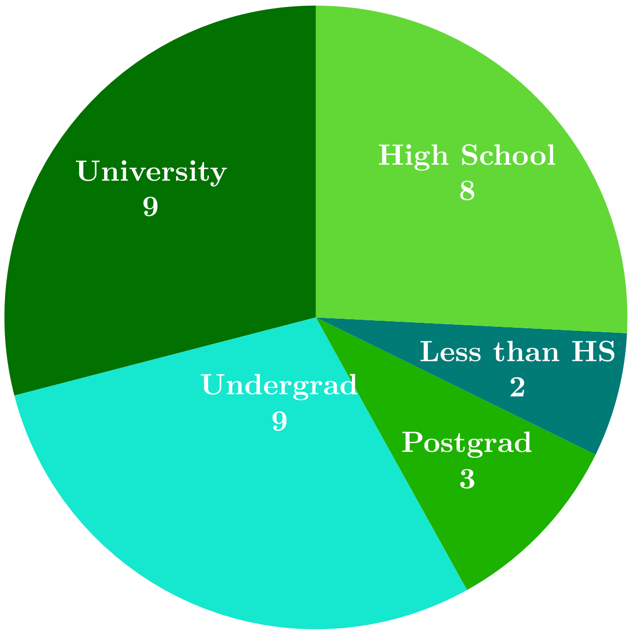


Figure 20: Educational background of participants

D. USER STUDY

	Initial	Extraction
Convenient	1.000	.872
Quick	1.000	.900
Enjoy using	1.000	.898
Happy to use again	1.000	.911
Helpful	1.000	.794
Not enjoy using	1.000	.851
User friendly	1.000	.840
Needed instructions	1.000	.737
Had to concentrate	1.000	.738
Stressful	1.000	.752
Did not match expectations	1.000	.601
Trustworthy	1.000	.418

Figure 21: Communalities across variables

	Total	**% of Variance**	**Cumulative %**
	Initial Eigenvalues		
1	8.154	67.946	67.946
2	1.159	9.655	77.602
3	.784	6.536	84.137
4	.473	3.944	88.081
5	.377	3.138	91.219
6	.263	2.195	93.414
7	.254	2.119	95.533
8	.206	1.717	97.250
9	.123	1.024	98.274
10	.088	.732	99.006
11	.068	.565	99.571
12	.052	.429	100.000
	Extraction Sums of Squared Loadings		
1	8.154	67.946	67.946
2	1.159	9.655	77.602
	Rotation Sums of Squared Loadings		
1	6.407	53.394	53.394
2	2.905	24.208	77.602

Figure 22: Variance

	Mean	N	Std. Deviation
SMS	2.2679	209	1.16230
Card Reader	4.0905	210	1.27035
Google Authenticator	1.9095	210	0.82218
Authreq	1.1952	210	0.58283
Cumulative	2.3659	839	1.46097

	Sum of Squares	df	Mean Square	F
Between Groups (Comb.)	958.113	3	319.371	321.081
Within Groups	830.552	835	0.995	
Total	1788.665	838		

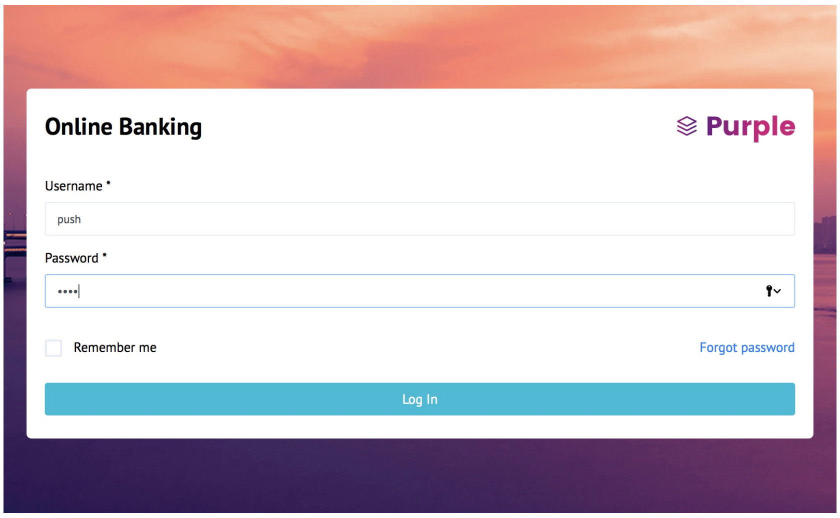
Figure 23: ANOVA for Ease of Use and Trust

	Mean	N	Std. Deviation
SMS	4.3067	150	1.08031
Card Reader	2.0933	150	1.24968
Google Authenticator	4.0733	150	1.14749
Authreq	4.4267	150	1.07658
Cumulative	3.7250	600	1.48314

	Sum of Squares	df	Mean Square	F
Between Groups (Comb.)	542.152	3	180.717	138.893
Within Groups	775.473	596	1.301	
Total	1317.625	599		

Figure 24: ANOVA for Cognitive Efforts

E. SCREENSHOTS OF ENROLMENT


Online Banking

Purple

Username *

push

Password *

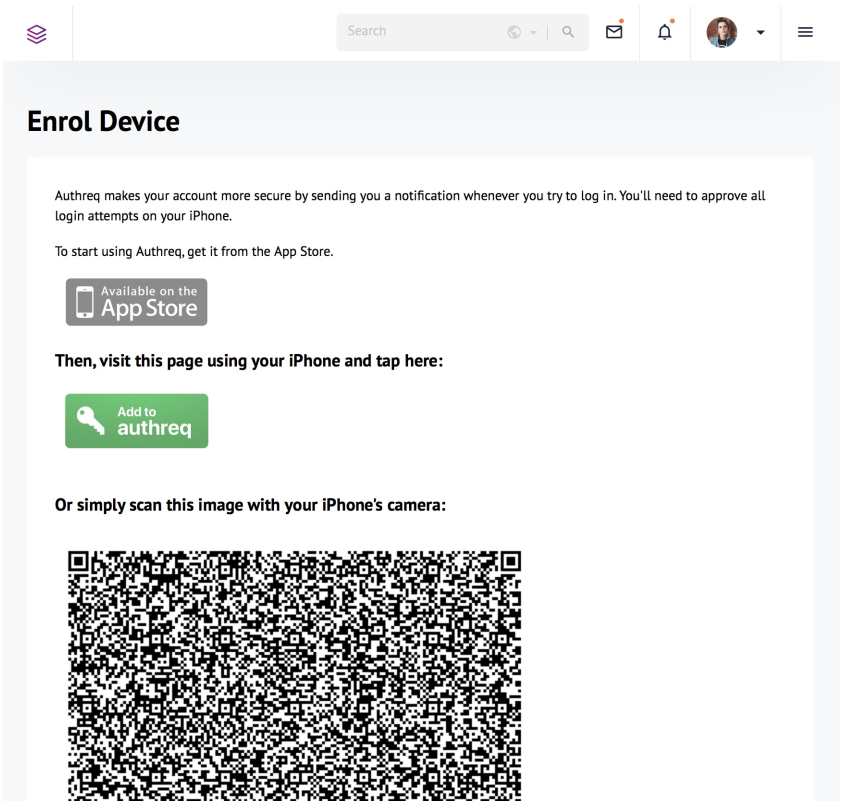
••••

Remember me

Forgot password

Log In

Figure 25: The user logs in with their usual credentials, and starts the enrolment procedure by selecting Protect account with Authreq


Enrol Device

Authreq makes your account more secure by sending you a notification whenever you try to log in. You'll need to approve all login attempts on your iPhone.

To start using Authreq, get it from the App Store.

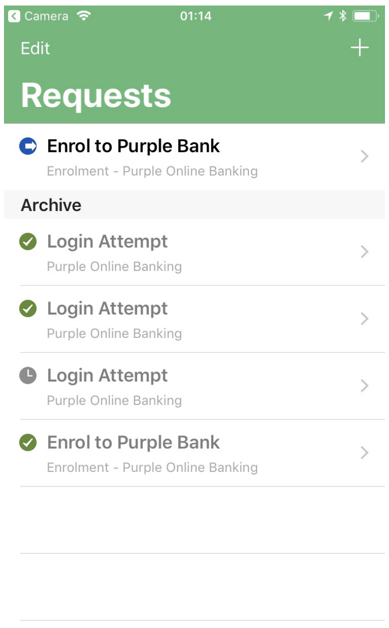
Available on the App Store

Then, visit this page using your iPhone and tap here:

Add to authreq

Or simply scan this image with your iPhone's camera:

Figure 26: The enrolment page presents a button that the user can tap, or a QR code that they can scan to add the enrolment challenge to Authreq.



Requests

Enrol to Purple Bank
Enrolment - Purple Online Banking

Archive

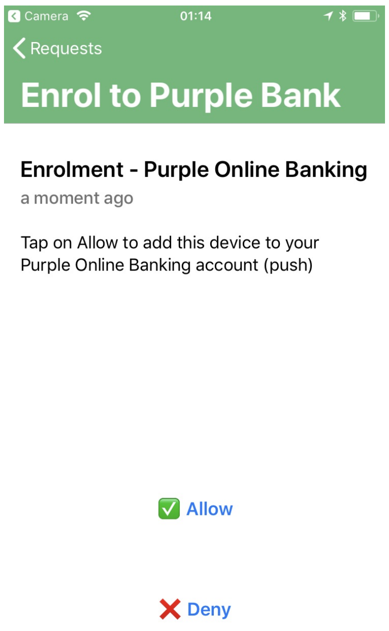
Login Attempt
Purple Online Banking

Login Attempt
Purple Online Banking

Login Attempt
Purple Online Banking

Enrol to Purple Bank
Enrolment - Purple Online Banking

Figure 27: Authreq is automatically opened by scanning the QR code or tapping on the button. An enrolment challenge appears.

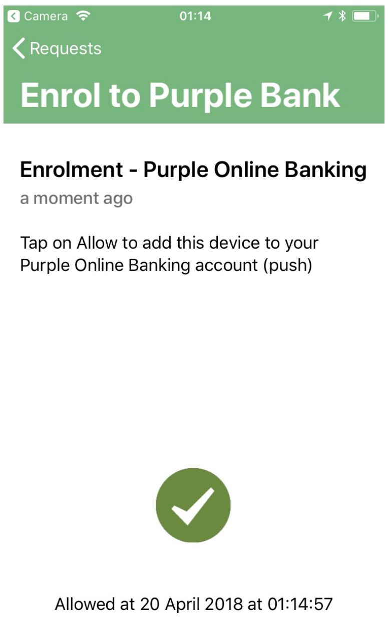


Enrol to Purple Bank

Enrolment - Purple Online Banking
a moment ago

Tap on Allow to add this device to your
Purple Online Banking account (push)

Figure 28: By tapping on Allow, the device sends a reply to the challenge — and thus sends its public key to the service provider.



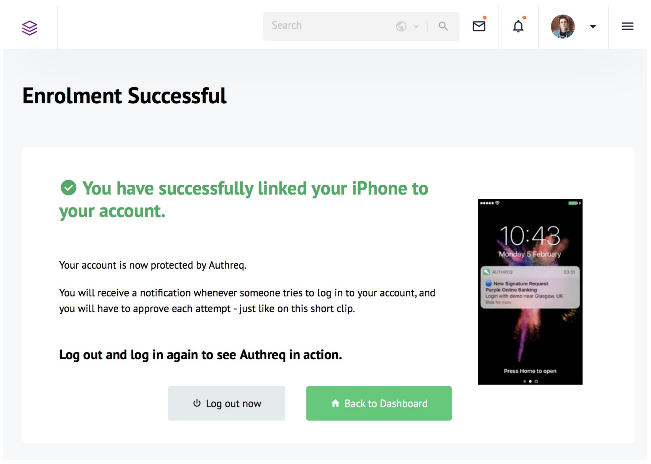
Enrol to Purple Bank

Enrolment - Purple Online Banking
a moment ago

Tap on Allow to add this device to your
Purple Online Banking account (push)

Allowed at 20 April 2018 at 01:14:57

Figure 29: Successful enrolment is marked by a green tick on the device.



Enrolment Successful

You have successfully linked your iPhone to your account.

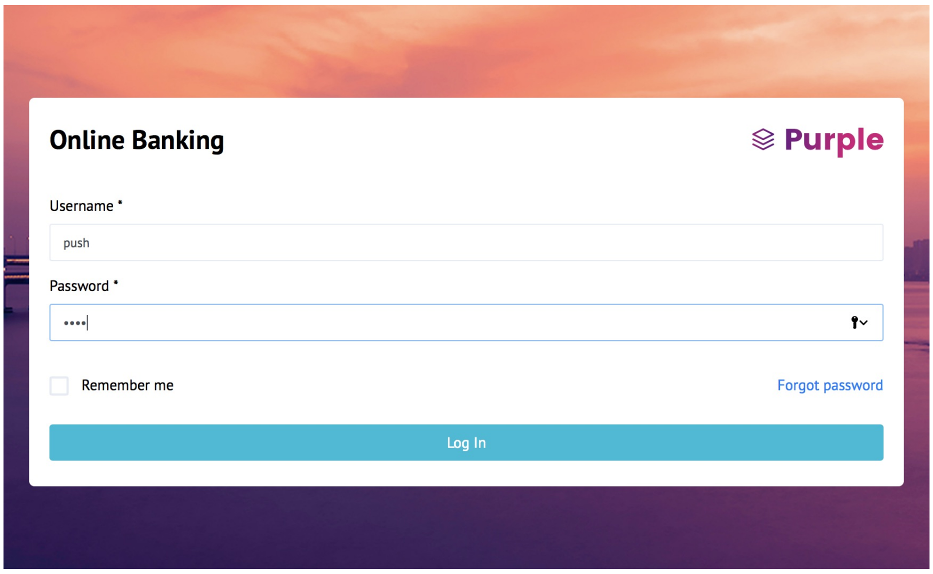
Your account is now protected by Authreq.

You will receive a notification whenever someone tries to log in to your account, and you will have to approve each attempt - just like on this short clip.

Log out and log in again to see Authreq in action.

Figure 30: The website automatically proceeds to a confirmation screen.

F. SCREENSHOTS OF AUTHENTICATION


Online Banking

Purple

Username *

push

Password *

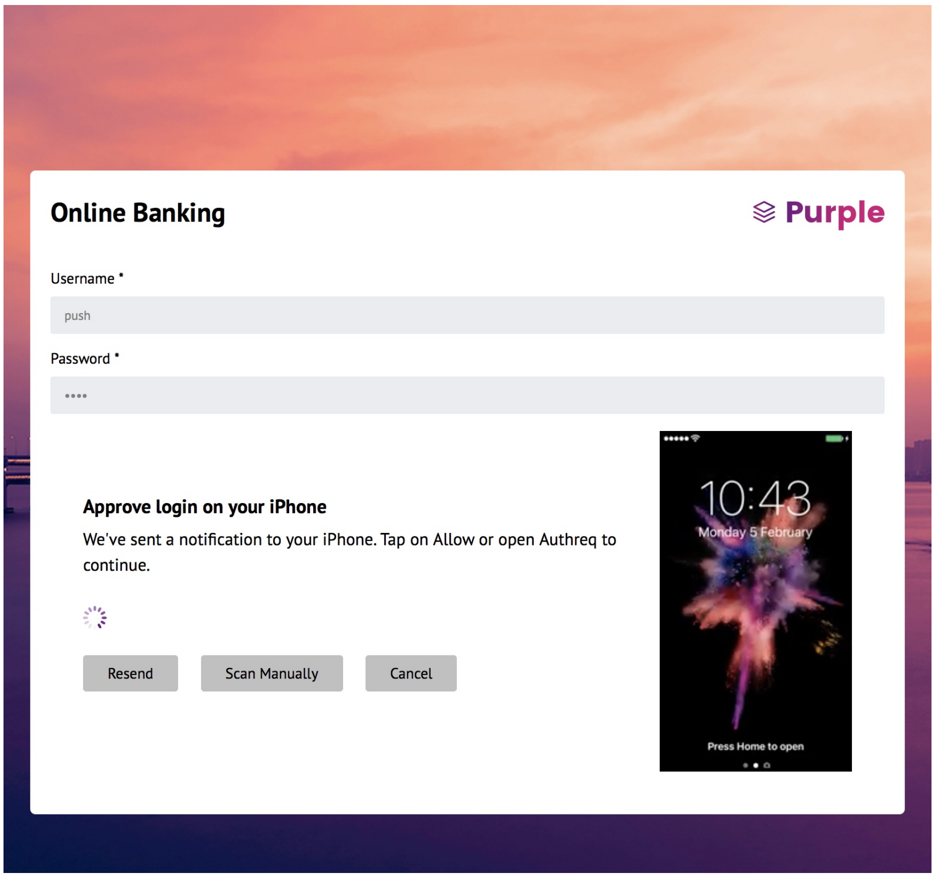
••••

Remember me

Forgot password

Log In

Figure 31: The user begins authentication by entering their user name and password.


Online Banking

Purple

Username *

push

Password *

••••

Approve login on your iPhone

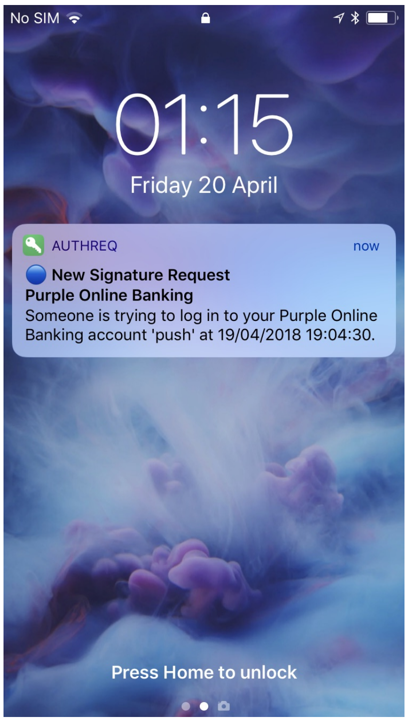
We've sent a notification to your iPhone. Tap on Allow or open Authreq to continue.

Resend

Scan Manually

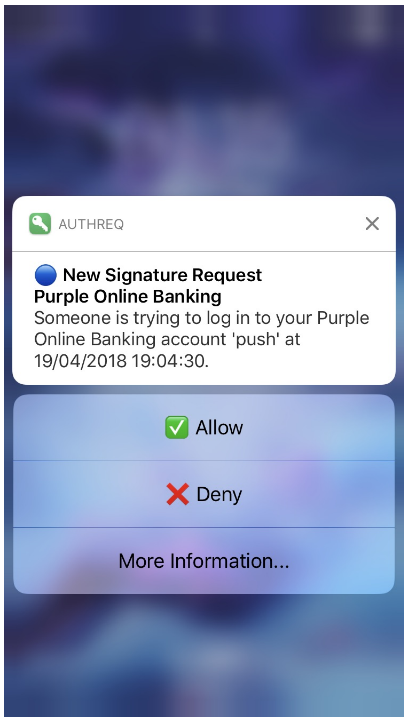
Cancel

Figure 32: The user is told to approve login on their iPhone.



🔵 New Signature Request
Purple Online Banking
Someone is trying to log in to your Purple Online Banking account 'push' at 19/04/2018 19:04:30.

Figure 33: A notification appears on the user's device, containing login details.



🔵 New Signature Request
Purple Online Banking
Someone is trying to log in to your Purple Online Banking account 'push' at 19/04/2018 19:04:30.

Figure 34: More information and actions can be revealed via 3D Touch.

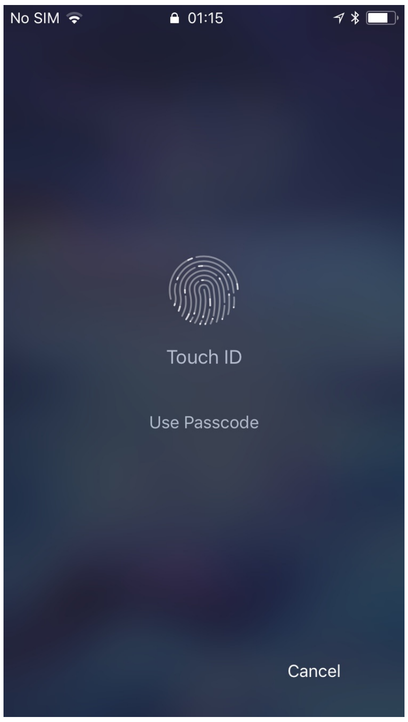
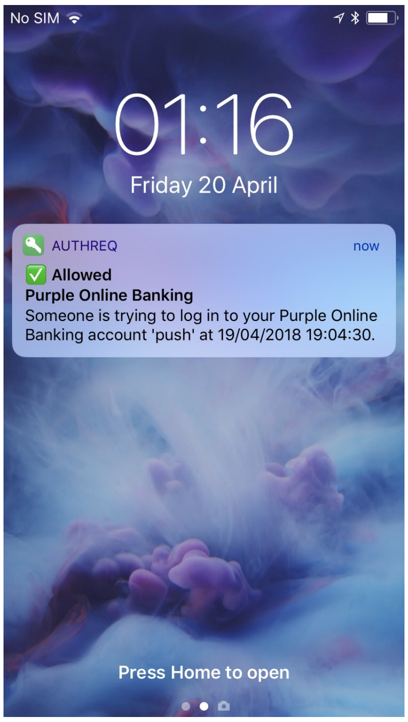


Figure 35: Upon tapping on Allow, the device prompts for local authentication.



✅ Allowed
Purple Online Banking
Someone is trying to log in to your Purple Online Banking account 'push' at 19/04/2018 19:04:30.

Figure 36: Successful authentication is acknowledged by another notification. User identity is now confirmed. The website automatically proceeds with login