KeyPocket - Improving Security and Usability for Provider Independent Login Architectures with Mobile Devices

André Ebert[1], Chadly Marouane[2], Benno Rott, and Martin Werner

[1] Ludwig-Maximilians-University, Mobile and Distributed Systems Group, Oettingenstrasse 67, Munich 80538, Germany, andre.ebert@ifi.lmu.de, marouane@virality.de, rott@virality.de, martin.werner@ifi.lmu.de

[2] Virality GmbH, Rauchstrasse 7, 81679 Munich, Germany

Abstract. Nowadays, many daily duties being of a private as well as of a business nature are handled with the help of online services. Due to migrating formerly local desktop applications into clouds (e.g., Microsoft Office Online, etc.), services become available by logging in into a user account through a web browser. But possibilities for authenticating a user in a web browser are limited and employing a username with a password is still de facto standard, disregarding open security or usability issues. Notwithstanding new developments on that subject, there is no sufficient alternative available. In this paper, we specify the requirements for a secure, easy-to-use, and third-party-independent authentication architecture. Moreover, we present KeyPocket, a user-centric approach aligned to these requirements with the help of the user's smartphone. Subsequently, we present its state of implementation and discuss its individual capabilities and features.

Key words: Multi-factor authentication; Mobile-based login architectures; Security and usability

1 Introduction

Whether booking a journey, transferring money to a bank account, or reading the most recent news: Mobile devices with online capabilities are ubiquitous and their usage is common today [1]. But service providers insist on the user to create a user account, commonly secured by a username and a password, still. This leads to the dilemma of having a trade-off between security and usability. Short passwords, which do not contain special signs, numerics, or capital and lowercase letters are good to remember, but have a lack of security. In contrast, a complex password is hard to remember and users tend to write it down, which makes it accessible for possible attackers [2]. Even in the pre-smartphone era, users already had an average number of thirty user accounts and about 6.5 passwords to secure them [3]. This results in a multiple or combined usage of the same passwords for different services. But if one account is compromised, all accounts

secured with its credentials are also compromised [4]. Furthermore, complex passwords which have been robust against Brute-force attacks in the past, get stolen by Phishing or Keylogging mechanisms today [5]. Parwani et al. stated, that a huge number of private as well as of business accounts are accessed illegally every day, which results in great personal and financial damage [6]. Therefore, Bonneau et al. explored more than 35 different authentication approaches and claim that their security and usability is not suitable to substitute authentication systems based on usernames and passwords, yet [7]. Due to rapid development of smartphones and their capabilities to act as a mobile sensor, hardware tokens, or transmission systems, their inclusion in multi-factor authentication architectures is accelerated. Target is the improvement of usability and security in relation to conventional systems as well as the provision of a trustworthy and easy accessible platform for managing different credentials and identities for a user. Still, we are not aware of a login architecture which satisfies these needs concerning usability, security, data privacy, and service-independence.

In the following, we first present some fundamentals related to mobile authentication as well as a choice of current authentication concepts in order to explain their technical and architectural concepts in Section 2. The following section defines requirements for a login architecture as a function of usability, security, and technical as well as conceptual circumstances. Afterwards, we present KeyPocket, a provider-independent, easy-accessible, and secured login architecture along with some of its unique features in Section 4. Furthermore, this section also deals with concrete insights into the system's implementation. Section 5 explores the mentioned concept in reference to the demanded requirements and discusses its features in respect of possible threats for mobile authentication. Afterwards, we summarize our findings and provide a glimpse towards open issues and future tasks.

2 Related work

In this section, we present some security fundamentals in context of mobile authentication. Moreover, we provide a brief overview across recent mobile-device-based authentication architectures.

2.1 Secure data encryption and transmission

Procedures for encryption are essential for the development of a authentication architecture based on a mobile device due to several reasons. On the one hand, it guarantees a secure data storage, on the other hand data integrity and confidence during a transmission process is ensured. Munro et al. proclaim, that it was difficult to encrypt data on Android smartphones, so far. Especially PIN-based encryption methods were susceptible for Brute-force attacks and did not provide adequate protection for sensible data [8]. But since the introduction of Android 5.0 Lollipop, hardware-based disk encryption is integrated into the operation system (OS). Therefore, a 128 bit Advanced Encryption Standard (AES)

algorithm in combination with Cipher Block Chaining (CBC) is used. The master key also uses 128 bit AES encryption. For secure usage a key length of 256 bit is recommended [9]. Until now, we are not aware of any successful attacks onto recent Android versions. The procedure can currently be rated as secure for encrypting data on Android devices.

Apple iOS offers its Keychain feature for secure credential management, which was already part of Apples desktop operation system OS X. It realizes secure handling and inspection of certificates as well as of user credentials. There also occurred different security issues with the Keychain since its release, especially in combination with the processor architecture of some older iPhone models and jail-breaked devices. Referring to this issue, Heider et al. show that data can be stored in a save way on recent iOS-devices by using the Keychain, still [10].

The TLS encryption protocol, formerly known as Secure Sockets Layer (SSL), was originally developed for the combined usage with HTTPS in web browsers and is a de facto standard for secured end-to-end communication in networks. Therefore, the server provides a valid public key certificate (issued by a trustworthy Certification Authority (CA)) to the client during the handshake procedure, who is now able to validate it (e.g., period of validity, listed domain names, etc.) [11, 12]. Due to security vulnerabilities, which emerged during the last years, there were doubts about the SSL technology's reliability. Despite that, Georgie et al. show that mainly inaccurate implementations (e.g., deactivated certificate validation) or poorly designed SSL libraries and not the protocol itself are responsible for these flaws [13]. An example for a security leak, which got a lot of attention during the last year was the Heartbleed bug. It was detected in the OpenSSL framework and facilitated the readout of 24 - 55% of the memory of popular HTTPS site's servers. However, the bug was fixed in version 1.0.1g [14, 15]. In addition, Georgie et al. indicate that SSL can be used in a secure manner without any issues, as long as configuration parameters are set explicitly and development guidelines are followed. Irrespective of that, HTTPS and TLS/SSL do not protect the user from every kind of connection-aimed attacks. Callegati et al. indicate that even HTTPS connections are not immune against Man-in-the-Middle attacks (MitM) [16].

2.2 Smartphone-based login architectures

Public-key-cryptography-based login architectures use encryption in order to protect the user's credentials from unauthorized access. In context of the chosen architectural concept, the used technologies, assigned roles and identified tasks are significantly dependent of the proposed system-design. All approaches introduced in the following integrate a smartphone into their authentication infrastructure. Its role differs in each individual concept (e.g., hardware token, data transmission, identity management, etc.).

Czeskis et al. proclaim an authentication system for opportunistically provided cryptographic identity assertion, called PhoneAuth [17]. The approach is called opportunistic because it is only used if the user fulfills the required system

setup, consisting of a compatible web browser and a smartphone. In order to use PhoneAuth, the user first visits the web page he desires to log in to and enters his credentials. Subsequently, the browser redirects this data to the server, which creates a login ticket with a challenge for authentication. This ticket is sent back to the browser by using a TLS encrypted connection, which forwards it to the user's smartphone. The device functions as the second factor possession and is registered explicitly as belonging to the user. By signing the browsers public key contained by the login ticket with the user's private key on the smartphone, the user's identity can be proofed without doubt to the server. As soon as the user's identity is verified, the server sets a cookie which is channel-bound to the browser's key pair and the user becomes logged in. The authors state, that all MitM attacks are perceived by the usage of a TLS channel ID, which is unique for each communication partner. However, in its current state the system is not able to provide a reliable and usable management for complex passwords. They still need to be entered manually into the websites login form.

Based on the research of the university of Tübingen, Borchert et al. present a system for an indirect login under the usage of NFC [18]. For processing it, a smartphone and a NFC smartcard containing the user's asymmetric key material is needed. In order to conduct the indirect login the server generates a challenge, which is encoded together with the server's address in a two dimensional code (e.g., QR-Code) and shown at the login page. After scanning the code, the login address is presented to the user for confirmation. This allows the system to suspend MitM attacks. Subsequently, the user brings his smartcard near the smartphone and the challenge as well as the server's name are forwarded from the mobile device to the card via NFC. The smartcard now computes the response in terms of a private-key-signed challenge, which is sent back to the sever in combination with the original challenge and the username. Because of an in prior carried-out registration process, the user's public key is already known to the server, which is now able to verify the user's proclaimed identity. The authors do not make any specifications about the usage of the smartcard's PIN function. Furthermore, the smartphone's only functionality is being a relay between the smartcard and the server. The predecessor of the indirect NFC login called *ekaay* was also developed by Borchert et al. and is a one-factor possession architecture without the inclusion of a smartcard [19]. In this scenario, a new key pair is created on registration and the pre-shared public key is sent to the user. To proceed with the login, the user scans the shown QR-Code and signs the contained challenge, which enables the server to verify the users authorization permission. Because of the pre-shared key practice, this method is susceptible for MitM attacks. In addition, the key is no one-time key, which means that as soon as the system is compromised, a secure data exchange is not possible any more.

Van Rijswijk et al. present *tiqr*, a concept similar to ekaay which facilitates the binding of each service account to a unique key on its creation [20]. The tiqr architecture itself is similar to ekaay and needs to be implemented on the service provider's website. Each user account is bound to a unique key. A difference

to ekaay is, that the key pair with the pre-shared public key is created locally on the user's device and subsequently transfered to the server. For the sake of login confirmation and protection from Phishing attacks, the user needs to check the login address and complete the procedure by entering his PIN. By binding accounts to individual keys as well as due to the constraint to implement tiqr on a service provider's server, the flexibility of the concept is weakened.

Snap2Pass also binds user accounts directly to a key and provides a symmetric as well as an asymmetric encryption mode [21]. Using the symmetric mode, a secret key is managed on the user's device for each server. A new account is created by scanning a QR-Code from the service provider's login page. The user provides only a username, the password is provided by the server in terms of a pre-shared secret. For logging in, the user again scans a QR-Code, which contains a challenge bound to the current browser session. The device computes the corresponding HMAC-SHA1-hash and sends the signed challenge together with the original challenge back to the server. The server now verifies both, the user and the browser session and completes the login procedure. In public key mode, a public key is generated on application startup on the user's device and is sent to the server, which now is able to verify the signed challenges. By using a pre-shared secret and storing it on the mobile device, the system is downgraded from the usage of two security factors (possession and knowledge) to one factor (possession).

The QR-Code based authentication concept propagated by *Galois Inc.* is similar to ekaay and tiqr, though it currently only exists as a loose concept without implementation and was published on the company's website [22]. For registration, a QR-Code containing a random secret, which is also saved in a session-bound cookie, must be scanned. In the following, either a username is entered by the user inside the corresponding smartphone application or a Unique-User-ID (UUID) is generated automatically. After sending the shared secret, the random secret, the session cookie and the UUID to the service provider, the registration process is completed. For logging in, the user scans a QR-Code containing a random secret and a session cookie. The smartphone's response contains the corresponding UUID and the appropriate random secret for the service's website. After reviewing the credentials the user becomes logged in automatically.

LastPass differs in multiple aspects from the systems mentioned above. It is a cloud-based SSO login architecture, which perches on the usage of a browser-plugin in order to communicate with the website of a service provider [23]. The user's inclusion into the LastPass system is carried out by using the corresponding authenticator application. To use LastPass, the user clicks a button in the browser to start the plugin. After entering his credentials, a one-time password is sent to the user's smartphone as a second authentication factor. This also needs to be entered into the browser-plugin, in order to complete the login process on the client's side. An advantage of this concept is its device independence due to storing and synchronizing user data in a cloud. By introducing the second factor possession, the system's security is increased. But in this case, the increased se-

curity also leads to a lack of usability, resulting in the login procedure becoming more complicated. Not only that the user needs to enter credentials for each login, additionally a one-time password is required, which results in multiple media breaks. Moreover, because of entering credentials in the browser-plugin the concept is vulnerable to Keylogging and Phishing attacks.

Besides smartphone-assisted login architectures without complex account or user management, there are also some commercially available architectures with extensive possibilities of managing personal data. Most of these do not use local, but cloud-based technologies to store user data, which leads to advantages regarding a provider's actionability, e.g., in case of device theft or loss. Otherwise, this also indicates privacy issues. Due to their proprietary architecture, the concepts presented in the following could not be evaluated completely in respect of their specific conceptual or technical details.

Click2Pass presents a login via smartphone application in combination with an own web API, which needs to be implemented on the provider's server [24]. Some PHP code fragments serve as an implementation guideline for developers, the comparatively protracted and complex registration process could discourage potential users.

MyDigipass provides a cloud-based two-factor authentication solution compatible with mobile applications as well as with websites. After the registration process, all personal data can be managed in a cloud. This offers a plus on usability,e.g., regarding data migration on device theft. Passwords are stored on a mobile device, which functions as a token. Currently, devices which are available as a token are Android and iOS smartphones and due to its eID function the Belgian identification card, among others. In order to log in, the user enters the MyDigipass launchpad where all registered services are listed. After choosing one service by a click on its icon, the user needs to enter a PIN code and after a successful verification he is redirected to the service's website or its mobile application.

LaunchKey and *Zapper* offer possibilities for multi-factor authentication based on a platform specific registration and implementation process [25, 26]. LaunchKey in particular features a decentralized architecture where the entire authentication layer resides on the user's mobile device. Therefore, a cryptographic connection between the user and mobile device is initiated via SMS, QR-Code, email or manual entry. Due to a variable usage of fingerprints, geofencing, bluetooth device check, PINs, etc., the implementation of granular security levels is feasible. After the pairing and the security level setup, the reception of authentication and authorization requests is possible.

Clef supplies a smartphone-based login working with OAuth 2.0, which is also used by OpenID and Twitter [27, 28]. A security enhancing feature of Clef is the usage of geolocations as well as the device's hardware information and usage data for fraud detection.

OneID also facilitates a cloud-based approach, where all user data is saved encrypted in order to enhance the users privacy [29]. For its transmission as well as its decryption, a pre-shared key is used, which is stored on the user's device.

Additionally an individual PIN can be set in order to secure this key. Even if a device gets lost, it is not possible to get access to the user's data itself due to its distributed storage in a cloud.

An authentication concept for Mac OS X based on BLE for an iPhone in combination with a MacBook is *Kocktounlock* [30]. As soon as the mobile phone is near the MacBook, it is prepared to become unlocked – only an additional knock onto the phone screen is needed as a signal of intent. Kocktounlock is not working anymore since Mac OS X Yosemite due to the lost feature of a MacBook to act as an iBeacon. Furthermore, the approach is not applicable to websites but only to unlock the OS' lock screen.

A generic, smartphone-based authentication solution which uses the smartphone as a key for vehicles and security doors or barriers is BlueID [31]. The communication between system components and the user's device is secured by an asymmetric public key infrastructure and the usage of certificates, which are issued by the system's own trust center. Data with login information is transferred optionally via WiFi, Bluetooth Smart, mobile network or NFC. For implementing the system on the service's side, a software development kit is provided.

3 Requirements of a mobile-based login architecture

In the context of developing an alternative login concept with the help of a mobile device, there is a rash of different requirements to be considered. Thus, we identified important basic points on basis of the concepts introduced before and by respecting the paradigms of usability.

1. **Security** There are several security aspects to be kept in mind: a) the user's account and personal data must be protected under all circumstances and stay secret as well as with integrity, even in case of device theft or loss, b) it must be ensured that only the user alone has access to personal data, and c) all data transfer is secured and encrypted in order to eliminate all kind of unwanted manipulation.
2. **Usability** The system's usage is as easy and its provided security as high as possible.
3. **Modularity, Compatibility and Scalability** Existing as well as new service accounts can be integrated into the login architecture easily and without technical limitations.
4. **Privacy** The user itself is the only person having access to user sensitive data.
5. **Third-Party-Independence** The architecture is independent of implementations, limitations, and restrictions, stated by existing service architectures or foreign providers.
6. **Hardware Independence** The hardware requirements are as little as possible; there is no need for additional hardware except the user's smartphone. All used devices are standard versions and commercially available.

4 KeyPocket - An architecture for secure and usable web service access

In the following we present KeyPocket, a user-centric, secured, and easy-accessible authentication architecture featuring multi-factor authentication as well as a decentralized identity management. The only precondition for usage is a smartphone and a computer with a compatible web browser. After highlighting its core concepts and main components, we also provide detailed information about its concrete implementation.

4.1 Architectural concept

The KeyPocket architecture consists of three main components: 1) the user entity, which is constituted of the user and a mobile device, 2) the relay-server, and 3) a browser-plugin.

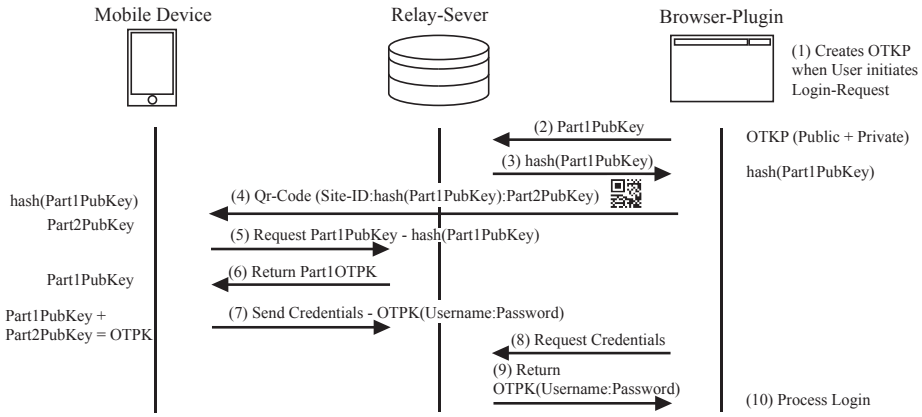

Fig. 1. Providing an overview across KeyPockets login process

Figure 1 illustrates the KeyPocket login concept in context of a flow chart. Core features of the architecture are its independence from third-parties, the decentralized user management as well as some unique security characteristics. E.g., the deployment of one-time key pairs and the commitment of a system for divided public key exchange. Instead of using a pre-shared key for encryption, an individual key pair is created on-demand for each login process. Multi-factor authentication becomes available due to knowledge (PIN), possession (the user's mobile device as a token) and being (e.g., analyzing the users fingerprint or voice). Another security factor is the proof of geographical proximity of the user to the device, which is about to become logged in due to QR-Code scanning. All network-based communication is secured by the usage of HTTPS and TLS.

4.2 Main components

There are three main entities, which are essential for our architectural concept.

User-side setup On the one hand, the user's smartphone is used as a storage for credentials, on the other hand its camera is needed for optical data transmission and its network connection for transferring data to the relay-server. Before the smartphone is ready to use, the KeyPocket application must be installed. Afterwards, no further registration is needed. This enables the user to manage data completely autonomous and on-device; privacy issues due to third-parties can be suspended. To ensure the secure storage of data, it is encrypted before saving and only available by entering a password or providing the correct biometric information.

Browser-side setup The browser-plugin is the architecture's control unit. Here, required one-time key pairs (OTKP) are generated, encoded and provided to the user on-demand. Moreover, the plugin is frequently polling at the relay-server for requested user credentials. As soon as they are available at the plugin, they become decrypted and filled into the form on the service provider's website. After confirming these values, the process is completed. The correct form fields are identified by unique Cascading Style Sheet (CSS) selectors.

Relay-Server The relay-server's main tasks are the forwarding of request and response calls (e.g., containing the users credentials) between the user's smartphone and the browser-plugin as well as the provision of a part of the one-time public key (OTPK). For realizing this connection, the relay-server supplies a Representational State Transfer (REST) interface.

4.3 Third-party independence and privacy enhancement

On the one hand, KeyPocket is a third-party independent system in terms of that there is no need to implement any code on the service provider's server. The service provider itself does not need to be aware of KeyPocket in order to make it work and due to its generic design it basically works with all web-based login sites without further conditions. On the other hand, the user downloads and installs the browser-plugin as well as the mobile application and is ready to use the architecture without any further registration processes. Moreover, no personal data, not even the user's email address for a registration is known to the KeyPocket infrastructure – all data is encrypted, managed and stored on-device. The relay-server only temporarily stores encrypted credentials until they are retrieved for processing a login.

4.4 Processing a login

The following process is visualized in Figure 1. A login is initiated by the user opening a service's login page in the browser. After entering a login page, the

user clicks the KeyPocket plugin symbol and the browser-plugin generates a new OTKP, containing a public key and a private key (1). A part of the public key (Part1PubKey) is forwarded to the relay-server (2). There it is stored temporarily and its hash is generated and sent back to the browser-plugin (3). There, a QR-Code is computed and shown for scanning by the user. The code contains three parameters: 1) the site ID, typically the domain name, 2) the hash of Part1PubKey and 3) the second part of the public key Part2PubKey (4). The smartphone now requests Part1PubKey from the relay-server (5). Due to the split public key system, it is impossible for attackers to obtain the complete public key and thereby is not able to channel malicious information into the system. Furthermore, a better readable and less granular QR-Code for easier scanning is created. As soon as the relay-server responds with Part1PubKey (6), the mobile device is able to merge the public key from its two fragments. By affirming the procedure due to provision of a password or a fingerprint, the user's credentials are decrypted from the device's storage. Subsequently, they become encrypted again with the one-time public key and transmitted to the relay-server (7). The browser-plugin frequently polls for the requested crendentials (8) and once they are available, they are transmitted to the plugin (9). The plugin encrypts the parameters with its private key, fills them into the designated form fields on the providers login page and confirms the procedure. If the user's data is valid, he is forwarded to the provider's individual welcome page. The presented approach is completely independent from third-party implementations and can generally be used for any kind of service.

4.5 On-device identity management

In order to avoid privacy issues as well as attacks onto a central database containing sensible data, KeyPocket resigns a cloud-based management of user data. All user data is encrypted, respectively, is stored decentralized on-device. Depending on technical features of the used device, personal data is only available by entering a password or by providing biometric features. As soon as credentials become requested, the application searches for corresponding data on the user's device, encrypts them with the OTPK and returns them. If no suitable data is found, the user can enter new credentials or link already existing account information. Due to the fact that the user's device is the only place where private data is stored, the migration to other devices is feasible due to credential export and import via encrypted database files, e.g., KeyPass' *.kdb*. There is no data distributed on a web server that needs to be updated.

An issue which is still unsolved within the current concept is a homogeneous sync process for changing an existing account's password on the smartphone as well as in a service provider's database. Currently, this procedure needs to be done manually by the user. The use of background HTTP requests for posting a new password entered on the user's smartphone to the provider's corresponding website could be a solution. Still, this approach lacks of a generic potential due to the need of knowing individual server addresses and parameter names.

Therefore, this issue still needs to be addressed by further research in order to solve it sufficiently.

4.6 Implementation

The current version of KeyPocket is available for Google Android and Apple iOS – general as well as further information can be found on the project's website[1]. A second version with some major changes is about to be released within the forth quarterly period of 2015. Some conceptual characteristics, e.g., the export of user credentials via database file, are not featured by the current KeyPocket version, but will be part of the next release. The following implementation details are coined to the currently published version.

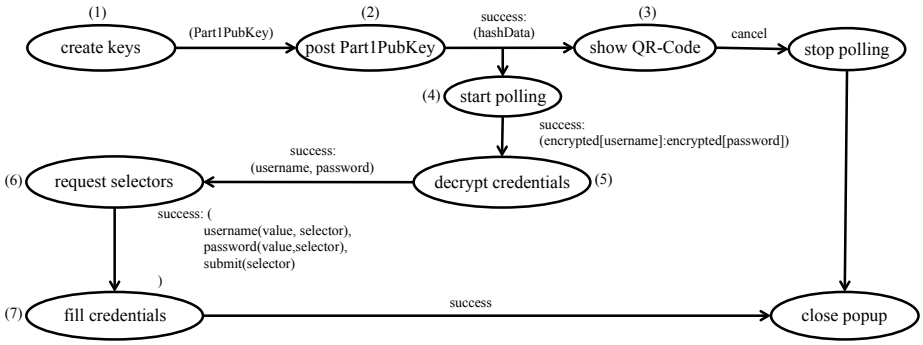

Fig. 2. Login process from the browser plugins point of view

Browser plugin Currently, browser-plugins for Google Chrome and Mozilla Firefox are provided. Both are implemented with JavaScript and individually optimized for the specific browsers. After their installation they are ready for usage without any further registration or setup process. The plugins' architecture is based on a message-oriented-architecture and Figure 4.6 shows the login process from a plugin's point of view. Therefore, an asymmetric key pair with a length of 2048 bit is created (1) with the Rivest Shamir Adleman (RSA) algorithm. In the following, the public key is split into two parts and one part is transferred to the relay server (2), which is responding with the corresponding hash of the public key part (3). The other part of the public key is shown to the user in the QR-Code. Subsequently, the plugin polls the server until it answers with the user's credentials or the login is terminated by the user (4). After the plugin has decrypted the credentials (5), it fills them into the selected login fields of a service's website (6,7). The identification of input fields is realized

[1] `https://www.keypocket.de/`

with the help of CSS selectors provided by an offline database. If a website's input fields are not registered in the database, the user is asked to mark the input fields manually with a mouse click. Unknown and new input fields become added automatically to the database after a successful login.

Relay server The relay-server is based on Java 1.8 and was implemented with the Play Framework[2]. It provides two REST interfaces, all data is JSON encoded and the connection is secured by TLS/HTTPS. The */pubkeys* interface is used to transfer one part of the public key to the server, which hashes the key and stores it temporarily in a database. The same interface also used for the data's retrieval. The */credentials* interface allows the temporarily storing and retrieval of the credentials, together with a corresponding hash. As soon as the credentials were transferred to the browser-plugin for the login, they become deleted automatically.

Smartphone application Within the current architectural concept, the user's smartphone is used to store all encrypted credentials, to transfer them to the relay server and for scanning the browser-plugin's QR-Code. Currently, Android and iOS devices are supported. Due to the fact, that not all Android devices are capable of using biometric data for securing the password vault, a master password is used as a fallback for devices without fingerprint sensor. In order to verify the password without storing it on device, a random salt with a size of 128 bit and SHA1PRNG is created first. Subsequently, it is hashed together with the user's password with 2000 iterations, 256 bit and PBKDF2WithHmacSHA1. The resulting hash is stored on device for password verification – a second hash key is created and stored with another salt and the users password, in order to use it for credential encryption. The credentials itself are encrypted with the AES algorithm with 256 bit in combination with HMAC. The creation of one hash key with $i = 1000$ iterations takes about $v = 0.15$ s, with $i = 2000$ iterations it takes about $v = 0.2$ s on a Samsung Galaxy S3 smartphone. The same amount of time is needed to decrypt the password later on for each decryption. This means, that even if an attacker had a CPU power of $f = 10000$ times faster than the user and the user's password an entropy of $n = 30$ bits, the attacker would still need about $\frac{v*2^{n-1}}{f} = 10.737$ hours to crack the password for $i = 2000$. In this context, the more iterations i are used, the more difficult it is for an attacker to learn a secret. On the other hand, especially due to limited resources on mobile devices it is necessary to find suitable parameters oriented on security as well as on usability.

For the iPhone implementation, we use the standard iOS Keychain in combination with a password or, if a fingerprint sensor is available, the user's fingerprint as a security feature.

[2] `https://www.playframework.com/`

5 Discussion

In this section we want to discuss the different features of the proposed KeyPocket architecture on a qualitative basis. Therefore, we first match its security characteristics to threats relevant in this context. The qualitative requirements, which we identified in Section 3 are the system's usability, modularity, compatibility, scalability, security, privacy protection, and its independence from external hardware as well as from third-party providers. Finally, some general issues which occurred during the implementation phase are mentioned.

5.1 Threat robustness

There are lots of different possibilities for attackers to compromise a user within the environment of mobile communication and authentication. In context of a **Man-in-the-Middle** attack, the attacker engages between two communicating parties and monitors their communication secretly. The so gained information can be modified, replaced or used for malicious purposes. Furthermore, the aggressor could disguise as the legitimate communication partner (e.g., a server) and response in this role to the other communication partner's requests in order thieve sensitive information [32]. Our application is not only robust to MitM attacks because of the usage of secured and fully encrypted communication channels. Furthermore, its split public key usage and the one-time key concept enhance the robustness significantly. Only in step (6) of Figure 1 an attacker is enabled to thieve a part of the key, which is useless without its counterpart. In general, **Brute-force** attacks can be aggrevated significantly by using sufficient passwords (e.g., sufficient length, special characters, capital and lowercase characters) [8]. Our architecture facilitates a completely user controlled data storing concept on the user's mobile device. Hence, the existing potential of Brute-force attacks is inevitable (e.g., on device theft). Nonetheless, as mentioned in Section 2, all data stored with the help of the iOS Keychain can be rated as relatively secure. For Android, we use a custom tailored encryption approach, which forces an attacker to invest several days or months in order to crack the user's password (accepting that the attacker has the CPU power of an up-to-date, high-end consumer machine and the user uses a password of sufficient complexity, see Section 4.6). After all, if no biometric sensor is available, the encryption security highly depends on the complexity of the user's password. The recording and readout of user input by the usage of malicious applications and without the user's knowledge is called **Keylogging** [33]. In general, our architecture offers no potential for Keylogging, because only the KeyPocket application is used for entering data. Still, if an attacker would be able to log the user's input while using KeyPocket (e.g., the OS security systems are compromised due to jail-breaking, rooting, etc.), the only potential for Keylogging attacks is present while the user enters his password or new account data. All already existing data is still safe due to its complete encryption. Furthermore, for doing harm with knowledge about the password, an attacker would still need the user's device. This also extends for **Shoulder-surfing** attacks, which means curious gazes across the user's shoulder

without his knowledge and in order to peek passwords or other sensitive data. Within the scope of a classic **Phishing** attack, the user becomes redirected to a fake website, which is based on a brands website the user knows and trusts in. After entering his credentials into a faked website's login form and confirming it, the user is redirected to a site of the original provider and is probably not aware of the attack or the resulting leak of private data. There are only a few possibilities to take actions against these attacks on a technical basis. For the most part it is up to the user to verify the authenticity of a website [34]. Related to that, Phishing attacks are also a weakness of our proposed system, for it relies on the user to check service addresses manually. But, although we cannot force a user to carefully check an unknown server address, he still needs to take notice of it and to confirm it manually. In context of **Sweep attacks**, an attacker is able to steal multiple credentials at the same time due to the exploitation of a password manager's autofill functions [35]. This danger is suspended due to the hold out on a clear signal of intention by the user before filling in credentials into input fields.

5.2 Qualitative requirements

In the following, we examine our architecture's features in respect of the requirements for an mobile-based authentication architecture. Concerning its **Usability**, KeyPocket offers an easy-to-use approach without ignoring the necessity for security standards – the only precondition for usage is the installation of the KeyPocket software. For logging in, there are only two steps: 1) scanning the QR-Code, and 2) confirming the login by providing a second security factor. Due to its need for only one password to be remembered by the user or the usage of biometric data for opening the credential's vault, the simplicity and usability is enhanced significantly for the user. In contrast to some of the approaches introduced in Section 2, KeyPocket facilitates no device pairing, which enables multi-device usage. Missing multi-device support is mostly related to the necessity for a registration process, an existing provider dependence, or pre-shared keys bound to a specific service or device. In order to guarantee **Modularity, Compatibility, and Scalability**, new and existing user accounts can be added to the application without restrictions and there are no limitations for their number or the assignment of accounts to domains and vice versa. A disadvantage of the KeyPocket architecture may be its need for a relay-server, which could be seen as a Single-point-of-failure or, in case of a large amount of login requests, could lead to scalability problems. Concerning **Hardware Independence**, there are no preconditions except the need for a smartphone with internet and a camera in order to use KeyPocket. Compared to other approaches, no additional tokens (e.g., Smartcards or ID cards) are used. As already mentioned, no cloud-systems or provider dependent storage is used in order to save and manage the user's credentials. There is no registration process and no information about the user within the system except the encrypted data on the smartphone and a temporary copy of credential pairs during the login process on the relay-server. But even this temporary copy is deleted after a few seconds. All of these routines

and practices are strengthening the users **Privacy** significantly. Furthermore, **Third-Party Independence** is strengthened due to the fact that no service providers or third-parties are involved in the handling of confidential data and no registration process is needed. Furthermore, there is no code about to be deployed on the service provider's systems. The user solely decides, where and when to use the system without our or the service provider's knowledge.

5.3 General notes

In the following, some specific details concerning KeyPocket are to be mentioned. Our generic plugin-based approach enables the adaptability for nearly each service provider featuring a website login without the providers inclusion. However, due to the necessity of using selectors for identifying form fields, the system's reliability can be weakened because of external influences (e.g., updates of provider sites, changes in technology, etc.). This may also lead to increased costs due to maintenance work. Nonetheless, this problem is addressed due to the feature of allowing users to mark unknown input fields manually and to use these information to update our database constantly. Furthermore, due to technical reasons the generation of on-demand key pairs took nearly 6 seconds with Google Chrome in a worst case scenario. This is way to much for a system with high usability requirements. In order to solve this problem, we changed our initial protocol for key creation and create the first key pair already on browser startup. Implying that the user is not logging in into different services with a frequency higher than each 6 seconds, we found this optimization to be sufficient. This pre-usage creation enables the system to simulate an on-demand usage experience.

6 Conclusion and future work

Within the scope of this paper, we provided a secured, and third-party independent platform for processing logins on websites under inclusion of the user's smartphone. The ability to manage digital identities without enforcing an additional registration process or effectuating privacy issues is provided to the user. Additionally, we introduced some unique features, namely the usage of a split one-time public key in combination with further security factors like possession, being and geographical proximity.

Another issue which is still unsolved is the development of a holistic process for updating and synchronizing account information automatically. Currently, it is the user's duty to do this manually.

Despite our efforts, there are still open issues as well as possibilities for extensions. For example, scanning of QR-Codes has performance issues due to technical reasons and furthermore, it lacks of user acceptance. It could be substituted with technologies featuring similar security aspects (e.g., geographical proximity, protection against eavesdropping, etc.) in this context, e.g., NFC or

BLE. This could also increase the system's usability due to the disposal of the additional scanning interaction and is subject of our research for future KeyPocket versions.

References

[1] Birgit Van Eimeren. Always on - smartphone, tablet und co. als neue taktgeber im netz (ard/zdf). *Media Perspektiven*, 7(2013):386–390, 2013.
[2] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.
[3] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th international conference on World Wide Web*, pages 657–666. ACM, 2007.
[4] Shirley Gaw and Edward W Felten. Password management strategies for online accounts. In *Proceedings of the second symposium on Usable privacy and security*, pages 44–55. ACM, 2006.
[5] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.
[6] Tarun Parwani, Ramin Kholoussi, and Panagiotis Karras. How to hack into facebook without being a hacker. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 751–754. International World Wide Web Conferences Steering Committee, 2013.
[7] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 553–567. IEEE, 2012.
[8] Ken Munro. Android scraping: accessing personal data on mobile devices. *Network Security*, 2014(11):5–9, 2014.
[9] Android 5.0 Encryption. `https://source.android.com/devices/tech/security/encryption/`, 2015. [Online; accessed 20-January-2015].
[10] Jens Heider and Matthias Boll. ios keychain weakness faq. *Fraunhofer Institute for Secure Technology*, 2011.
[11] Eric Rescorla. Rfc 2818: Http over tls. *Internet Engineering Task Force*, 2000.
[12] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X. 509 internet public key infrastructure online certificate status protocol. *IETF RFC2560, June*, 1999.
[13] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: validating ssl certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 38–49. ACM, 2012.
[14] Zakir Durumeric, James Kasten, David Adrian, J Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Math-

ias Payer, et al. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 475–488. ACM, 2014.

[15] Nektarios Georgios Tsoutsos and Michail Maniatakos. Trust no one: Thwarting" heartbleed" attacks using privacy-preserving computation. In *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, pages 59–64. IEEE, 2014.

[16] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security and Privacy*, 7(1):78–81, 2009.

[17] Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan Wallach, and Dirk Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 404–414. ACM, 2012.

[18] B Borchert. Ekaay-smart login. *http://www. ekaay. com/*, 2013.

[19] eKaay Smart Login System. `http://www.ekaay.com/`, 2015. [Online; accessed 14-January-2015].

[20] Roland M Van Rijswijk and Joost Van Dijk. Tiqr: A novel take on two-factor authentication. In *LISA*, 2011.

[21] Ben Dodson, Debangsu Sengupta, Dan Boneh, and Monica S. Lam. Snap2pass: Consumer-friendly challenge-response authentication with a phone. *Stanford University*, 2010.

[22] Galois QR Authentication. `http://galois.com/blog/2011/01/quick-authentication-using-mobile-devices-and-qr-codes/`, 2015. [Online; accessed 19-January-2015].

[23] Jörg Schieb. *Schieb. de Wissen— Das sichere Login: So haben Hacker keine Chance, Pages 42-44*, volume 1. Jörg Schieb, 2014.

[24] Click2Pass Handy statt Passwort. `http://www.click2pass.net/`, 2015. [Online; accessed 14-January-2015].

[25] Next Authentication and Authorization Plattform. `https://launchkey.com/platform/mobile/`, 2015. [Online; accessed 14-January-2015].

[26] Zapper. `https://www.zapper.com/about.php/`, 2015. [Online; accessed 19-January-2015].

[27] CLEF Secure Two Factor Login. `https://getclef.com/features/`, 2015. [Online; accessed 19-January-2015].

[28] M Jones and D Hardt. The oauth 2.0 authorization framework: Bearer token usage. Technical report, RFC 6750, October, 2012.

[29] OneID. `https://www.oneid.com/`, 2015. [Online; accessed 03-September-2015].

[30] KnockToUnlock. `http://www.knocktounlock.com/`, 2015. [Online; accessed 03-September-2015].

[31] BlueID. `https://www.blueid.net/`, 2015. [Online; accessed 03-September-2015].

[32] Nadarajah Asokan, Valtteri Niemi, and Kaisa Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *Security Protocols*, pages 28–41. Springer, 2005.

[33] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 3–14. ACM, 2011.

[34] Jason Hong. The state of phishing attacks. *Communications of the ACM*, 55(1):74–81, 2012.

[35] David Silver, Suman Jana, Eric Chen, Collin Jackson, and Dan Boneh. Password managers: Attacks and defenses. In *Proceedings of the 23rd Usenix Security Symposium*, 2014.