OpenKeychain: An Architecture for Cryptography with Smart Cards and NFC Rings on Android

DOMINIK SCHÜRMANN, TU Braunschweig, Germany
SERGEJ DECHAND, University of Bonn, Germany
LARS WOLF, TU Braunschweig, Germany

While many Android apps provide end-to-end encryption, the cryptographic keys are still stored on the device itself and can thus be stolen by exploiting vulnerabilities. External cryptographic hardware solves this issue, but is currently only used for two-factor authentication and not for communication encryption.

In this paper, we design, implement, and evaluate an architecture for NFC-based cryptography on Android. Our high-level API provides cryptographic operations without requiring knowledge of public-key cryptography. By developing OpenKeychain, we were able to roll out this architecture for more than 100,000 users. It provides encryption for emails, messaging, and a password manager. We provide a threat model, NFC performance measurements, and discuss their impact on our architecture design. As an alternative form factor to smart cards, we created the prototype of an NFC signet ring. To evaluate the UI components and form factors, a lab study with 40 participants at a large company has been conducted. We measured the time required by the participants to set up the system and reply to encrypted emails. These measurements and a subsequent interview indicate that our NFC-based solutions are more user friendly in comparison to traditional password-protected keys.

CCS Concepts: • **Security and privacy** → **Usability in security and privacy**; *Key management*; *Hardware-based security protocols*; • **Human-centered computing** → *Mobile devices*;

Additional Key Words and Phrases: NFC, near-field communication, smart card, ring

ACM Reference Format:
Dominik Schürmann, Sergej Dechand, and Lars Wolf. 2017. OpenKeychain: An Architecture for Cryptography with Smart Cards and NFC Rings on Android. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3, Article 99 (September 2017), 24 pages.
https://doi.org/10.1145/3130964

1 INTRODUCTION

Nowadays, it is widely acknowledged that smartphones hold the most sensitive information, such as emails, short messages, and photos. Besides private usage, the same devices are often used for accessing privileged company information due to Bring Your Own Device (BYOD) policies. Privacy-aware app developers take this into account and provide apps for secure messaging, encrypted cloud storage, and other use cases. Unfortunately, secret keys generated by these apps are unprotected and stored on the internal flash memory. Thus, an attacker can fully compromise end-to-end security by retrieving secret keys through privilege escalation exploits or direct physical access. While this is a well-known problem, the security of many apps and the mobile operating system is subpar [82]. Since the Stagefright-bug [85], Google started rolling out monthly Over-The-Air (OTA)

2474-9567/2017/9-ART99 $15
https://doi.org/10.1145/3130964

updates [53] documented in Nexus Security Bulletins [4]. Unfortunately, not all fixes are backported for devices of other Original Equipment Manufacturers (OEMs). Even if the system is up-to-date, widespread vulnerabilities in secure email and messaging apps have been found, potentially exposing secret keys to attackers [78, 82]. Even worse, app developers often lack knowledge of cryptography, which leads to insecure implementations [25, 29].

Traditional desktop encryption software provides protection against key theft by encrypting keys using high-entropy passwords. One of the main reasons this is not done on mobile devices is due to the difficulty of entering passwords on mobile on-screen keyboards. One often recommended password alternative are biometric fingerprints. These can be used since Android 6 with the Keystore API. While they are suitable for locking devices, they should not be used alone to protect secret keys. They do not provide enough entropy to reach the same strength as passwords or hardware-backed solutions [49, 56].

For desktop systems, external hardware in form of smart cards exist which can be used with USB card readers. They replace password-protected key files with external cards and corresponding PINs. In contrast to traditional key files, which are stored on the same device where the password is entered, the secret key of a smart card is stored physically separated from the device where the PIN is entered. For smartphones, On-The-Go USB cables or other external peripherals, such as hardcovers with card slots, exist. Because these are unwieldy and, thus, unsuitable for day-to-day usage, Near-Field Communication (NFC) has been proposed for smart card communication. A small selection of NFC tokens, which are capable of encryption and signature generation and not only authentication is available. Still, no high-level cryptographic NFC API was available.

1.1 Contributions

In this paper, we introduce and evaluate an architecture for NFC-based cryptography on Android via external security tokens, such as smart cards. First, we discuss the problem setting by looking into currently available NFC hardware and software libraries to derive a clear set of requirements. In the threat model we consider for this work, three main attack areas have been identified: NFC, security token, and smartphone. All attacks are evaluated in comparison to traditional password-protected secret keys. Based on our requirements and threat model, we carefully design our architecture and, thus, make the following contributions in this paper:

(1) Our main contribution is the design of a high-level API for Android, which not only includes cryptographic primitives but also a variety of pre-defined UI components for common end-user interactions. Previously available cryptographic NFC APIs have been designed for authentication only.

(2) For developers, we provide a set of API methods, which can be used without knowledge of public-key cryptography. Keys are handled transparently independent of their storage location (password-protected key file or via NFC). In addition to the card's PIN authentication, we provide a security layer via an Android app that handles the PIN input and allows only certain apps access to cryptographic methods, which have been explicitly granted by the user. This way, under the assumptions of Android's security model, only one trusted app handles the PIN input, while many semi-trusted apps can access cryptographic methods.

(3) As one of the main developers of *OpenKeychain*, an encryption app for Android, we were able to roll out this architecture for more than 100,000 devices. Its API is used by several other apps, such as *K-9 Mail*, the XMPP client *Conversations*, and the password manager *Password Store*.

(4) As an alternative form factor, we consider a NFC ring. Because no NFC rings with a cryptographic processor are available, we created our own prototype using an NXP Integrated Circuit (IC) and a 3D printer.

(5) To evaluate the end-user usability of our architecture in combination with password-protected keys, NFC cards, and NFC rings, we conducted a usability lab study with 40 participants outside of the university environment in cooperation with the IT security department of a large company. As a use case scenario we implemented end-to-end encrypted email communication. Finally, an interview has been conducted to evaluate user perception and form factor acceptance.

2 RELATED WORK

The work presented in this paper touches the research areas of (a) usability of end-to-end encryption, (b) cryptographic API design, and (c) usability of security tokens, especially in conjunction with smartphones.

While our API supports a wide range of differing use cases, end-to-end email encryption is one prominent example, which we also included as an experiment in our study. The famous "Why Johnny Can't Encrypt" [81] publication discusses a case study on PGP 5.0 and concluded that end-to-end email encryption software was still not usable enough for most end-users. This led to several user studies over the few last years, e.g., by Garfinkel et al. [35, 36] and Fahl et al. [30]. They studied larger groups and proposed how email encryption or Facebook encryption can be made usable.

While there is a broad range of research on usable security for end-users, developer usability is often neglected. It has been shown that in a set of 12000 apps at least 88 % include at least one cryptographic error [25]. Similarly, developers often override safe defaults due to being unaware of the implications as shown in the evaluation of SSL apps by Fahl et al. [31]. To prevent these incidents from happening, high-level cryptographic APIs, such as NaCl [7], Sodium [20], and Keyczar [21], have been published in the last years. High-level 'crypto-box'-methods, based on a fixed set of algorithms, are provided that execute several steps at once, which are usually done individually when using low-level APIs such as Java Cryptography Extension (JCE) or Bouncy Castle [11]. Still, these libraries provide no automated way of handling password/PIN input and are not designed to fit into the developer ecosystem of mobile operating systems such as Android.

Research on the usability of security tokens for asymmetric cryptography in the context of encryption of emails or instant messages is limited. Most publications focus on authentication not encryption with security tokens, in particular smart cards. Sasse concludes that smart cards can offer usability benefits compared to password authentication [75]. Strouble et al. conducted a survey, answered by 300 participants, evaluating the usability of smart cards [79]. A notable result is that 67 % left their smart cards in the USB reader at least once, which increases the possibility of theft. Paul et al. conducted a field study with 24 participants over 10 weeks to evaluate the user behavior and perceptions in regards to smart cards [67]. They conclude that "The greatest perceived benefit was the use of an easy-to-remember PIN in replacement of complicated passwords. The greatest perceived drawback was the lack of smartcard-supported applications" [67]. Recently, Google compared their 'Security Keys' with other hardware tokens, passwords alone, and two factor authentications using smartphone apps [52] using the usability framework by Bonneau et al. [10]. They measured the raw performance and found out that users are twice as fast authenticating via 'Security Keys' in comparison to in-app One Time Passwords. Failure rates dropped from 3 % with OTP to 0 %. Taking into account the burden of physically carrying around authentication hardware, Mare et al. found that participants greatly differ in their preference regarding form factors [54]. To the best of the authors' knowledge, no studies are considering the usage of NFC security tokens, in our case NFC cards and rings, for end-to-end encryption instead of authentication on smartphones.

Available security tokens, which are qualified for our architecture, must include a cryptographic processor for asymmetric operations, such as RSA or ECC, and a NFC interface. For desktop systems with USB or card readers, tokens such as NitroKey [62], variants of YubiKey [83] (w/ OpenPGP support), and Java Card OpenPlatform (JCOP) smart cards with ICs by NXP are available. A number of smartphone accessories exist, e.g., back covers like the BlackBerry Smart Card Reader [72], modern variants like the Smart Card Reader by Precise Biometrics [69] (∼100 EUR), and the Smart Fold Android Contact Smart Card Reader [46]. Due to the cost and bulkiness of external peripherals, NFC security tokens are considered. Only a limited number of the presented ones have an NFC interface, such as YubiKey NEO [83], the Fidesmo Card [32], and NXP developer cards with dual interface.

In addition to evaluating the traditional card form factor, we consider the usage of wearable NFC rings. A first example from 1998 is the Java Ring by Dallas Semiconductor [18] running an early version of the Java Card environment. During the following years, this form factor has been proposed for different communication scenarios [16, 60, 71, 73]. A famous commercially available NFC ring has been designed by McLear and marketed via Kickstarter [58]. To the best of the authors' knowledge, no consumer-ready NFC ring supporting asymmetric cryptography over NFC together with an implementation was available until the first author started the work outlined in this paper.

3 PROBLEM SETTING

As discussed in the introduction, important secret keys should not be kept on internal flash storage as they can be stolen using widely available exploits. Instead, it is good practice to store those on external security tokens. In this section, we discuss related security and usability problems to derive a set of requirements for our architecture.

Cryptographic operations should be executed via a connection between smartphone and token, while the secret key itself should never be exposed to the smartphone. This connection is traditionally established via card readers, which require On-The-Go USB cables or external peripherals to work with modern smartphones. Also, smart cards are often left inside the reader [79], which poses a security risk. Communication over NFC has been proposed as a mobile alternative but no API is available for encryption/digital signature generation. Even though a small selection of security tokens with NFC and cryptographic processor are available, these are designed as credit cards or USB sticks. While other form factors have been proposed for simple NFC tags with read/write capability, these are not available with cryptographic processors. Furthermore, while the usage of security tokens has been evaluated for two-factor authentication, no studies exist in the context of end-to-end encryption. Thus, it is not known how users perceive the usage of NFC in the context of email encryption for example. It is also not known if other form factors could improve the acceptance of tokens or their usability.

When using NFC, security tokens must be held against the device's NFC antenna for differing durations depending on the cryptographic operation. Users can easily get frustrated if these are too long. Access control to operations should be done by entering a numeric PIN on the device that is easy to remember. To restrict brute force attempts, the token should deactivate after a number of failed attempts and a special Admin PIN must allow the user to re-activate the token. In available implementations for desktop operating systems with card readers, the users are not properly guided through the selection of an appropriate PIN and Admin PIN. For example, in GnuPG [26] and Enigmail [14] users are not forced to change the default PINs after key generation.

Not all apps should have access to the PIN to execute arbitrary cryptographic operations while the security token is held against the NFC antenna. Also, in traditional desktop implementations, it is not possible to restrict the execution of cryptographic operations to a specific set of client applications. Thus, client applications often handle PIN input directly, even though they may not be trusted fully. Furthermore, usage of secret keys is not restricted in any way, as long as a contact smart card is inserted into the reader. Secret keys for special purposes or higher levels of classification are not protected differently than other keys. Currently, no high-level API exists for smartphones that supports cryptographic operations, but also securely handles PIN/password caching and provides user interaction for common functionality, such as public key import. All available low-level APIs integrated in mobile operating systems such as Java's Cipher API [2] on Android, whose internals are based on Bouncy Castle/OpenSSL, as well as iOS Cryptographic Services [5] require a substantial effort from the developer. This naturally leads to vulnerabilities implemented over and over in many apps as well as to re-implementations of the same functionality in different contexts, such as password input as well as caching layers. Furthermore, they do not provide re-usable UI components. Even high-level APIs, such as NaCl [7], Sodium [20], and Keyczar [21], require a substantial amount of app-specific code to handle password input and caching, migrations from older key algorithms, and user to key mappings.

In addition, these are designed without special hardware in mind, i.e., no external security tokens are supported out of the box. Thus, keys stored on security tokens must be handled completely different and require more complexity than keys stored on the device, e.g., when using Android APIs [27].

3.1 Requirements

Based on the outlined problem setting, a set of architectural requirements is derived:

R_1 Support for multiple form factors without external smartphone accessories
R_2 Short durations of cryptographic NFC operations
R_3 PIN input and caching solely handled by a trusted entity on the smartphone
R_4 High-level versionable cryptographic API including UI components for common user interactions, secure defaults, and standardized packet formats
R_5 Access control on the token-side by numeric PIN/Admin PIN and on the app-side by restriction of secret keys to specific clients
R_6 Transparent handling of secret keys independent of their storage location

3.2 Threat Model

The main advantage of security tokens is that its key storage is physically separated from the mobile device. The hardware and firmware of the security token does not provide an API to retrieve secret keys, only cryptographic operations are exposed executed on the processor of the token. In the following, the discussed threat model is subdivided by the attacked entity, i.e., NFC, security token, and smartphone. While we evaluate all scenarios relevant for the whole architecture, an emphasis lies on mitigations provided by our own contributions.

3.2.1 NFC. First, we discuss attacks against the NFC connection itself that is established between the smartphone as an active initiator and the security token as a passive target.

Denial of Service Since radio jamming in general is difficult to prevent, NFC lacks sophisticated countermeasures. However, simply preventing communication is of low value to an attacker. In particular, signing and decrypting emails is not a time-critical activity and can, thus, tolerate short-term disruptions. More importantly, this attack does not put the security of the secret key at risk.

Relay Attack This attack is also called a wormhole or Mafia attack. In the field of NFC payment and authentication systems, a connection is established between the victim's smart card and an attacker's NFC reader. This connection is relayed over the Internet to a second device of the attacker to actually authenticate or pay at a different NFC reader physically far away [34]. This attack can potentially be executed unnoticed by holding the attacker's NFC reader against the victim's pocket containing the smart card. In our architecture, the security token is protected by a PIN. Thus, NFC relay attacks can only be executed if the PIN has been compromised beforehand.

Eavesdropping Kortvedt and Mjolsnes were able to eavesdrop on NFC in a range of up to 29 cm [51]. Brown et al. experimentally showed that eavesdropping capabilities largely depend on the amount of background noise [12]. Thus, if the attacker is very close to the victim and has the required equipment, it might be possible to extract the following information: In case of signature generation, a hash is transmitted and a signed hash is received. In case of decryption, an encrypted session key packet is transmitted and the decrypted session key is received. It is important to note that the plaintext that should be signed or the ciphertext that should be decrypted is never transmitted. Furthermore, the secret key never leaves the security token. Therefore, to decrypt an email with an eavesdropped session key, the encrypted email must also be intercepted at the corresponding email provider. Still, this is a valid attack scenario against targeted individuals. While our current prototype assumes channel security, for a future version we consider deploying the NFC-SEC standard [23, 24] that provides an Elliptic Curve Diffie Hellman key exchange with AES encryption. An application-level alternative for securing NFC has been proposed by Hölzl et al. using the Secure Remote Password (SRP-6a) protocol authenticated by a user-provided password [45].

Man-in-the-Middle (MitM) While eavesdropping might be possible in a certain range, MitM are extremely difficult because the attacker needs to block existing NFC. In detail, these attacks differ depending on the type of NFC connection: With an active-passive or passive-active connection, an attacker has to both block the originator's channel and to create an own RF field with perfect timing [42]. This is hard to achieve in practice and can possibly be detected by the user. In case of active-active connections, the interceptor has to completely block the communication between both partners without them noticing. Usually, NFC should abort if two RF fields exists, but is has been shown for the EMV protocol that some implementations do not follow the standard and it is possible to win the timing race [59]. Yet, this has not been shown for other protocols besides EMV. Conclusively, while Haselsteiner and Breitfuß [42] consider MitM attacks practically impossible, we at least consider them extremely difficult.

3.2.2 Security Token. In general, access to the security token is protected by a PIN with a length of 6 digits to protect against attackers in physical proximity. Thus, every cryptographic operation requires an authentication step. For the remaining attacks, we assume that the security token is protected against manipulations until it is received by the end user. Hence, the manufacturing process, warehousing, and shipping are considered to be secure, such that the initial key generation by the user is uncompromised. From this point on, though, the security token is vulnerable to theft and loss. As stated in the Introduction, there is no pre-deployed secret key on the security token, but the keys are generated by the user. In the following, we will mainly focus on attacks against the authentication step and hardware.

Brute Force PIN For memorability, we let the user chose the PIN, but prevent certain commonly chosen combinations, such as 123456. An attacker gaining access to the device can brute force up to 3 possible combinations of the PIN. After this, the security token is locked to prevent further brute forcing, and can only be unlocked entering the Admin PIN. In our architecture, the Admin PIN is not chosen by the user, instead it is securely-generated from random.

Physical attacks Due to theft or while the owner leaves the token unattended, an attacker can gain access to the security token. Physical attacks [80] aim to read, to modify or to erase data on the security token. Examples are provoking a power outage, examination with a probe station, chip re-wiring, as well as addition and cutting of a track. Given the physical access to the security token, these are generally difficult to defeat completely. Yet, they are typically expensive, destructive, and time consuming, especially since attacks are very target dependent. Additional protections against physical attacks, such as additional metal layers, bus scrambling, or on-board sensors can also be implemented on the hardware side. For these countermeasures, we rely on the security of the utilized NXP IC.

Side-Channel Attacks While being in close proximity to the owner, who currently uses the token with her smartphone, information about the cryptographic operation can be leaked by the token and smartphone. Timing attacks, for instance, exploit that the computing time of an operation differs with the used parameters, which in turn, can then be derived. As with physical attacks, we rely on the countermeasures provided by the IC. As discussed for an attacker who eavesdrop on NFC communication, also for side-channel attacks against the hardware, it could provide an additional advantage to monitor the corresponding email communication to correlate the decryption process with a particular message.

3.2.3 Smartphone. Recent vulnerabilities, such as the Stagefright bug [85], show the limited security on mobile devices. While local and remote software/firmware vulnerabilities are considered, we assume state-of-the-art cryptographic algorithms to be secure.

Physical Access An attacker, who gains physical access to a smartphone, while the owner leaves it unattended, is assumed to be able to download all data. The secret key, however, is never stored on the phone and, thus, not at risk. If no sophisticated attacks are performed, such as flashing a whole new operating

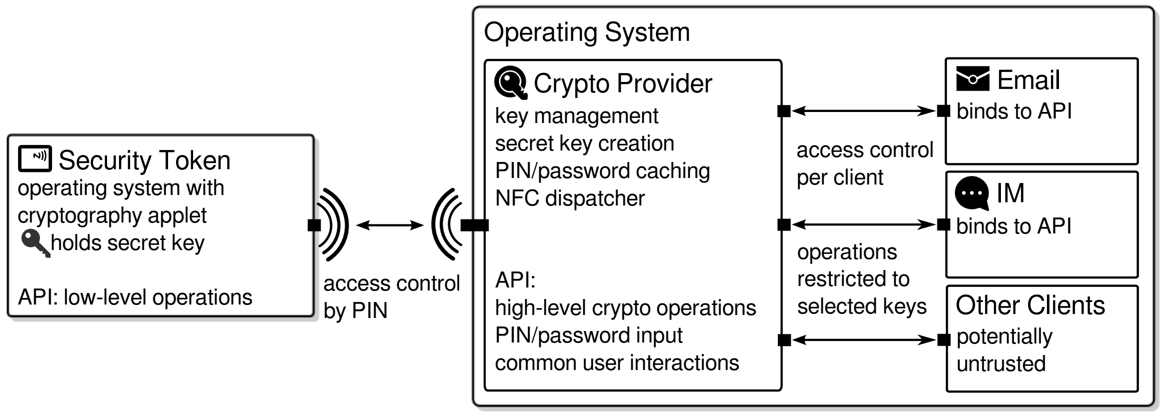


Fig. 1. Architecture Overview.

system, Android's system security prevents the installation of malware with root access. Still malicious apps with normal privileges could be installed. Due to the fact that PINs/passwords are only entered via a single trusted cryptography provider, malicious apps without root access cannot intercept these.

Vulnerabilities in Client Apps An attacker can try to exploit vulnerabilities in client apps that use our API. As described before, passwords and PINs cannot directly be retrieved as these are handled by the trusted cryptography provider only. Some vulnerabilities could potentially allow the attacker to trick the user into decrypting different content than originally opened by the user. In this case the password/PIN is properly entered by the user and the attack succeeds. To reduce the privacy impact in this scenario, a client app's API access is restricted to specific secret keys. Thus, only the keys selected for this client can be misused by an attacker.

Vulnerabilities in Android Critical vulnerabilities in Android can lead to exploits being used by an attacker to install malware with system access. In this case all installed apps are potentially insecure, even the PIN/password input and caching. Still, after detecting such a breach and removing the malware, future communications are secure, because the attacker was not able to retrieve the secret key.

UI Spoofing/Task Hijacking A malicious app could start a PIN/password dialog overlaying the original one and mimicking its design to intercept user secrets. This could be done by installing malware or a patched version of our cryptography provider. More sophisticated attacks building upon this scenario exploiting Android specific mechanisms are discussed by Cooley et al. [17] and Ren et al. [70]. As a future countermeasure we are plan to allow the user to set a personal image that will appear in all trustworthy UI components of our architecture. This has been implemented for example by Mailvelope [55].

GUI Side Channel Attacks As a special subcategory of attacks on Android, side channel attacks on the GUI of Android apps and hardware interrupts can potentially leak PINs/passwords to an attacker [15, 22]. As Diao et al. [22] remark, these side channels need to be closed on a system level by providing less runtime statistics to installed apps.

4 ARCHITECTURE

Considering the requirements and threat model, we propose a security architecture as depicted schematically in Figure 1. A security token, e.g., in credit card format, runs a smart card operating system together with an implementation of the cryptographic operations. The user's secret key is stored solely on this external token. It receives power via induction while holding it against the device and communicates with the operating system over NFC. The device's NFC interface is standardized and, thus, works with security tokens of multiple form factors (R_1). The token's API is protected via PIN authentication and provides all required cryptographic operations such as key generation, signature creation, and decryption. A single trusted cryptography provider is installed as an app on the smartphone's operating system. It implements all required low-level cryptographic operations as well as communication over NFC, optimized for short durations (R_2) and usability. PIN/password input and caching is done solely by this trusted cryptography provider (R_3). In addition to PIN/password input, several other common user interactions are supported to prevent re-implementations of the same interactions in different clients and decrease implementation complexity for client developers. An API is exposed to client developers providing high-level versioned cryptographic methods, e.g., a method for encryption combined with signatures, in the standardized OpenPGP message format (R_4). Access to this API is granted per app by user choice (R_5). The cryptography provider provides a key generation wizard that includes a secure selection of PIN/Admin PIN. Furthermore, it provides a unified way to transparently use secret keys without exposing their storage location or asymmetric algorithms (R_6).

4.1 Prerequisites

Our architecture is based on several existing technologies that are discussed briefly as prerequisites.

OpenPGP To integrate with existing protocols, the OpenPGP standard [13] has been chosen. It provides standardized email [28] and instant messaging [76, 77] encryption. Thus, it supports most common use cases with extensions for standardized communication protocols. OpenPGP support for smart cards has been standardized for ISO-compatible card operating systems [68] and primarily three open source implementations exist [33, 61, 84].

NFC Typically operating at 13.56 MHz, NFC is a wireless transmission technology for short ranges allowing active-active and active-passive modes. In our case, an active-passive connection is established, where the smartphone serves as the initiator and the security token is the passive target. The ISO 14443-4 [47] standard is used as the physical/link layer protocol between initiator and target and ISO 7816-4 [48] defines the basic structure of commands and Application Protocol Data Units (APDUs).

Operating System Support NFC support in Apple's current iOS 9 only supports NFC store loyalty cards as part of the Apple Pay API [6] and Windows Phone has only limited support for smart cards [1]. In contrast, Android's NFC API allows to exchange APDUs. It supports a foreground dispatch mode ($\geq$ Android 2.3.3) and reader mode ($\geq$ Android 4.4) that allow an app to manage the NFC connection without interfering with other installed NFC apps [3].

4.2 API Design

The presented architecture has been fully implemented for the Android operating system as part of OpenKeychain [65], an app implementing the OpenPGP standard. Currently, OpenKeychain has over 100,000 installations on Google Play and is also available via alternative stores, such as F-Droid. Besides the API proposed in this paper, OpenKeychain also provides encryption/decryption as well as signature generation/verification functionality of messages and files within the app. Since October 2015, a version has been released that was audited externally [43].

The following design provides a high-level API that complies with all requirements and can be used by other installed apps in a convenient way: In agreement with the OpenIntents project [64], the API definition lives in the namespace "org.openintents.openpgp". In contrast to similar architectures like JCE, these can also be chosen at runtime via app settings, i.e., the cryptographic backend for an email app can be provided by multiple implementations of the same high-level API.

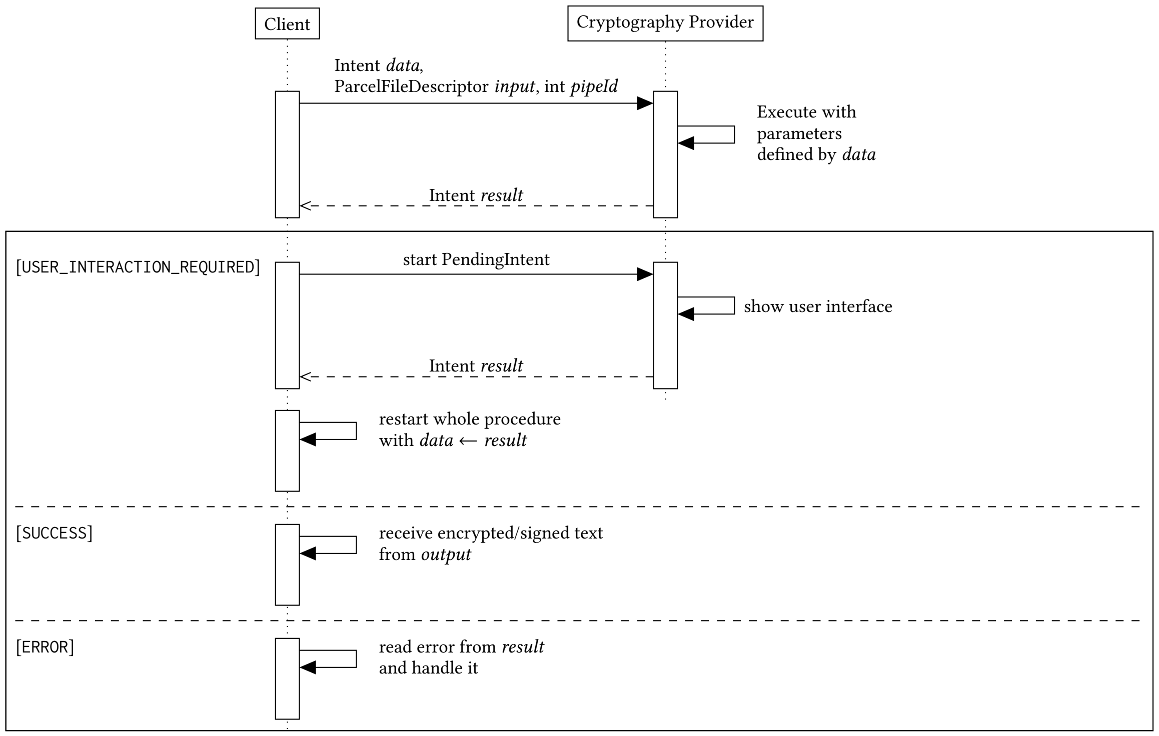
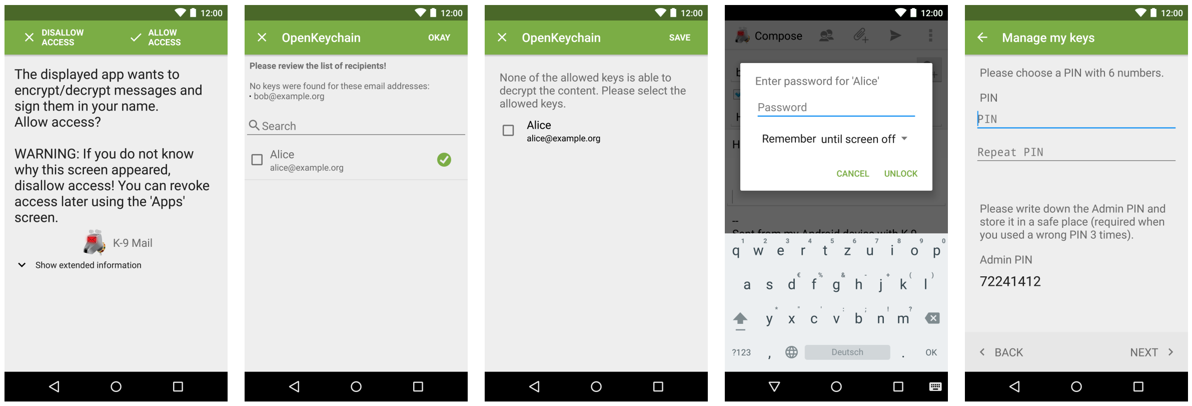


Fig. 2. Control flow of the proposed API.

Instead of providing an API via exported Android Activities, the API has been defined with the Android Interface Definition Language (AIDL). This allows for streaming of larger content via file descriptors. While this allows for performant access to shared memory between two apps, the immutable definition of the methods' type signatures makes it difficult to maintain backward compatibility with API definitions included in older client apps. Thus, instead of defining the method name and parameters directly as part of the type signature, these are defined using an Intent with a specific action (method name) and extras (method parameters), which are well-known to Android developers. Using Intents allows for easier backward compatibility and more flexible method definitions, which are not constrained to a specific parameter combination. To satisfy the requirement of backward compatibility, these are made versionable by including a version field together with a size calculated over the remaining fields at the first position when flattening the object for serialization.

Following Figure 2, after binding to the service, a client can execute a remote method. If a parameter is not specific enough or a required parameter is missing that can be provided by the user, the operation is canceled and the `USER_INTERACTION_REQUIRED` result code is returned together with a immutable PendingIntent. This PendingIntent can be started by the client at an appropriate time and is executed in the cryptography provider's process sandbox to handle interaction using appropriate UI components. One common use case is that a public key is missing for a given email address and must be downloaded. After user interactions, the operation is executed again with the parameters from the first execution combined with those retrieved from user input via the *result* Intent. The client app holds all parameters and decides by itself at which point in its control flow to


The displayed app wants to encrypt/decrypt messages and sign them in your name. Allow access?

WARNING: If you do not know why this screen appeared, disallow access! You can revoke access later using the 'Apps' screen.

Please review the list of recipients!

No keys were found for these email addresses:
• bob@example.org

None of the allowed keys is able to decrypt the content. Please select the allowed keys.

Enter password for 'Alice'

Remember until screen off

Please choose a PIN with 6 numbers.

Please write down the Admin PIN and store it in a safe place (required when you used a wrong PIN 3 times).

Admin PIN
72241412

(a) Access control per app via user decision. (b) Missing public key. (c) Restriction of allowed keys per app. (d) Password input for password-protected keys. (e) PIN selection during key creation.

Fig. 3. A selection of UI components shown via PendingIntent for the USER_INTERACTION_REQUIRED state.

Table 1. API specification. No understanding of public-key cryptography is required.

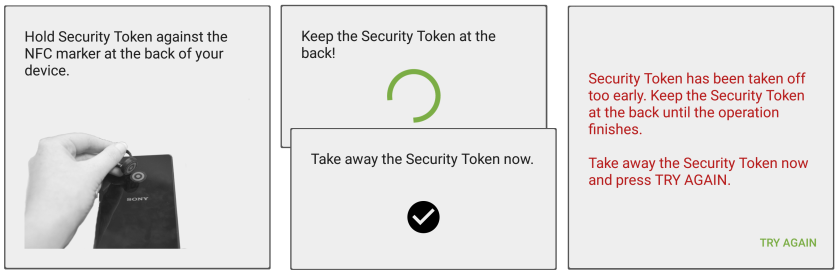
Action	Req. Extras	Description
SIGN_AND_ENCRYPT	USER_IDS	Encrypt to email addresses (USER_IDS) and generate signature
DECRYPT_VERIFY	-	Decrypt and verify signature

start the PendingIntent for user interaction. On SUCCESS, the encrypted and signed text has been streamed into the file descriptor given by *pipeId*.

For an API it is desirable to be stateless, i.e., the cryptography provider should be implementable without caching parameters or method calls for connected clients. Provider-side caching is unnecessary because the *result* Intent (previously passed through the PendingIntent), which is required for a second execution, is returned to the client after every user interaction (USER_INTERACTION_REQUIRED case). While this is possible for most parameters, PIN and passwords should never be exposed to the client and, thus, cannot be returned via the *result* Intent. Therefore, a PIN/password cache has been implemented using key IDs as unique identifiers. No session management is required inside the cryptography provider due to this architectural design. Due to their high abstraction, the exposed API methods work independently from the storage location of the secret key.

4.3 API

We provide a simple API specification in Table 1. A combined signature generation with encryption can be executed by creating an Intent with SIGN_AND_ENCRYPT with at least one extra holding the email addresses of the recipients named USER_IDS. The plaintext is streamed into the file descriptor and read from the file descriptor previously opened by the client. On first execution, the operation will result in the USER_INTERACTION_REQUIRED state three times before finishing with the ciphertext in SUCCESS. As shown in Figure 3a, the cryptography provider asks the end-user to allow the requesting client access to the API. Afterward, the user is asked to select her own key (secret key) by another UI component of the provider. If no key is available for the requested



Hold Security Token against the NFC marker at the back of your device.

Keep the Security Token at the back!

Take away the Security Token now.

Security Token has been taken off too early. Keep the Security Token at the back until the operation finishes.

Take away the Security Token now and press TRY AGAIN.

Fig. 4. Users are guided through the usage of security tokens over NFC.

email address a screen for key selection and retrieval is displayed, as shown in Figure 3b. Finally, based on the selection of the secret key, either a password (cf. Figure 3d) or a PIN is requested (similar to Figure 3d but with numeric keypad). This example shows that even without knowledge of public-key cryptography, a developer can effectively encrypt and sign data that can only be read by the recipient. Also, the secret key storage location is handled automatically and either a password input or NFC interaction with PIN input is returned. A second execution of the same Intent for different data will succeed earlier because access has now already been allowed, the public key is available and the PIN/password is already cached.

Regardless of whether the input is only encrypted, only signed, or a combined encryption with signature, an Intent with `DECRYPT_VERIFY` can be started to process the cryptographic input. No additional extras are required. Based on the PIN/password caching status of the specific secret key, the required screens are shown, e.g., Figure 3d. A special screen allows the user to restrict the secret keys that can be used by a particular client app (cf. Figure 3c). Besides the plaintext, two Parcelable objects are always returned indicating the decryption and signature result. These include the information if the given input was signed and/or encrypted.

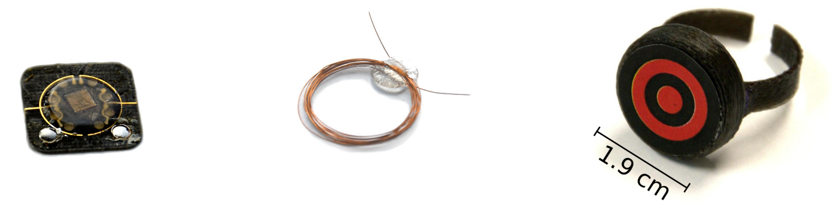
For security tokens, an appropriate PIN and Admin PIN must be chosen (cf. Figure 3e). Here, we prevent the user from chosing one of the top 20 common PIN combinations following Berry's PIN number analysis [9], e.g., 123456, 000000, and similar ones. An attacker can try up to 3 different PINs until the security token locks itself and can only be unlocked by the Admin PIN. We provided a trade-off between usability and security by letting the user select her favorite PIN, but securely generate an Admin PIN that should be written down. Our design decisions are similar to current practices of PIN/PUK selection for SIM cards.

Advanced API calls, for example to generate backups or detached signatures, can be found in OpenKeychain's API documentation [65].

4.4 NFC UI Component

We conducted a pre-study with 12 participants using a preliminary design of our NFC UI component. In this pre-study, we mainly focused on qualitative feedback, whereas the main goal of this pre-study was to find flaws in the UI design and user experience. We provided a Sony Xperia Z3 smartphone and a white NFC smart card that has been pre-configured for the scenario and asked the participants to send an encrypted email. We observed them during this task, especially their interaction with the NFC UI components. Finally, we interviewed them about their experience.

We found out that it is important to give clear instructions to guide the users through the steps of using a security token. In previous versions, users took away the security token too early or were confused when the dialog closed automatically after a successful operation. Thus, we improved the process by dividing it into three steps shown in Figure 4: 1) clearly depict how to hold the token against the device, 2) display a progress indicator together with the instruction to keep the token at the back, and finally 3) display the instruction that the token can



(a) IC extracted from NXP J3D081.

(b) Circular coil as new NFC antenna.

(c) 3D printed ring prototype.

Fig. 5. Components of our NFC signet ring prototype (photos have the same proportions).

be taken away. Our implementation keeps polling for the established NFC connection after successful operations to detect when the token is taken away. When the token indicates with an error that a wrong PIN has been used for authentication, the previous cached one for this key will be cleared for the next try.

4.5 NFC Smart Cards and Signet Rings

In addition to evaluating our architecture, in particular our proposed cryptographic API for developers, we want to study the impact of a different form factor for end-users. Widely available form factors are that of smart cards or USB sticks. Available NFC rings solely include read/write NFC tags (cf. Section 2) No challenge-response or asymmetric cryptographic protocols are supported. Thus, they do not satisfy the requirements in this paper to support full asymmetric cryptography.

Due to the unavailability of such rings, we created our own prototypes. Because smart card ICs are only sold to smart card manufacturers, we bought blank *NXP J3D081* developer cards with dual interface support for NFC. The IC, depicted in Figure 5a, has been extracted using acetone [74]). A new induction coil functioning as the NFC antenna has been created using magnet wire to fit the form factor (cf. Figure 5b). The number of turns for an NFC antenna operating at 13.56 MHz depends on the IC configuration. Thus, to estimate the correct number of turns, we measured the frequency of the original antenna with a signal generator and oscilloscope. The original antenna's frequency has been measured as 875 kHz, thus, the inductance can be calculated as $L = \frac{4.57}{875\,\text{kHz}} \approx 5.223\,\mu\text{H}$ [19]. According to NXP [63], the number of turns for circular coils can be calculated with

$$L = \frac{24.6 \cdot N^2 \cdot D}{1 + 2.75 \cdot \frac{s}{D}}\,.$$

The magnet wire has been wrapped around a metal cylinder with a diameter of $D = 1.53$ cm resulting in a circular coil with $s = 0.2$ cm. Choosing the number of turns with $N = 14$ results in an inductance close to the original one:

$$5.426\,\mu\text{H} \approx \frac{24.6 \cdot 14^2 \cdot 1.53\,\text{cm}}{1 + 2.75 \cdot \frac{0.2\,\text{cm}}{1.53\,\text{cm}}}$$

The resulting coil has been soldered with the IC and inserted into a 3D printed ring prototype as depicted in Figure 5c. It should be noted that the cylinder height on top of the ring can be reduced drastically by a more sophisticated production process.

5 EVALUATION

In this section we evaluate different aspects of our architecture by API comparison, NFC performance measurements, and a user study of our UI components.

5.1 Methodology

The evaluation of our architecture consists of several parts. (1) To understand its designated use for developers, a comparison with existing APIs, implemented as libraries and applications, has been done. (2) The raw performance of the cryptographic operations over NFC have been measured in a controlled environment. This helps to understand the technical constraints we needed to design UI components for. (3) As mentioned in Section 4.4, we conducted walkthroughs with 12 users from our university. The results indicated that users did not know when to keep NFC tokens at the back or when to take them away, which led to an improved UI for NFC operations. (4) Finally, we recruited 40 participants from a large company to test the full architecture from an end-user perspective including the UI components of our API and different NFC security tokens in a real-world environment. Furthermore, the usability and user perception of our NFC signet ring prototype as an alternative form factor has been evaluated.

5.2 API Comparison

Many cryptographic APIs are available that have been designed with different features and goals in mind. Thus, not every API suites every purpose and when designing communication systems the selection of an appropriate one largely depends on the following: If the design should be interoperable with existing standards, the API must support *standardized formats*. If it operates in a closed ecosystem, modern *high-level APIs* can be chosen where less programming errors can be made [7]. For higher security standards, especially in cooperate environments, the API should *support security tokens*. Furthermore, a category of APIs exist supporting functionality that go beyond cryptographic methods and require storage and GUI, such as *PIN/password cache*, *key management*, and *Graphical User Interfaces (GUI)*. These features provide complex functionality via API calls and reduce the burden on client developers who otherwise need to implement these on their own.

We selected prominent representatives for the categories of traditional low-level APIs, modern high-level APIs, and fully integrated systems and compared them in regards to the discussed functionality in Table 2. Only APIs for supporting encryption and signature generation for end-to-end security are considered, no authentication or transport security APIs are included. While more APIs exist, they typically fall into one of these categories and are thus evaluated similar to the selected ones. While modern libraries such as libsodium or Keyczar provide 'crypto-box'-methods with a fixed set of algorithms, GnuPG's selected algorithms depend on local configuration files and preferred algorithms defined in public key files. In our implementation, similar high-level operations with fixed algorithms exist that do not even require the knowledge of public-key cryptography due to additional UI components. While libraries such as Bouncy Castle must be integrated with additional libraries, such as OpenSC, to support security tokens, in our architecture security token support is an integral part. Modern libraries often lack a standard format and a corresponding key/algorithm migration path.

The features that require either support by the operating system or depend on specific GUI toolkits are typically not found in libraries, but in apps/integrated systems. An exception is Keyczar that provides basic command line tools for key management. One of the main goals of our system is to provide common user interactions via UI components. GnuPG specifies a 'UI Server Protocol' [40] that has similar goals and is implemented for Kleopatra and GPA. The library GpgME makes accessing this API easier [40]. In comparison to our implementation, the specification is not stateless and the implementation in Kleopatra does not provide dialogs for security tokens. Its Inter-Process Communication (IPC) is based on libassuan for platform-independent sockets. GNOME's Seahorse,

Table 2. Feature comparison of cryptographic APIs for end-to-end security. Libraries that only offer authentication or transport security are not considered here.

			High-Level API w/ Secure Defaults	Supports Security Tokens	Standardized Formats	Cross-Platform	PIN/Password Cache	Key Management	GUI
Low-Level APIs	libcrypto	[66]	○	○	●	●	○	○	○
	Bouncy Castle	[11]	○	○	●	●	○	○	○
	OpenSC	[37]	○	●	●	◐	○	○	○
High-Level APIs	NaCl/libsodium	[8]	●	○	○	●	○	○	○
	Keyczar	[21]	●	○	○	●	○	◐	○
Fully Integrated Systems	GnuPG	[40]	○	●	●	◐	●	●	○
	GNU Privacy Assistant (GPA)[a]	[41]	○	●	●	◐	●	●	●
	Kleopatra[a]	[50]	○	●	●	◐	●	●	●
	GNOME Keyring[a]	[38]	○	●	●	○	●	●	●
	Our work		●	●	●	○	●	●	●

[a] uses GnuPG as its backend

which is the frontend to GNOME Keyring, provides similar capabilities using a dbus service. Still, it misses functionality such as searching for and importing keys when choosing recipients [39].

While our API is not used as widely as the listed competitors, several client applications have already been released with an active user base. Its usage spans easy use cases, such as password managers (*Password Store*), as well as sophisticated ones, such as instant messaging (*Conversations*) and email clients (*K-9 Mail*) [65]. Most client apps are developed by third-parties and available on Google Play.

5.3 NFC Performance

We measured the performance of executing cryptographic operations over NFC using a Sony Xperia Z3 and the NXP J3D081 smart card running Yubico's OpenPGP app version 1.0.10. The average durations can be found in Table 3. Besides generating secret keys, which is only done once for new users, our measurements show average durations below 1 s for day-to-day operations. Only the asymmetric operations are executed on the smart card: For signatures, the hash of the input is generated on the smartphone, only the RSA signature is calculated on the smart card. For decryption, AES is executed on the smartphone, only the session key is decrypted on the smart card. 2048 bit RSA keys have been transferred and generated. ECC has not been evaluated, because no OpenPGP applet with ECC support was available for JCOP operating system during this work.

Table 3. Mean durations (w/ standard deviation) of cryptographic operations over NFC (10 experiments per operation).

Operation	Duration	σ
Signature calculation	787.9 ms	3.18
Decrypt session key	830.9 ms	55.86
Transfer existing secret key	711.9 ms	32.66
Generate secret key on-token[a]	9476.2 ms	2297.71

[a] Roughly, only every third key generation succeeded

Generating keys on the smart card turned out to be unreliable. Only roughly every third generation succeeded, while all other operations canceled by losing the connection. Even when having the card lying on a flat surface with the smartphone on top, we were never able to generate three keys in a row that makes this method unsuitable in practice. The same issues have been encountered with different smartphone-token combinations. By building a self-contained implementation, we ruled out issues in our architecture design. Instead, we suspect that the induction does not provide a perfectly stable energy supply, which is required by the key generation process. Because on-token key generation was too unreliable, in our current version keys are generated on-smartphone. We will investigate this issue further and will fix this in an upcoming version, possibly using ECC providing faster key generation methods.

5.4 User Study

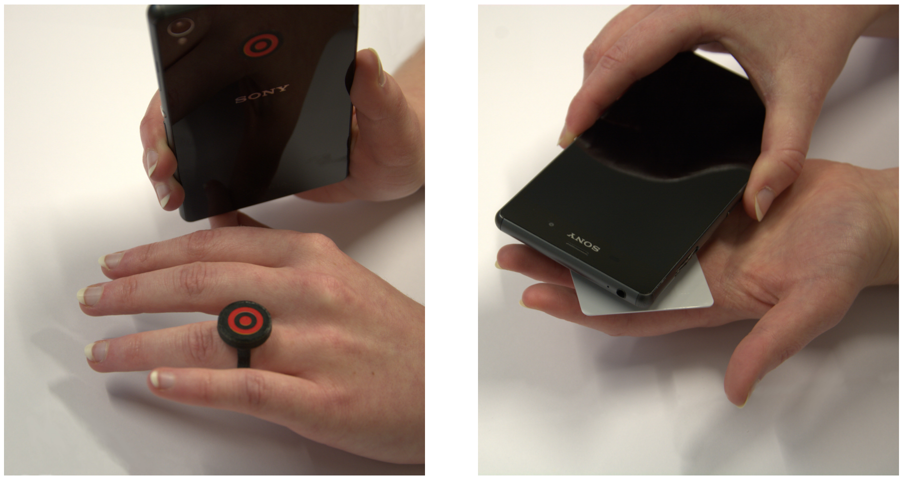
To evaluate the usability of NFC ring and card form factors in comparison to password-protected keys, we conducted an end-user study with 40 participants at a large company outside of the university environment. The main goal of our study is to test the usability of the NFC-based approaches in comparison with state-of-the-art password protection of secret key material. In this section, we present our study design and discuss the corresponding results.

5.4.1 Participant Recruitment. Our 40 participants were recruited in the IT department of a large company based in Germany. Taking part in the study was considered as working time, i.e., the participants were paid their normal hourly wages. Due to restrictions in their employment contract, we were not allowed to pay additional money on top. Our university does not have an Institutional Review Board (IRB), but the study conformed to the strict data protection law of Germany and informed consent was gathered from all participants.

5.4.2 Study Design. Our conducted study consists of two parts: (1) a lab experiment observing objective measurements such as *setup time*, *decryption time* and (2) a follow-up user survey for analyzing end-user perception.

The experiment consists of different tasks to be performed with different approaches.

5.4.3 Variables, Conditions and Participant Assignment. Our independent variable among all tasks was the chosen authentication type with following conditions *password*, *NFC card*, and *NFC ring*. The in the evaluation relevant dependent variables (objective measurements) were (1) duration to measure the efficiency of each task and (2) user perception based on a follow-up survey. The effectiveness was not considered as a separate dependent variable in our evaluation since all users were able to perform the tasks. For the condition assignment, we opted for a within-group design where all study participants had to perform tasks from all approaches: *password*, *NFC card*, and *NFC ring*. We did not test against other methods commonly used for authentication, such as biometric fingerprints or pattern-based techniques. Generally, these provide a much lower security level than passwords satisfying modern length requirements, let alone security tokens [56] and are thus not suitable for end-to-end encryption. Our design allows us to gather user perception at the end of the study where users give feedback and



(a) Ring with smartphone. (b) Card with smartphone.

Fig. 6. Handling of NFC rings and cards together with smartphones. A sticker has been used to indicate the best NFC spot on smartphone and ring.

ratings for all approaches. To mitigate learning and fatigue effects in our within-group study design, the order in which participants were asked to perform the approaches was randomized.

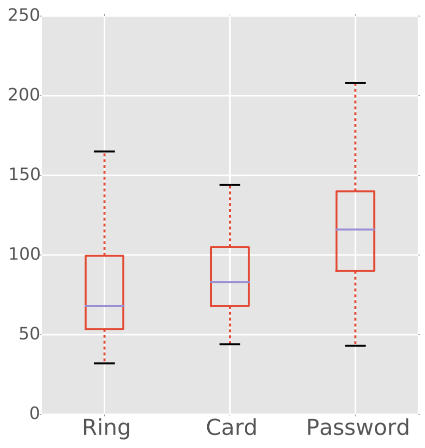
5.4.4 Tasks. The performed tasks are:

Task 1 When users start the application for the first time, they have to follow a wizard to create a key pair for usage with the app. They are guided through the process, which consists of entering a name and an email address, select a password/PIN and depending on the approach to hold an NFC security token against the smartphone. The actual key generation is indicated by a progress bar, while the user has to wait until it finishes.

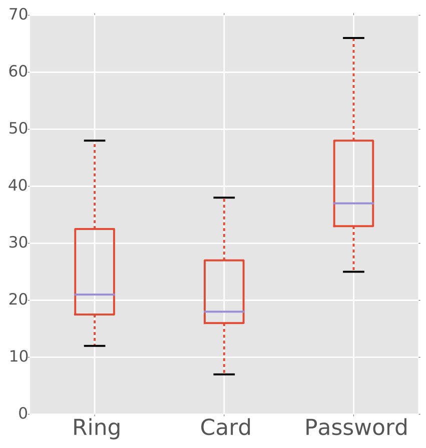
Task 2 In the second task, the participants are asked to receive and read an encrypted email. Depending on the approach, users might be asked to enter a PIN or password, or hold an NFC device against their smartphone. To avoid bias due to variable password/PIN complexities, during this step we provide a pre-defined password/PIN.

Task 3 At last, the participants are asked to reply with a secure email by writing an appropriate response text and sending it.

To begin with the study, we provided a detailed explanation of the concept and the procedure to participants. We gave them a Sony Xperia Z3 smartphone and optionally depending on the approach, either the NFC ring or NFC card to let them get accustomed to the hardware themselves. As depicted in Figure 6, the usage patterns between the NFC ring and card differ due to their physical size. Before conducting the study, a sticker has been attached to the smartphone and ring indicating the best spot and the affiliation between these objects. Right after this, the participants continued with the key creation wizard of the first tested approach followed by the other remaining tasks. After completing the first tested approach, the other approaches follow. At the end of the study, we interviewed the participants for their ratings with regards to the single approaches and additional feedback. Finally, they were asked to participate in an anonymized questionnaire to collect demographic statistics.



(a) Setup time.



(b) Decryption time.

Fig. 7. Time measurements (in seconds, no outliers, lower is better).

5.4.5 Statistical Testing. For the statistical hypothesis testing, we opted for the common significance level of $\alpha = 0.05$. To account for multiple testing, all our study p-values are reported in the Holm-Bonferroni corrected version [44]. All time intervals and user-ratings are tested with the *nonparametric* (applicable to unknown statistical distributions) *Mann–Whitney U* test (two-tailed, Holm-Bonferroni corrected). We opted for this nonparametric statistical test due to the ordinal nature of our data and to avoid any statistical distribution assumptions. All our effect sizes are reported by mean comparisons and the usage of the *common language effect size* method [57], i.e., the meaning of the effect size is explained in plain English.

5.4.6 Objective Measurements. We measured following objective measurements in our experiment:

Setup time We measure the entire time of the setup process in Task 1. This includes input of name and email address, password/PIN selection, key generation on-smartphone, and optionally transfer to security token.

Decryption time Here, we measure only the time where the users have to perform an action related to the cryptographic operation of Task 2, i.e., password input and on-smartphone asymmetric decryption or PIN input and on-token operation (requiring holding the token against the smartphone's back side).

Sign/Encryption time Again, we measure the time where the users have to perform an action in Task 3. Due to PIN/password caching, no input is required for signing. Thus, we only measured the time required for executing the NFC operations by holding the security tokens against the smartphone's back side.

Figure 7a shows a box plot with a time distribution overview for the setup process. Our main hypothesis is that passwords are less efficient (especially on smartphones) in comparison to NFC-based approaches which is also a common belief in the usable security community. As can be clearly seen, the password-based approach tends to require extra time: a median of 114.5 seconds indicated by the blue line in box ($p < 0.0001$ in comparison to the NFC-based approaches supporting our main hypothesis). NFC-based approaches, on the other hand, have shown a better performance during the wizard process: a median of 83.5 and 68.5 seconds ($p = 0.083$ indicating that based on our sample size, we could not observe a significant difference between those). For instance, only 14 people were able to type in a valid password on the first try. By comparison, 22 people were able to position the ring correctly and choose a valid PIN on the first try.

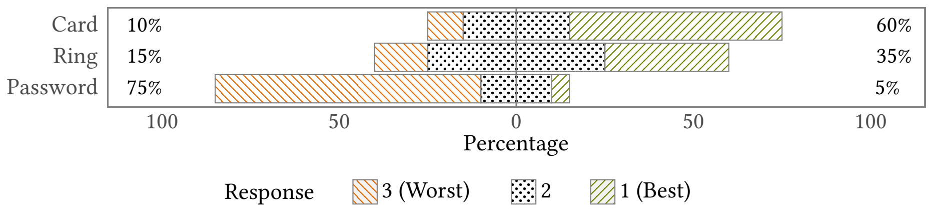


Fig. 8. Aggregated user perception showing the ranking choices in the interview.

Figure 7b shows a box plot with a time distribution overview for decrypting an email. This process consists of the following steps: when a secured email is opened, the cryptographic API immediately starts with the decryption process. To be able to decrypt, depending on the approach, either a password input or the positioning of the NFC device in combination with a PIN input is required. Note, that during this task we provided the participants a pre-defined password/PIN to avoid bias due to variable complexities. As can be clearly seen, the password-based protection not only performs worst in security and setup time, but also requires additional time for reading encrypted emails: median of 37.0 seconds (18.5 and 22.0 for NFC-based solutions) as indicated by the blue lines in the boxes ($p < 0.001$ supporting our main hypothesis). In the follow-up writing of a secured email where the password is cached, NFC-based approaches require additional time for positioning of the NFC devices: mean of 9.6 seconds (11 seconds for the ring and 8.3 for the card).

5.4.7 User Perception. To measure user perception, after completing all tasks in the experiment, we asked the users to answer questions with regards to their tested approaches in an interview. Here, we distinguish between *quantitative* results aggregated from users' ratings and *qualitative* open-end questions asking the users for their feedback and justifications. Our hypothesis is based on a common belief that passwords are troublesome on mobile devices. The full interview can be seen in the Appendix A. Furthermore, we asked them to fill out a short questionnaire form with additional details about their demographics, education, computer literacy, and previous usage of security apps. The full questionnaire can be seen in the Appendix B.

Quantitative. As our major quantitative question in the interview, we asked our participants to rank their tested approaches: 1 as best and 3 as worst. As depicted in Figure 8, the majority of our participants ranked NFC-based approaches to be superior to the password-based approach. The color coding shows the ranking level and the percentage numbers on the y-axes summarize the percentage of participants who ranked an approach as best (1) and as worst (3) respectively. A pairwise comparison between the approaches shows a high statistical significance between the token-based approaches and password-based protection ($p < 0.0001$ in both cases). However, we could not observe significant results between the NFC card and the ring ($p = 0.073$). The NFC card approach achieved a slightly higher mean of 1.5 in comparison the NFC ring that achieved a mean of 1.8. By comparison, password-based key storage achieved a mean rating of 2.7 where only 5 % of the users consider this approach to be best.

Qualitative. During the interview we asked the participants to describe the advantages and disadvantages of each approach (cf. Appendix A). While the participants accepted passwords as a well-known approach, most participants agreed that "imagining good passwords is incredibly difficult" and "good passwords are difficult to enter on smartphone keyboards". Whereas, cards and rings have the advantage of "requiring only a short PIN instead of a complicated password". In general, participants were in favor of cards, due to their common form factor, which allows to "store them easily in the wallet". Some mentioned that it is annoying to constantly take it out of the wallet, thus they prefer wearing the card attached to their belt. Many participants remarked that the card was more easily be placed below the smartphone and then worked perfectly with NFC and did not require precise positioning like the ring. Participants who favored the ring found the idea great and described it as a "cool gadget". Some noted that "rings are more secure than cards because they are more difficult to steal than wallets" and their "security purpose is not immediately obvious to an outsider". Interestingly, participants assessed it differently if cards or rings are more easily lost. Some argued that "rings can easily be forgotten on a bedside cabinet while not worn at night", while they argued that they "never forget the wallet in the morning before work". Others said that "cards are easily misplaced as they are not constantly worn on the body".

Demographics. A total of 40 users from the same company participated in our user study. 33 participants were male, 6 participants were female and one participant opted not to disclose the gender. The mean and median age of the participants was 34. In the quantitative analysis, we could not find any statistically significant differences between the genders, although women tend to prefer wearable NFC devices over cards. During the interview we also noted different reasons for liking/disliking particular approaches. For instance, 9 out of 33 men preferred cards instead of rings simply because they usually do not wear rings at all and are not accustomed to it. Some of them proposed the usage of watches or wristbands as an alternative form factor. A woman argued that, because dresses are often worn without belts, she "prefer[s] to wear cards attached to a necklace". Naturally, she and two other woman preferred the ring as it can be worn as a fashion accessory and has a smaller size.

14 participants did not have a university degree (3 of them were students planning to complete a degree). 8 participants completed a Bachelor's degree or similar, 17 a Master's degree or similar and 1 a doctorate's degree. We could not observe statistical significant differences between the degrees.

Our question set (cf. Appendix A) indicates a high level of technical background. On a scale from 1 (novice user) to 20 (experienced) the participants achieved a mean score of 17.1 wheres the participant with least knowledge achieved a score of 13.

Limitations. First-off, our study does not test whether end users will actually switch to NFC-based encryption or even start encrypting their emails during daily work. As with any lab study, more issues might arise in the field and thus an actual field study is an important future work. Our 40 participants were recruited in the IT department in a large company based in Germany, which is not the representative of the general population in Germany. As mentioned before, our questionnaire (cf. Appendix A) indicates a high level of technical background.

6 CONCLUSION

We proposed and implemented an architecture for NFC-based cryptography on Android devices. Our architecture includes a high-level cryptographic API especially designed for developers accustomed to Android's IPC mechanisms. It allows for cryptographic operations without knowledge of public-key cryptography, works transparently with password-protected key files as well as NFC security tokens, and provides carefully designed user interactions. In addition to traditional NFC smart cards, our NFC signet ring prototype represents an alternative form factor for end-users. Performance measurements show that cryptographic operations over NFC can be executed fast enough to be usable for day-to-day use. In our lab study with 40 participants, 95 % chose one of the NFC solutions as the best approach. Conclusively, we have shown the advantages of our architecture for NFC-based cryptography.

A INTERVIEW

The original questions were asked in German.

- Which operating system are you using on your own smartphone?
- For every approach {password, smart card, signet ring} (in the order the approaches have been tested by the participant):
  - What did you think was good?

  - What did you think was bad?
- Which approach was the best in your opinion? Please create an order by assigning the numbers 1, 2, 3.

Approach	Order
Password	[ ]
Smart Card	[ ]
Signet Ring	[ ]

- Would you consider replacing passwords by a signet ring?
- Would you consider replacing smart cards by a signet ring?

B QUESTIONNAIRE

The original questionnaire was written in German. Furthermore, it included in addition gender, age, and qualification questions (Question 1–3), which are not displayed in this appendix.

Question 4

Please rate how much you agree (or disagree) with the statements below.

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
I have a very good understanding of computers and the Internet.	☐	☐	☐	☐	☐
I often ask others for help when I have computer problems.	☐	☐	☐	☐	☐
Others often ask me for help when they have computer problems.	☐	☐	☐	☐	☐
I have a very good understanding of computer security.	☐	☐	☐	☐	☐

Question 5

Are you already using apps for encryption or secure communication on your *computer* in your *private* life?

☐ Yes, in particular: ....
☐ No

Question 6

Are you already using apps for encryption or secure communication on your *smartphone* in your *private* life?

☐ Yes, in particular: ....
☐ No

Question 7

Are you already using apps for encryption or secure communication on your *smartphone* in your *job*?

☐ Yes, in particular: ....
☐ No

Question 8

Have you already used NFC before this study?

- ☐ Yes, in particular for: ....
- ☐ No

ACKNOWLEDGMENTS

The authors would like to thank Vincent Breitmoser, Joey Castillo, and Signe Rüsch for contributing to the NFC-related code, Johannes van Balen and Yannic Schröder for their help creating the signet ring prototype, as well as Linda Fliss and Stephan Merker for their assistance and coordination during the lab study. Furthermore, we would like to thank all participants of the study.

REFERENCES

[1] ‘Alex288’. 2014. NFC Smart Card Reader PC/SC Library: Project Description. (2014). Retrieved July 2017 from https://nfcsmartcardreader.codeplex.com

[2] Android Documentation. 2017. Cipher class. (2017). Retrieved July 2017 from http://developer.android.com/reference/javax/crypto/Cipher.html

[3] Android Documentation. 2017. Near Field Communication. (2017). Retrieved July 2017 from http://developer.android.com/guide/topics/connectivity/nfc/index.html

[4] Android Open Source Project. 2017. Nexus Security Bulletins. (2017). Retrieved July 2017 from https://source.android.com/security/bulletin

[5] Apple Inc. 2017. About Cryptographic Services. (2017). Retrieved July 2017 from https://developer.apple.com/library/ios/documentation/Security/Conceptual/cryptoservices/Introduction/Introduction.html

[6] Apple Inc. 2017. PassKit Package Format Reference. (2017). Retrieved July 2017 from https://developer.apple.com/library/ios/documentation/UserExperience/Reference/PassKit_Bundle/Chapters/TopLevel.html

[7] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. 2012. *Progress in Cryptology – LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter The Security Impact of a New Cryptographic Library, 159–176. https://doi.org/10.1007/978-3-642-33481-8_9

[8] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. 2012. *The Security Impact of a New Cryptographic Library*. Springer Berlin Heidelberg, Berlin, Heidelberg, 159–176. https://doi.org/10.1007/978-3-642-33481-8_9

[9] Nick Berry. 2012. PIN analysis. (Sept. 2012). Retrieved July 2017 from http://datagenetics.com/blog/september32012/index.html

[10] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy*. 553–567. https://doi.org/10.1109/SP.2012.44

[11] Bouncy Castle Inc. 2017. The Legion of the Bouncy Castle. (2017). Retrieved July 2017 from http://www.bouncycastle.org

[12] T. W. C. Brown, T. Diakos, and J. A. Briffa. 2013. Evaluating the eavesdropping range of varying magnetic field strengths in NFC standards. In *7th European Conference on Antennas and Propagation (EuCAP)*. 3525–3528.

[13] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. 2007. OpenPGP Message Format. RFC 4880 (Proposed Standard). (Nov. 2007).

[14] ‘Cane’, ‘Topo’, and ‘Orso’. 2017. Privacy-Handbuch: GnuPG-SmartCard und NitroKey. (2017). Retrieved July 2017 from https://www.privacy-handbuch.de/handbuch_32r.htm

[15] Qi Alfred Chen, Zhiyun Qian, and Z. Morley Mao. 2014. Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks. In *23rd USENIX Security Symposium (USENIX Security)*. USENIX Association, San Diego, CA, 1037–1052.

[16] Yongsoon Choi, Jordan Tewell, Yukihiro Morisawa, Gilang A. Pradana, and Adrian David Cheok. 2014. Ring*U: A Wearable System for Intimate Communication Using Tactile Lighting Expressions. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology (ACE ’14)*. ACM, Article 63, 4 pages. https://doi.org/10.1145/2663806.2663814

[17] Brett Cooley, Haining Wang, and Angelos Stavrou. 2014. *Activity Spoofing and Its Defense in Android Smartphones*. Springer International Publishing, Cham, 494–512. https://doi.org/10.1007/978-3-319-07536-5_29

[18] Stephen M. Curry. 1998. An introduction to the Java Ring. (April 1998). Retrieved July 2017 from http://www.javaworld.com/article/2076641/learn-java/an-introduction-to-the-java-ring.html

[19] Ronald Dekker. 2017. A Simple Method to Measure Unknown Inductors. (2017). Retrieved July 2017 from http://www.dos4ever.com/inductor/inductor.html

[20] Frank Denis. 2017. The Sodium crypto library (libsodium). (2017). Retrieved July 2017 from https://libsodium.org

[21] Arkajit Dey and Stephen Weis. 2008. Keyczar: A Cryptographic Toolkit. (Aug. 2008). Retrieved July 2017 from http://keyczar.googlecode.com/files/keyczar05b.pdf

[22] W. Diao, X. Liu, Z. Li, and K. Zhang. 2016. No Pardon for the Interruption: New Inference Attacks on Android Through Interrupt Timing Analysis. In *IEEE Symposium on Security and Privacy (SP)*. 414–432. https://doi.org/10.1109/SP.2016.32

[23] ECMA International. 2015. NFC-SEC-01: NFC-SEC Cryptography Standard using ECDH and AES, 4rd edition. ECMA-386. (June 2015). http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-386.pdf

[24] ECMA International. 2015. NFC-SEC: NFCIP-1 Security Services and Protocol, 4rd edition. ECMA-385. (June 2015). http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-385.pdf

[25] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*. ACM, 73–84. https://doi.org/10.1145/2508859.2516693

[26] Rebecca Ehlers, Thorsten Ehlers, Werner Koch, and Matthias Kirschner. 2006. The GnuPG Smartcard HOWTO. (June 2006). Retrieved July 2017 from https://www.gnupg.org/howtos/card-howto/en/smartcard-howto.html

[27] Nikolay Elenkov. 2014. *Android Security Internals: An In-Depth Guide to Android's Security Architecture.* No Starch Press.

[28] M. Elkins, D. Del Torto, R. Levien, and T. Roessler. 2001. MIME Security with OpenPGP. RFC 3156 (Proposed Standard). (Aug. 2001).

[29] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 50–61.

[30] Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith, and Uwe Sander. 2012. Helping Johnny 2.0 to Encrypt His Facebook Conversations. In *Proceedings of the Eighth Symposium on Usable Privacy and Security (SOUPS '12)*. ACM, Article 11, 17 pages. https://doi.org/10.1145/2335356.2335371

[31] Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. 2013. Rethinking SSL Development in an Appified World. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*. ACM, 49–60. https://doi.org/10.1145/2508859.2516655

[32] Fidesmo. 2017. Card App Store. (2017). Retrieved July 2017 from http://www.fidesmo.com

[33] 'Fluffy'. 2017. OpenPGP-Card. (2017). Retrieved July 2017 from https://github.com/FluffyKaon/OpenPGP-Card

[34] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. 2010. *Practical NFC Peer-to-Peer Relay Attack Using Mobile Phones.* Springer Berlin Heidelberg, Berlin, Heidelberg, 35–49. https://doi.org/10.1007/978-3-642-16822-2_4

[35] Simson L. Garfinkel, David Margrave, Jeffrey I. Schiller, Erik Nordlander, and Robert C. Miller. 2005. How to Make Secure Email Easier to Use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. ACM, 701–710. https://doi.org/10.1145/1054972.1055069

[36] Simson L. Garfinkel and Robert C. Miller. 2005. Johnny 2: A User Test of Key Continuity Management with S/MIME and Outlook Express. In *Proceedings of the 2005 Symposium on Usable Privacy and Security (SOUPS '05)*. ACM, 13–24. https://doi.org/10.1145/1073001.1073003

[37] GitHub. 2017. OpenSC/OpenSC. (2017). Retrieved July 2017 from https://github.com/OpenSC/OpenSC

[38] GNOME. 2014. Keyring. (2014). Retrieved July 2017 from https://wiki.gnome.org/action/show/Projects/GnomeKeyring

[39] GNOME. 2014. Seahorse Roadmap. (2014). Retrieved July 2017 from https://wiki.gnome.org/Apps/Seahorse/Roadmap

[40] GnuPG authors. 2017. Appendix A The GnuPG UI Server Protocol. (2017). Retrieved July 2017 from https://www.gnupg.org/documentation/manuals/gpgme/UI-Server-Protocol.html

[41] GnuPG authors. 2017. GPA - The Gnu Privacy Assistant. (2017). Retrieved July 2017 from https://www.gnupg.org/software/gpa/index.html

[42] Ernst Haselsteiner and Klemens Breitfuß. 2006. Security in Near Field Communication (NFC). In *Printed Handout of Workshop on RFID Security (RFIDSec)*. Philips Semiconductors.

[43] Mario Heiderich, Jann Horn, Abraham Aranguren, Jonas Magazinius, and Dario Weißer. 2015. Pentest-Report OpenKeychain. (Aug. 2015). https://cure53.de/pentest-report_openkeychain.pdf.

[44] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[45] Michael Hölzl, Endalkachew Asnake, René Mayrhofer, and Michael Roland. 2014. Mobile Application to Java Card Applet Communication using a Password-authenticated Secure Channel. In *12th International Conference on Advances in Mobile Computing and Multimedia (MoMM)*. ACM Press, New York, NY, USA, 147–156. https://doi.org/10.1145/2684103.2684128

[46] Identiv. 2015. uTrust 2910 R Data Sheet. (Feb. 2015). http://www.identiv.com/pdf/technicaldata/technical-datasheets/uTrust_2910R_Reader_DS_2015_02.pdf

[47] ISO/IEC. 2008. *ISO/IEC 14443-4: Identification cards – Contactless integrated circuit cards – Proximity cards – Part 4: Transmission protocol.*

[48] ISO/IEC. 2013. *ISO/IEC 7816-4: Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange.*

[49] A. K. Jain, A. Ross, and S. Pankanti. 2006. Biometrics: a tool for information security. *IEEE Transactions on Information Forensics and Security* 1, 2 (June 2006), 125–143. https://doi.org/10.1109/TIFS.2006.873653

[50] KDE. 2017. Kleopatra - Certificate Manager and Unified Crypto GUI. (2017). Retrieved July 2017 from https://www.kde.org/applications/utilities/kleopatra/

[51] Henning Kortvedt and S Mjolsnes. 2009. Eavesdropping near field communication. In *The Norwegian Information Security Conference (NISK)*, Vol. 27.
[52] Juan Lang, Alexei Czeskis, Dirk Balfanz, Marius Schilder, and Sampath Srinivas. 2017. *Security Keys: Practical Cryptographic Second Factors for the Modern Web*. Springer Berlin Heidelberg, Berlin, Heidelberg, 422–440. https://doi.org/10.1007/978-3-662-54970-4_25
[53] Frederic Lardinois. 2015. Google And Samsung Will Now Release Monthly OTA Android Security Updates. (Aug. 2015). http://techcrunch.com/2015/08/05/google-and-samsung-will-now-release-monthly-ota-android-security-updates
[54] Shrirang Mare, Mary Baker, and Jeremy Gummeson. 2016. A Study of Authentication in Daily Life. In *Twelfth Symposium on Usable Privacy and Security (SOUPS)*. USENIX Association, Denver, CO, 189–206.
[55] Mario Heiderich and Krzysztof Kotowicz. 2013. Pentest-Report Mailvelope 12.2012 - 02.2013. (2013). Retrieved July 2017 from https://cure53.de/pentest-report_mailvelope.pdf
[56] Mindi McDowell, Jason Rafail, and Shawn Hernan. 2009. Cyber Security Tip ST04-002. US-CERT. (2009). Retrieved July 2017 from http://www.us-cert.gov/cas/tips/ST04-002.html
[57] Kenneth O McGraw and SP Wong. 1992. A common language effect size statistic. *Psychological bulletin* 111, 2 (1992), 361.
[58] John McLear. 2013. NFC Ring - One Smart Ring, Unlimited Possibilities. (July 2013). Retrieved July 2017 from https://www.kickstarter.com/projects/mclear/nfc-ring
[59] Maryam Mehrnezhad, Mohammed Aamir Ali, Feng Hao, and Aad van Moorsel. 2016. *NFC Payment Spy: A Privacy Attack on Contactless Payments*. Springer International Publishing, Cham, 92–111. https://doi.org/10.1007/978-3-319-49100-4_4
[60] MOTA. 2017. MOTA DOI SmartRing. (2017). Retrieved July 2017 from http://shop.mota.com/mota-doi-smartring.html
[61] Nitrokey. 2017. OpenPGP support. (2017). Retrieved July 2017 from https://github.com/Nitrokey
[62] Nitrokey. 2017. Secure your digital life. (2017). Retrieved July 2017 from https://www.nitrokey.com
[63] NXP Semiconductors. 2010. AN1445: Antenna design guide for MFRC52x, PN51x and PN53x. (Oct. 2010). http://data.nxp.com/doc/published_files/1270733179751
[64] OpenIntents. 2017. Where applications unite. (2017). Retrieved July 2017 from http://www.openintents.org
[65] OpenKeychain. 2017. Easy PGP. (2017). Retrieved July 2017 from https://www.openkeychain.org
[66] OpenSSL. 2016. Libcrypto API. (2016). Retrieved July 2017 from https://wiki.openssl.org/index.php/Libcrypto_API
[67] Celeste Lyn Paul, Emile Morse, Aiping Zhang, Yee-Yin Choong, and Mary Theofanos. 2011. A field study of user behavior and perceptions in smartcard authentication. In *Human-Computer Interaction–INTERACT 2011*. Springer, 1–17.
[68] A. Pietig. 2009. Functional Specification of the OpenPGP application on ISO Smart Card Operating Systems. (April 2009). http://www.g10code.com/docs/openpgp-card-3.0.pdf
[69] Precise Biometrics. 2017. Smart Card Readers for Convenient and Secure Access. (2017). Retrieved July 2017 from http://precisebiometrics.com/smart-card-reader
[70] Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. 2015. Towards Discovering and Understanding Task Hijacking in Android. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 945–959.
[71] Arne Renkema-Padmos, Jerome Baum, Melanie Volkamer, and Karen Renaud. 2014. Shake Hands to Bedevil: Securing Email with Wearable Technology.. In *Proceedings of the Eighth International Symposium on. Human Aspects of Information Security & Assurance (HAISA 2014)*. 90–100.
[72] Research In Motion Limited. 2007. Smart Card Security Solved: The BlackBerry Smart Card Reader. (2007). Retrieved July 2017 from http://www.blackberry.com/newsletters/connection/it/i5-2007/smart-card-reader.shtml
[73] RINGLY. 2017. Smart Jewelry and Accessories. (2017). Retrieved July 2017 from https://ringly.com
[74] Michael Roland and Michael Hölzl. 2015. Evaluation of Contactless Smartcard Antennas. (July 2015). http://arxiv.org/abs/1507.06427
[75] Martina Angela Sasse. 2005. Usability and trust in information systems. In *Trust and Crime in Information Societies*, R Mansell and B Collins (Eds.). Edward Elgar, Cheltenham, UK, 319–348.
[76] Florian Schmaus, Dominik Schürmann, and Vincent Breitmoser. 2016. *XEP-0373: OpenPGP for XMPP*. Technical Report. XMPP Standards Foundation, http://xmpp.org/extensions/xep-0373.html.
[77] Florian Schmaus, Dominik Schürmann, and Vincent Breitmoser. 2016. *XEP-0374: OpenPGP for XMPP Instant Messaging*. Technical Report. XMPP Standards Foundation, http://xmpp.org/extensions/xep-0374.html.
[78] Dominik Schürmann and Lars Wolf. 2016. Surreptitious Sharing on Android. In *Sicherheit 2016 (Lecture Notes in Informatics)*, Vol. P-256. Gesellschaft für Informatik, Bonn, Germany, 137–148. http://www.ibr.cs.tu-bs.de/papers/schuermann-sicherheit2016.pdf
[79] Dennis D Strouble, GM Schechtman, and Alan S Alsop. 2009. Productivity and usability effects of using a two-factor security system. *Proceedings of SAIS* (2009), 196–201.
[80] Michael Tunstall. 2006. Attacks on Smart Cards. (2006). http://www.cs.bris.ac.uk/home/tunstall/presentation/AttacksonSmartCards.pdf
[81] Alma Whitten and J. Doug Tygar. 1999. 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)*. USENIX Association.
[82] Meng Xu, Chengyu Song, Yang Ji, Ming-Wei Shih, Kangjie Lu, Cong Zheng, Ruian Duan, Yeongjin Jang, Byoungyoung Lee, Chenxiong Qian, Sangho Lee, and Taesoo Kim. 2016. Toward Engineering a Secure Android Ecosystem: A Survey of Existing Techniques. *ACM*

Comput. Surv. 49, 2, Article 38 (Aug. 2016), 47 pages. https://doi.org/10.1145/2963145
[83] Yubico. 2017. Trust the Net with YubiKey Strong Two-Factor Authentication. (2017). Retrieved July 2017 from https://www.yubico.com
[84] Yubico. 2017. YubiKey NEO's OpenPGP app. (maintained fork of "Java Card OpenPGP Card"). (2017). Retrieved July 2017 from https://github.com/Yubico/ykneo-openpgp
[85] Zimperium. 2015. Experts Found a Unicorn in the Heart of Android. (July 2015). Retrieved July 2017 from https://blog.zimperium.com/experts-found-a-unicorn-in-the-heart-of-android

Received May 2017; accepted July 2017.