information

MDPI

Article

# Protecting Touch: Authenticated App-To-Server Channels for Mobile Devices Using NFC Tags

**Fernando Kaway Carvalho Ota [1,*], Michael Roland [2,*], Michael Hölzl [3,*], René Mayrhofer [3] and Aleardo Manacero [4]**

1 Banco do Brasil S.A., 70790-125 Brasília, Brazil
2 University of Applied Sciences Upper Austria, 4232 Hagenberg, Austria
3 Institute of Networks and Security, Johannes Kepler University Linz, 4040 Linz, Austria; mayrhofer@ins.jku.at
4 Department of Computer Science and Statistics, São Paulo State University—UNESP, 15054-000 São José do Rio Preto, Brazil; aleardo@ibilce.unesp.br
* Correspondence: fernando.kaway@bb.com.br (F.K.C.O.); michael.roland@fh-hagenberg.at (M.R.); hoelzl@ins.jku.at (M.H.)

Received: 26 May 2017; Accepted: 28 June 2017; Published: 6 July 2017

**Abstract:** Traditional authentication methods (e.g., password, PIN) often do not scale well to the context of mobile devices in terms of security and usability. However, the adoption of Near Field Communication (NFC) on a broad range of smartphones enables the use of NFC-enabled tokens as an additional authentication factor. This additional factor can help to improve the security, as well as usability of mobile apps. In this paper, we evaluate the use of different types of existing NFC tags as tokens for establishing authenticated secure sessions between smartphone apps and web services. Based on this evaluation, we present two concepts for a user-friendly secure authentication mechanism for mobile apps, the Protecting Touch (PT) architectures. These two architectures are designed to be implemented with either end of the spectrum of inexpensive and widely-available NFC tags while maintaining a reasonable trade-off between security, availability and cost.

**Keywords:** secure channel; two-factor authentication; Near Field Communication (NFC); Android; mobile security

## 1. Introduction

Currently, one of the most wide-spread security mechanisms to authenticate users against online services are username and password combinations. While passwords are seemingly easy to implement from a developer's perspective, they put a significant amount of responsibility on the user. In order to make passwords secure, users would have to choose and remember long and complex combinations of letters or words, digits and other symbols (cf. Yan et al. [1]). Particularly on mobile devices, the limited input capabilities (e.g., small on-screen keyboards) often prevent users from choosing such a secure password (see the study by Greene et al. [2]). Moreover, eavesdropping (e.g., through shoulder surfing or keyloggers), phishing, taking notes of passwords and even intentional sharing can pose a risk to systems protected by this security mechanism.

In order to reduce the risk associated with passwords (or user knowledge in general), critical online services often add a secondary authentication factor (e.g., something that the user has) to the authentication process. For instance, online banking systems often use one-time passwords transmitted to the user's mobile phone via SMS messages or an app. In that scenario, the mobile device is considered something personal, that users would always carry close to themselves. Consequently, the mobile device together with the SMS channel are considered a second factor held by the user.

This second factor is well separated from the main path to the online service when that service is, for example, accessed through a web browser on a desktop PC. However, with the mass adoption of smartphones, more and more online services that were originally designed to be accessed from desktop PCs are now used through the same mobile devices that were previously considered as second factor (cf. Schartner and Bürger [3]). Consequently, the malware on a smartphone could both eavesdrop on keystrokes to record passwords, as well as intercept received SMS messages to get hold of one-time passwords transmitted over the secondary channel (see Konoth et al. [4]).

This suggests that different methods are needed to implement two-factor authentication for services targeting mobile devices. The adoption of Near Field Communication (NFC) on a broad range of smartphones opens up NFC-enabled tokens as a secondary authentication factor usable on mobile devices. An NFC tag could be used as the storage for the secret key material that is then used for authentication and to secure communication between mobile apps and their online backend services.

NFC-enabled tokens are already widely used for physical access control systems. Similar concepts were brought to the smartphone world. For instance, the FIDO Alliance recently published a specification for using a special NFC-enabled token as a second authentication factor based on public-key cryptography [5]. Furthermore, recent Android versions support using NFC tags as identifiers to unlock the lock screen. However, these methods typically either require new generations of NFC tags or rely on easily duplicable information (e.g., anti-collision identifiers of NFC tags). Therefore, they are not usable with NFC tags and contactless smart cards that are already widely in use by and available to smartphone users (e.g., membership cards, employee ID badges, general-purpose NFC tags available in online stores).

In order to remain with the previously-mentioned example of online banking, we focus our considerations on an exemplary mobile banking app. This app, targeting smartphones and tablets, gives users access to their bank account. Users may check their account balance, review transactions and transfer money to other bank accounts. To perform each of these operations, the mobile banking app connects to a backend server of the bank through a secure channel. Both endpoints of the secure channel, i.e., the backend server on one side and the specific app installation and its user on the other side, need to be authenticated. Using, for instance, TLS (Transport Layer Security) and certificate pinning, the app can easily authenticate the backend server. Our approach focuses on authentication of the mobile device side. For one, we use key material stored on an NFC tag to verify that the user possesses a specific token. Second, we combine that with key material stored within the app (or a system-provided secured key vault accessible by the app) to bind the NFC tag and the access to the secure channel to one specific installation of the app (after an initial password-based login).

In this paper, we assess the capabilities of different types of existing NFC tag technologies and evaluate how these tags can be used as tokens for establishing authenticated secure sessions between smartphone apps and web services. Two concepts for a user-friendly secure authentication mechanism for mobile apps based on NFC tags, the Protecting Touch (PT) architectures, are outlined. These two architectures cater to both ends of the spectrum of available, low-cost NFC tag products providing a reasonable trade-off between security, availability and cost. Without loss of generality, we focus on the requirements of our exemplary mobile banking app, but these concepts could easily be applied to other app-to-backend communication use-cases with similar security requirements, as well.

## 2. Related Work

The most recent approach to two-factor authentication using NFC on mobile devices was designed by the FIDO Alliance. Their Universal 2nd Factor (U2F) series of specifications [6] standardizes authentication tokens based on public-key cryptography that can, among others, be used over NFC [5]. U2F specifically targets two-factor authentication for web services with cryptographically-strong asymmetric key pairs. The U2F NFC transport [5] permits these key pairs to be securely stored on dedicated physical tokens (e.g., a contactless smart card chip, such as YubiKey NEO [7]).

Similarly to the U2F approach, several national identity cards (such as the Spanish DNIe [8] or the German nPA [9]) are usable for authentication over their contactless NFC interface and, consequently, could be used in combination with NFC-enabled smartphones. However, the implementations of these authentication protocols typically vary between different countries.

With regard to NFC tags, WISeKey offers their VaultIC product line as a solution against counterfeiting, cloning and identity theft [10]. Based on elliptic curve public-key cryptography, these NFC tags can be authenticated using a challenge-response protocol. However, WISeKey seems to be restrictive on providing information on how that system works, and the tags are not easily available.

Urien [11] describes an extension to the TLS protocol using the Chip Authentication Program (CAP) protocol of EMV payment cards such as credit and debit cards to establish a shared key for TLS-PSK between the banking server and the client web browser. This can reduce the threat of Man-In-The-Middle (MITM) attacks using forged or untrusted TLS certificates. According to [11], this protocol can only be implemented if the server backend has access to an issuer authentication server, a service provided by the card issuer, that can verify EMV-CAP cryptograms. Consequently, it is questionable if this technology would be open to other areas except banking websites.

The most prominent authentication use-case with NFC tags in the context of the Android platform is Smart Lock. Introduced in Android 5.0, Smart Lock unlocks the lock screen when in proximity of another "trusted" device. One of the technologies that can be used for Smart Lock is NFC. According to Elenkov [12], the unlock mechanism usually relies on the freely readable serial number (e.g., the anti-collision identifier) of an NFC tag. Consequently, it relies on simple identifiers that could easily be cloned onto specially-crafted tags or emulated by special card emulators (cf. Duc et al. [13]). No cryptographic mechanisms are used to validate the authenticity of tags.

The goal of this paper is to assess the possibilities and design new protocols for leveraging existing, widely-used, low-cost NFC tags and contactless smart cards for the purpose of establishing an authenticated channel between a mobile app and a web service. Existing approaches either rely on special and not (yet) widely-used NFC tokens, or they do not use any cryptography at all. Moreover, current approaches mainly focus on using NFC tags/cards as tokens that are authenticated through challenge-response protocols between two endpoints. As opposed to this, our Protecting Touch architecture aims at using NFC tags as a storage for key material used for establishing an encrypted and authenticated communication channel between the mobile app and its backend and, consequently, as an integral part of that secure channel.

## 3. Functionality of NFC Tags

The NFC Forum, a non-profit industry association driving the development of the whole NFC ecosystem, maintains five specifications for NFC tag types: the Tag Operation specifications [14–18]. These standardize the memory structures and interface commands for access to different types of NFC tag hardware. Memory is either organized in a flat linearly-addressable structure or in a file-oriented structure. The NFC Data Exchange Format (NDEF) [19] maps the hardware-specific memory structures to one common data format and defines an abstraction layer on top of any tag hardware. The memory allocatable to NDEF data on currently available tag products ranges from a few bytes up to a few kilobytes (though the theoretical limit according to the specifications is approximately 1 MB).

According to the specifications, NFC tags are designed to be freely readable by any device in close proximity. Therefore, NFC tags may only be used as freely readable data storage based on NDEF in order to achieve a maximum level of interoperability. No authentication or protection mechanisms against reading of tag contents are standardized. The only protection that is part of the tag operation specifications is a mechanism to restrict writing or modification of tag contents. Consequently, contents of NFC tags can easily be copied from one tag to another. Moreover, an attacker eavesdropping on the communication between the app and the tag could also obtain sufficient information to duplicate the tag since the communication is not encrypted.

Most NFC tags have identifiers (4–10 bytes) that are used for anti-collision and enumeration of NFC tags in the range of a reader device. Manufacturers usually assign them in a way that each tag gets a unique identifier. The identifiers are assigned during production and typically stored in read-only memory. Since these identifiers cannot be modified on regular tag hardware, it is possible to use them to uniquely identify tags or to cryptographically bind the NDEF data to one specific tag. However, special emulators and special tags with modifiable identifiers are known to exist (cf. Duc et al. [13]). Consequently, these identifiers cannot reliably protect against tag cloning.

Some existing tag products implement additional security features that go beyond permanently locking tag contents to read-only by providing different forms of authentication mechanisms (e.g., MIFARE DESFire). These product-specific security features make use of proprietary commands that are not part of the NFC Forum Tag Operation specifications. As a result, these extensions can only be used on platforms that permit the exchange of the necessary low-level commands (e.g., Android).

The most simple mechanism is a password that is transmitted to the tag in clear-text and confirmed or rejected by the tag. Data on the tag may only become readable after authentication with the password. This further protects against cloning, unless an attacker has the capability to eavesdrop on the communication with the tag or to extract the password from the app. Moreover, if the password verification yields a binary "yes"/"no" result and the data are freely readable, special tag or emulator hardware could simply acknowledge any password in order to bypass actual verification (cf. Murdoch et al. [20], who revealed a similar issue with PIN verification on payment cards).

A more sophisticated approach is symmetric mutual challenge-response authentication using a shared key. Some tag products also use the challenge-response authentication to establish an ephemeral session key that is then used to authenticate and optionally encrypt the communication between the reader and the tag. Mutual challenge-response authentication prevents eavesdropping on the shared secret. Such authentication can be performed directly between a tag and a backend server using the app as a transparent proxy. Thus, the app on the mobile device does not need to store the shared secret and is, consequently, not prone to extraction of the secret key from its memory.

Besides simple NFC tags, there are also other tag and contactless smart card products specifically focusing on cryptographic functionality. For instance, application-specific and programmable processor smart cards (e.g., Java Card) can be used to implement authentication applications (e.g., electronic ID cards, bank cards, solutions like YubiKey). However, these types of tags and smart cards are either not widely available, bound to specific existing applications and infrastructure or not available as low-cost products. Therefore, these products are beyond the scope of this paper, since our focus is on widely available, low-cost, general-purpose NFC tag products.

## 4. Protecting Touch

Analyzing the functionality of existing NFC tag products revealed that there is a broad spectrum of tag capabilities ranging from pure, freely-readable memory to tags that allow mutually authenticated and encrypted communication channels. We designed two concepts for a user-friendly secure authentication mechanism for mobile apps based on NFC tags: the Protecting Touch (PT) architectures. Each of the two PT architectures was designed for one of the two ends of the spectrum of available low-cost NFC tag products:

1. The first architecture (PT1) uses freely-readable NFC tags as storage for key material that is bound to a specific installation of the authenticating mobile app and can be used for authentication towards the backend system. This architecture is designed to work with any NFC tag, given that the tag has sufficient memory space for the key material. PT1 relies solely on the NDEF abstraction layer to provide a maximum level of interoperability.
2. The second architecture (PT2) uses the mutual three-pass authentication protocol of MIFARE DESFire tags and specific side-effects of the DESFire communication protocol to generate deterministic random symmetric session keys between the mobile app and its backend. This architecture specifically targets MIFARE DESFire and, thus, the upper end (in terms of

capabilities and cost) of available NFC tags. While PT2 requires more expensive tags, the price is still far below smart card solutions such as the YubiKey NEO (more than a factor of 10; see Section 4.5).

### *4.1. Requirements and Threats*

As described in the Introduction, our protocols are designed to fit the requirements of an exemplary mobile banking app, but should also be applicable to other app-to-backend communication use-cases with similar security requirements.

Currently, banking apps primarily use passwords and one-time passwords (transmitted out-of-band) as a means of user authentication. After an initial password-based authentication, apps store a session token that can be used to authenticate the backend in future communication (e.g., until explicit user-triggered logout). Thus, while this session token is valid, users do not need to re-type the password when using the app. Only certain operations (e.g., money transfers) require an additional authentication step where users explicitly need to re-type their password and/or provide a one-time password. There are several problems with this approach:

1. Anyone with access to the app may be able to use the session token to access some functionality of the app and some data about the bank account without knowing the password. Even if the user interface of the app is protected by a lock pattern or PIN that is verified locally in the app, an attacker may be able to bypass or discover that code by analyzing the app and its memory (e.g., using malware running on the mobile device).
2. Going one step further, it is questionable whether session tokens are sufficiently secured within the app. Typically, it should not be possible to duplicate the app (including its session token) onto another device (e.g., through a backup and restore) without voiding the authenticated session.
3. Users who often perform operations that require re-typing the password may tend to use passwords that can be easily typed using the limited input capabilities of the mobile device (cf. Greene et al. [2]). Additionally, these users are more frequently exposed to the possibility of eavesdropping on the password through shoulder surfing, etc.
4. The mobile device can no longer be considered a secure out-of-band channel for the delivery of one-time passwords (e.g., through SMS or through an app) when the mobile banking app is accessed from the same device (cf. Schartner and Bürger [3] and Konoth et al. [4]).

Our solution based on NFC tags addresses all of these issues:

1. The app requires users to present their NFC tag whenever they use the app. Authenticating the NFC tag from the backend side prevents attacks possible with locally-verified lock mechanisms.
2. Secured key vaults accessible only by the app, for instance the Android Keystore, are a means to facilitate storage for key material used by the app. Key vaults are dedicated hardware (or software) components that encapsulate the key material in a way that it never leaves the vault. In order to use the keys, apps delegate their cryptographic operations to the key vault.
   Binding key material stored on the tag to key material stored within the app (or a key vault) reduces the attack surface against extraction of that key material from the key vault since the bound key material is useless unless the attacker also has access to the NFC tag (or potentially a copy of it). Moreover, it binds the tag to one specific app installation.
3. Requiring users to carry an NFC tag besides the smartphone is potentially less cumbersome than memorizing and frequently typing a complex password. Since the NFC tag is required for authentication, this may also reduce the risk of the password getting known to any third party since they would also need to get hold of the NFC tag. Nevertheless, new risks may be introduced through the use of NFC tags as well. For instance, the NFC interface may be easily eavesdropped on. Further, an NFC tag is a physical object that needs to be carried around by its user and may be subject to loss or theft.

4. The NFC tag can be used besides a normal password as a secondary authentication factor. Moreover, it acts as an additional factor besides session tokens stored within the app. However, we acknowledge that in the latter case, both factors, the NFC tag and the smartphone app installation instance, follow the same principle, i.e., they are both something that the user has.

### *4.2. Protecting Touch 1*

With Protecting Touch 1, any freely-readable NFC tag can be used as an additional authentication factor. The tag is used to store a shared key valid for authenticating one session between the app and its backend server and refreshed during each session. In addition, key material stored within a key vault on the mobile device is used to bind the tag contents to one specific installation instance of the app. This key material can either be symmetric keys for encryption ($k_{App,enc}$) and message authentication ($k_{App,mac}$) or, if the key vault does not support storage of symmetric keys (like the Android Key Store before Android 6.0), an asymmetric key pair ($K_{App}$) used to encrypt temporary symmetric keys that can be stored in potentially insecure memory. See Table 1 for the definition of symbols used in the protocol description and throughout this paper.

**Table 1.** Symbols used in cryptographic protocols.

| Symbol | Definition |
|---|---|
| $\mathrm{E}_k\,(m)$ | Encryption of $m$ using key $k$ |
| $\mathrm{D}_k\,(c)$ | Decryption of $c$ using key $k$ |
| $\mathrm{HMAC}\,(k, m)$ | Message authentication code over $m$ using key $k$ |
| $\mathrm{DH}_{A,B}\,(k)$ | Authenticated key-agreement between $A$ and $B$ using shared key $k$ based on the principles of Diffie-Hellman key exchange |
| $\mathrm{PAKE}_{A,B}\,(p)$ | Password-authenticated key-agreement between $A$ and $B$ using password $p$ |
| $\mathrm{ROL}_n\,(x)$ | Rotation of $x$ to the left by $n$ bits |
| $\mathrm{SCR}\,(x)$ | $x$ scrambled following a defined scheme |
| $a \,\|\|\, b$ | Concatenation of $a$ and $b$ |
| $r_x$ | Random value |
| $k_x$ | Shared key |
| $K_{x,Pub}$ | Public key of asymmetric key pair $K_x$ |
| $K_{x,Priv}$ | Private key of asymmetric key pair $K_x$ |

#### 4.2.1. Tag Memory Layout

The data for PT1 are packed into an NDEF message that can be stored on any NFC tag. The payload $OTK_{Payload}$ of a One-Time Key (OTK) record consists of a user identifier $ID_{user}$, a key index counter $CT_{OTK}$ and a one-time key $k_{OTK}$. Additionally, the record payload is encrypted and contains a message authentication code. Both encryption and message authentication are done using keys known to the app and stored inside (or protected by) a key vault. A random initialization vector for cipher block chaining is maintained within the app memory.

$$OTK_{plain} = ID_{user} \,||\, CT_{OTK} \,||\, k_{OTK} \tag{1}$$

$$OTK_{enc} = \mathrm{E}_{k_{App,enc}}\left(OTK_{plain}\right) \tag{2}$$

$$OTK_{mac} = \mathrm{HMAC}\left(k_{App,mac}, OTK_{enc}\right) \tag{3}$$

$$OTK_{Payload} = OTK_{enc} \,||\, OTK_{mac} \tag{4}$$

Assuming four bytes each for $ID_{user}$ and $CT_{OTK}$, a length of 32 bytes for $k_{OTK}$, AES-128 (with CBC) for encryption and HMAC-SHA-256 as the message authentication code (32 bytes), the payload of an OTK record would consist of 80 bytes. Including the overhead introduced by the NDEF header and the record type name "usmile.at:ptotk" (arbitrarily chosen), an OTK record can fit into 100 bytes. With HMAC-SHA-512 (and AES-256), the record fits into 130 bytes.

In addition to the OTK record, the NFC tag may contain an NDEF record that triggers automatic launching of the app upon tapping the tag. For instance, on Android, this can be achieved with an Android Application Record (AAR).

#### 4.2.2. Protocol Operation

The authentication protocol is started upon tapping the NFC tag with the mobile device. This may either happen after the app instructed the user to tap the tag or when the user taps the tag to launch the app. The authentication protocol consists of the following steps (see Figure 1):

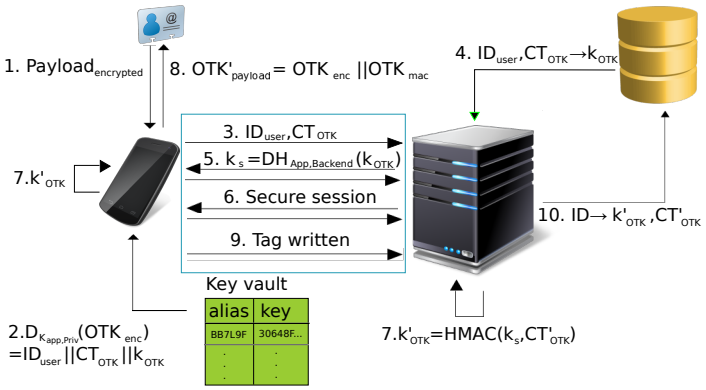


**Figure 1.** Protecting Touch 1 (PT1) protocol operation.

1. The app reads the NDEF message from the NFC tag and extracts the OTK record.

$$\text{App} \leftarrow \text{Tag}: \quad OTK_{Payload} \tag{5}$$

2. The app splits the payload into $OTK_{enc}$ and $OTK_{mac}$, verifies the message authentication code, decrypts the OTK record payload, and extracts the data elements ($ID_{user}$, $CT_{OTK}$, $k_{OTK}$) using keys $k_{App,mac}$ and $k_{App,enc}$ stored in the key vault.

$$OTK_{Payload} \rightarrow OTK_{enc} \,||\, OTK_{mac} \tag{6}$$

$$OTK_{mac} \stackrel{?}{=} \text{HMAC}\left(k_{App,mac}, OTK_{enc}\right) \tag{7}$$

$$OTK_{plain} = \text{D}_{k_{App,enc}}\left(OTK_{enc}\right) \tag{8}$$

$$OTK_{plain} \rightarrow ID_{user} \,||\, CT_{OTK} \,||\, k_{OTK} \tag{9}$$

If the key vault does not support symmetric keys directly, an asymmetric key pair stored within the key vault may be used instead. In that case, the symmetric keys $k_{App,mac}$ and $k_{App,enc}$ are stored in "insecure" device memory in encrypted form $e_{App,...}$. The keys are only decrypted using the private key $K_{App,Priv}$ of the key pair inside the key vault when needed.

$$k_{App,...} = \text{from key vault} \tag{10}$$

$$\stackrel{or}{=} \text{D}_{K_{App,Priv}}\left(e_{App,...} \text{ from app data}\right) \tag{11}$$

3. The app initializes the interaction with the backend server by sending $ID_{user}$ and $CT_{OTK}$ in order to give the server a hint about the authenticating user and the used tag.

$$\text{App} \rightarrow \text{Backend}: \quad ID_{user}, \; CT_{OTK} \tag{12}$$

4. The backend loads $k_{OTK}$ from its key database (based on $ID_{user}$ and key index counter $CT_{OTK}$) and starts the authentication phase. If no matching one-time key is in the database, communication with the app is ended.
5. The app and backend perform an authenticated key-agreement using $k_{OTK}$ as the shared authentication key (see Shin et al. [21] or Saha and Roy Chowdhury [22] for possible implementations).

$$\text{App} \leftrightarrow \text{Backend}: \quad k_s = \text{DH}_{App,Backend}\left(k_{OTK}\right) \tag{13}$$

6. After agreeing on a session key $k_s$, the app and backend continue their communication over a secure channel (authenticated and encrypted based on the agreed session key). The implementation of the secure channel protocol is beyond the scope of this paper.
7. The app and the backend increment the key index counter and derive a new $k'_{OTK}$ from $k_s$. The key derivation is based on the HMAC-based One-Time Password (HOTP) algorithm [23].

$$CT'_{OTK} = CT_{OTK} + 1 \tag{14}$$

$$k'_{OTK} = \text{HMAC}\left(k_s, CT'_{OTK}\right) \tag{15}$$

8. The app constructs the new OTK record (from $ID_{user}$, the incremented key index counter and the new one-time key), encrypts it and adds a message authentication code using the symmetric keys inside the key vault.

$$OTK'_{plain} = ID_{user} \,||\, CT'_{OTK} \,||\, k'_{OTK} \tag{16}$$

$$OTK'_{enc} = \text{E}_{k_{App,enc}}\left(OTK'_{plain}\right) \tag{17}$$

$$OTK'_{mac} = \text{HMAC}\left(k_{App,mac}, OTK'_{enc}\right) \tag{18}$$

$$OTK'_{Payload} = OTK'_{enc} \,||\, OTK'_{mac} \tag{19}$$

If the key vault does not support symmetric keys directly, fresh symmetric keys $k_{App,mac}$ and $k_{App,enc}$ are generated and used for the above operations. They are then encrypted to $e_{App,mac\,|\,enc}$ using the asymmetric key pair in the key vault and stored in regular device memory.

$$k_{App,\ldots} = \text{random value} \tag{20}$$

$$e_{App,\ldots} = \text{E}_{K_{App,Pub}}\left(k_{App,\ldots}\right) \tag{21}$$

Then, the OTK record is stored onto the tag.

$$\text{App} \rightarrow \text{Tag}: \quad OTK'_{Payload} \tag{22}$$

9. Upon successfully writing the data to the tag, the app sends a confirmation to the backend. The user is instructed to reattach the tag in case of write errors.
10. The backend, in turn, stores the new one-time key and the incremented key index counter in its key database and removes the old one-time key. If the app does not confirm a successful write operation, the server discards the new one-time key and terminates the session. After too many failed attempts, it may block the current one-time key and require a fresh tag enrollment.

#### 4.2.3. Initial Tag Enrollment

During initial tag enrollment, the user identifier and a first one-time key are stored on an NFC tag. Moreover, the app generates a new key pair and stores it in its key vault. The initialization protocol

is started by the user by typing their user ID ($ID_{user}$) and enrollment password ($p_{enroll}$) into the app. The protocol consists of the following steps (see Figure 2):

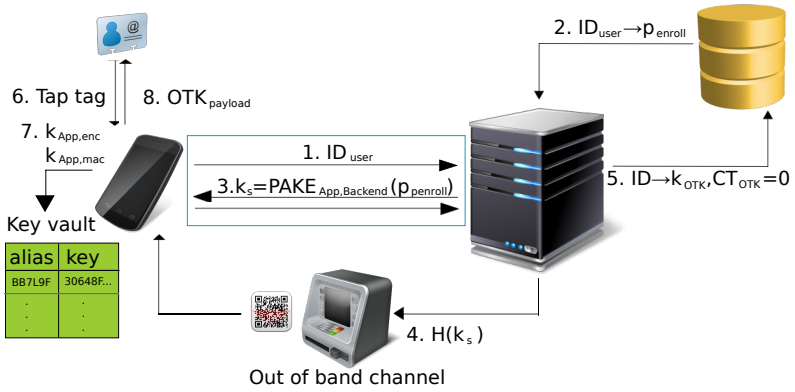


**Figure 2.** Initial tag enrollment in PT1.

1. The app initializes the interaction with the backend server by sending $ID_{user}$ and indicating enrollment of a new tag.

$$\text{App} \rightarrow \text{Backend}: \; ID_{user} \tag{23}$$

2. The backend loads the enrollment password from its user database (based on $ID_{user}$) and starts the authentication phase. If no matching $ID_{user}$ is found, the communication with the app is terminated.
3. The app and backend perform a password-authenticated key-agreement (e.g., SRP [24]) using $p_{enroll}$ as the password.

$$\text{App} \leftrightarrow \text{Backend}: \; k_s = \text{PAKE}_{App,Backend}\left(p_{enroll}\right) \tag{24}$$

4. We acknowledge that the enrollment password may not be considered sufficiently strong for all application scenarios. For these cases, the authentication phase may be enhanced through an out-of-band channel. Both sides generate a hash over the shared secret $k_s$. The backend then transmits this hash to the app over the out-of-band channel. The app compares the hashes in order to verify it was communicating directly with the backend server.
   Such an out-of-band channel could, for instance, be an Automatic Teller Machine (ATM) that displays the hash in the form of a QR code scannable by the app and requests the user to provide the verification result after the user logged in using their bank card and PIN.
5. The app and the backend set the key index counter to zero and derive $k_{OTK}$ from the session key. The backend stores the one-time key and the initial key index counter in its key database.

$$CT_{OTK} = 0 \tag{25}$$

$$k_{OTK} = \text{HMAC}\left(k_s, CT_{OTK}\right) \tag{26}$$

6. The user taps an NFC tag (after being instructed to do so by the app).
7. The app generates new symmetric keys $k_{App,mac}$ and $k_{App,enc}$ in its key vault.
   If the key vault does not support symmetric keys directly, a new asymmetric key pair is generated instead. In that case, random temporary keys $k_{App,mac}$ and $k_{App,enc}$ are generated, encrypted using the public key of the key pair in the key vault, and the encrypted version is stored in regular device memory.

8. The app constructs the new OTK record (from $ID_{user}$, the initial key index counter and the one-time key), encrypts it and adds a message authentication code using the symmetric keys from the key vault or the temporary symmetric keys (see (1)–(4) in Section 4.2.1).

$$\text{App} \rightarrow \text{Tag}: \quad OTK_{Payload} \tag{27}$$

### *4.3. Protecting Touch 2*

Protecting Touch 2 targets the other end of the spectrum of available NFC tag products. It is designed specifically for MIFARE DESFire tags. While we had specifically DESFire EV1 in mind when designing the protocol, it also works with other generations of DESFire after minor modifications.

PT2 uses the three-pass authentication protocol to mutually authenticate the tag and the backend server and to establish a shared secret (a session key) between the tag and the backend. The protocol (ab)uses a specific implementation detail of the DESFire communication protocol: the fact that only the response of the read command (that is used to read a file) is encrypted and authenticated. The app uses this fact to obtain the data contents of a file encrypted by the session key. Since the command header packet of the read command is neither encrypted nor authenticated (even though the file itself is readable only after successful mutual authentication), the app can get hold of the encrypted version of the file data without knowing the authentication key or the session key shared between the tag and the backend. The encrypted file data are then used as a deterministic random (cf. $r_A$, $r_B$, $k_{s,tag}$ and $k_s$ in (30)–(35) in Section 4.3.2) symmetric session key between the app and the backend. The plain-text version of the file data and the tag authentication key are considered a long-term shared secret between the tag and the backend.

#### 4.3.1. Tag Memory Layout

The tag contains a DESFire application consisting of two files. The first file contains the user ID ($ID_{user}$) and is configured to be freely readable. The second file contains the long-term shared secret ($k_{LT}$) and is configured to be readable only in encrypted mode. In addition, the DESFire application contains one key for mutual authentication using the AES algorithm ($k_{tag}$). Authentication and encryption is required to read the second file.

In addition, the DESFire tag may be configured as the NFC Forum Type 4 tag containing an NDEF record for automatically launching the app upon tapping (e.g., an AAR).

The authentication protocol is started upon tapping the NFC tag with the mobile device. This may either happen after the app instructed the user to tap the tag or when the user taps the tag to launch the app. The authentication protocol consists of the following steps (see Figure 3):

1. The app reads $ID_{user}$ and $ID_{tag}$ (anti-collision identifier) from the tag.

$$\text{App} \leftarrow \text{Tag}: \quad ID_{user},\ ID_{tag} \tag{28}$$

2. The app and the backend mutually authenticate using a previously-established session token (which could rely on an application-accessible key vault). While the exact mechanism used for the session token is not relevant for our protocol, we implemented the token as an asymmetric key pair stored in the Android Key Store. The app uses the private key to authenticate sessions, and the backend stores the public key of this key pair during enrollment for verifying the session authentication.
3. The app initializes the authentication phase with the backend server by sending $ID_{user}$ and $ID_{tag}$ in order to give the server a hint about which user and tag are used to authenticate. Note that $ID_{user}$ needs to also match the session.

$$\text{App} \rightarrow \text{Backend}: \quad ID_{user},\ ID_{tag} \tag{29}$$

4. The backend loads $k_{tag}$ and $k_{LT}$ from its key database (based on $ID_{user}$ and $ID_{tag}$) and starts the tag authentication phase. If no matching key is in the database, communication with the app is ended.
5. The backend starts the three-pass mutual AES authentication with the tag by sending appropriate DESFire commands to the app. The app operates as a transparent proxy between the backend and the tag in order to exchange the commands needed for the mutual three-pass authentication.

$$\text{Backend} \rightarrow \text{Tag}: \quad \text{Start authentication} \tag{30}$$

$$\text{Backend} \leftarrow \text{Tag}: \quad \text{E}_{k_{tag}}\left(r_B\right) \tag{31}$$

$$\text{Backend} \rightarrow \text{Tag}: \quad \text{E}_{k_{tag}}\left(r_A \,||\, \text{ROL}_8\left(r_B\right)\right) \tag{32}$$

$$\text{Backend} \leftarrow \text{Tag}: \quad \text{E}_{k_{tag}}\left(\text{ROL}_8\left(r_A\right)\right) \tag{33}$$

First, the tag generates a random $r_B$, encrypts it using the shared $k_{tag}$ and returns the cryptogram to the backend. The backend decrypts the received value, generates a random $r_A$, concatenates that value with a rotated version of the received $r_B$ and sends that concatenated value encrypted with the shared $k_{tag}$. The tag, in turn, decrypts the received value and verifies $r_B$ to authenticate the backend. Upon successful verification, the tag returns a rotated and encrypted version of $r_A$. Finally, the backend decrypts the received value and verifies $r_A$ to authenticate the tag.

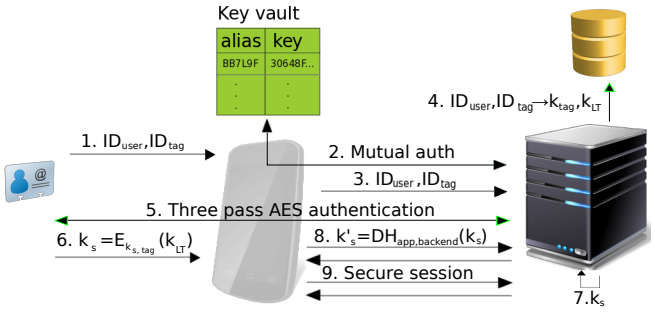


**Figure 3.** PT2 protocol operation.

#### 4.3.2. Protocol Operation

At the end of the mutual authentication, the tag and backend share a secret ephemeral session key $k_{s,tag}$ based on the random values $r_A$ and $r_B$ (scrambled as defined by the DESFire protocol):

$$\text{Backend} \leftrightarrow \text{Tag}: \quad k_{s,tag} = \text{SCR}\left(r_A \,||\, r_B\right) \tag{34}$$

6. The app reads the file containing the long-term shared secret ($k_{LT}$) and gets its encrypted form $k_s$ (encrypted with the session key $k_{s,tag}$ shared between the tag and the backend).

$$\text{App} \leftarrow \text{Tag}: \quad k_s = \text{E}_{k_{s,tag}}\left(k_{LT}\right) \tag{35}$$

7. The backend also calculates $k_s$ from $k_{s,tag}$ and $k_{LT}$. At this point, the app and the backend both know the shared session key $k_s$.
8. The app and backend perform an authenticated key-agreement using $k_s$ as the shared key in order to prevent passive attackers spying on the NFC link from using an eavesdropped $k_s$ to decrypt any of the further communication.

$$\text{App} \leftrightarrow \text{Backend}: \quad k'_s = \text{DH}_{App,Backend}\left(k_s\right) \tag{36}$$

9. After agreeing on a session key $k'_s$, the app and backend continue their communication over a secure channel (authenticated and encrypted based on that agreed session key). The implementation of the secure channel protocol is beyond the scope of this paper.

#### 4.3.3. Initial Tag Enrollment

The initial tag enrollment creates the PT2 DESFire application and, if necessary, the NDEF tag application (cf. [17]) on a DESFire tag. Moreover, the procedure creates and writes the two files containing the user ID ($ID_{user}$) and the long-time shared secret ($k_{LT}$) and configures the permissions and the application-specific authentication key $k_{tag}$.

As a pre-requisite, we require the tag to have a pre-configured shared master key $k_{m,tag}$ with permission to create new DESFire applications. In addition, users must have received their enrollment credentials ($ID_{user}$ and enrollment password $p_{enroll}$) over some out-of-band channel (e.g., via mail). Alternative approaches are discussed in Section 4.3.4.

Enrollment is started by the user by typing their user ID ($ID_{user}$) and enrollment password ($p_{enroll}$) into the app. The protocol consists of the following steps (see Figure 4):

1. The app initializes the interaction with the backend server by sending $ID_{user}$ and indicating enrollment of a new tag.

$$\text{App} \rightarrow \text{Backend}: \quad ID_{user} \tag{37}$$

2. The backend loads the enrollment password from its user database (based on $ID_{user}$) and starts the authentication phase. If no matching $ID_{user}$ is found, communication with the app is terminated.
3. The app and backend perform a password-authenticated key-agreement using $p_{enroll}$ as the password.

$$\text{App} \leftrightarrow \text{Backend}: \; k_s = \text{PAKE}_{App,Backend}\left(p_{enroll}\right) \tag{38}$$

4. The same mechanism as in PT1 may be used to perform additional verification over an out-of-band channel (see Section 4.2.3, Step 4).
5. If no out-of-band verification is to be performed, the app and backend continue their communication on a secure channel using the session key $k_s$ (proceeding with Step 6). Otherwise, after out-of-band verification, they restart their communication by resending $ID_{user}$ and performing an authenticated key-agreement using $k_s$ as the shared key. The newly established session key $k'_s$ is then used for the secure channel. This additional step reduces the risks resulting from key leakage during the (potentially long) period of time it takes to complete out-of-band verification.

$$\text{App} \rightarrow \text{Backend}: \quad ID_{user} \tag{39}$$

$$\text{App} \leftrightarrow \text{Backend}: \; k'_s = \text{DH}_{App,Backend}\left(k_s\right) \tag{40}$$

6. The backend and the app agree on a session token used to authenticate the app installation instance in future communication sessions. While the exact mechanism used for the session token is not relevant for our protocol, we implemented the token as an asymmetric key pair stored in the Android Key Store with the public key stored by the backend (see Step 2 in Section 4.3.2).
7. The user taps the DESFire tag (after being instructed to do so by the app).
8. The app passes $ID_{tag}$ to the backend in order to allow the backend to lookup the pre-shared tag master key $k_{m,tag}$ in its key database.
9. The app then operates as a transparent proxy between the backend and the tag in order to exchange all DESFire commands necessary to configure the tag and its memory contents. This includes initial mutual three-pass authentication using the pre-shared tag master key $k_{m,tag}$ and

the configuration of the DESFire applications including the application-specific key $k_{tag}$, the file structure, the file data for the file containing $ID_{user}$, the file data for the file containing $k_{LT}$ and all file access permissions.

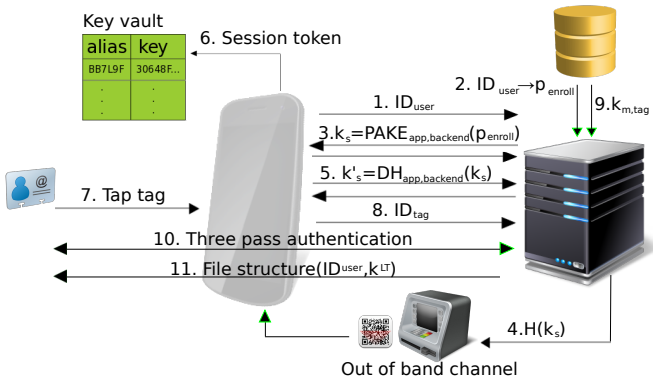


**Figure 4.** Initial tag enrollment in PT2.

#### 4.3.4. Alternative Enrollment Strategies

For our current enrollment scenario, we assume that the tag is pre-configured with a shared master key $k_{m,tag}$. This can, for instance, be the case when the DESFire tags are already distributed to the users for another application use-case such as a public transport card, an access control token or a membership card. In such cases, cooperation with the operator of the existing system is required in order to gain access to the master key or to delegate the initial tag configuration (Step 9 in Section 4.3.3) to that operator. Tags could even be completely pre-enrolled before distribution and be delivered to their users in a ready-to-use state. This would eliminate the initial tag enrollment from within the app.

Another alternative would be to allow enrollment of blank DESFire cards (that have their master key set to its factory-default value). This would give users the opportunity to choose their own tags. However, this also has an impact on security: a malicious user or an active MITM on the NFC link could use this to intercept the communication between the backend and the tag and to, consequently, discover $k_{tag}$ and $k_{LT}$. Therefore, this scenario is only acceptable for use-cases where malicious users and active attackers on the NFC link do not pose a problem.

Finally, the latest generation of MIFARE DESFire, DESFire EV2, features delegated application management, which could significantly simplify enrollment of tags already used for other applications and issued by other service providers.

### *4.4. Security Analysis*

The security of the PT architectures relies on the combination of multiple factors for the creation of session keys, i.e., the NFC tag, the mobile app with an underlying key vault, as well as the channel protocols between the backend server and the client app. Each of these factors could potentially be attacked in order to get in control of one side of the channel. Note that we assume the backend system to be secure and not compromised by an adversary.

#### 4.4.1. Attacks on the NFC Tag

An adversary could potentially take over the content of an NFC tag by cloning the whole tag. This is a potential risk for PT1, which relies on NFC tags that do not provide any cloning protection. Still, the data stored on the tag are encrypted with keys only accessible by the app running inside the phone. Cloning or even stealing the tag alone is not useful to an attacker. In other words, the whole tag content is bound to one specific phone and, therefore, has no use to an adversary without access to that phone. For instance, this reduces the risk compared to an SMS-TAN-based authentication scheme since an attacker needs to get hold of two separate pieces of hardware. Nevertheless, we acknowledge

that users may tend to carry both components close to each other, which results in a high chance of both components being lost or stolen together.

A potential attack scenario in the PT2 architecture targets the communication between the tag and the app. The communication itself is encrypted with keys shared between the tag and the backend. Hence, the application does not have the capability to decrypt messages. This is the intended behavior as the actual long-term shared secret on the tag should never be stored in plain text on the phone. The encrypted message itself is used as a basis for establishing the secure channel between the app and the backend server (see Step 6 in Section 4.3.2). An adversary that is in close proximity to the user could therefore eavesdrop this key. However, unless the attacker also actively intercepts the authenticated key-agreement protocol between client and server, this information is not usable by the attacker.

Further attacks on the security of MIFARE DESFire itself (such as side-channel against the obsolete first generation of DESFire as in [25]) to extract the long-term shared secret ($k_{LT}$) or the mutual authentication key ($k_{tag}$) in PT2 are beyond the scope of this paper.

#### 4.4.2. Attacks on the Client-Side Targeting the User

There is no immediate need for additional user credentials, such as PIN or password (except for enrollment), when using PT1 or PT2. However, PT1 and PT2 may be used in addition to PIN/password authentication (e.g., by the mobile device itself or within the app). Not requiring the user to remember and type PINs/passwords improves the usability of the app. Moreover, adding a factor that does not rely on potentially weak passwords chosen by the user also improves security. Hence, an adversary trying to steal information by using social engineering on the user will not have success without access to the actual phone and tag.

Nevertheless, requiring the user to carry an additional item, the NFC tag, may also be considered a burden. Further user study is needed to evaluate if this actually improves the situation over PINs/passwords even though we intuitively assume this to be the case.

#### 4.4.3. Attacks on the Client-Side Targeting the Device

An adversary targeting the app on the mobile device could attempt to attack the key vault that is used to store session tokens and to encrypt and decrypt the tag information in PT1. The security of the keys thereby significantly relies on the implementation of the key vault. A hardware-backed key vault naturally provides a higher security level than a software-backed one as it stores the keys in additional secure hardware (e.g., in a trusted execution environment or a secure element). However, even if attackers gain control over the phone and manage to get access to the key material stored in the key vault, they would also need access to the NFC tag in order to make use of that key material. This implies that the app should never store the long-term secret key information of the tag to guarantee forward secrecy in case the device is compromised or stolen.

A malicious app could try to intercept the communication with the NFC tag. For instance, on an Android device, any app with access to NFC can read any tag. For PT1, this would allow an adversary to clone the whole tag contents (see Section 4.4.1) or to perform a denial-of-service attack by erasing the tag data. With PT2, the malicious app would not be able to obtain any useful data from the tag.

Compared to other methods such as a password or SMS-TANs, one-time access to the tag is only of limited use for an adversary. A password, once observed by the malicious app, remains usable until changed by the user. An app that registers to receive SMS messages can eavesdrop on all SMS-TANs sent to the device. Since SMS-TANs are automatically delivered, a malicious app could easily access all future SMS-TANs. When using NFC tags, the tag is only usable by the malicious app upon explicit user interaction (i.e., while a user explicitly holds it within the read range of their device). This is only the case when a user wants to perform an actual authentication. Thus, the use of an NFC tag improves security over pure password or SMS-TAN based solutions due to the requirement of an explicit user interaction ("human-in-the-loop").

If a malicious app manages to elevate its privileges in a way that it gains full control over the mobile device system, the app may be capable of intercepting both, the NFC channel and the communication channel with the backend. Similarly, the adversary app may be able to access the key vault or even any aspect of the app under attack. Our protocols cannot protect against such attacks. However, in order to successfully perform an authentication with PT2, an adversary would still need to have access to the actual NFC tag during authentication.

#### 4.4.4. Attacks on the App-to-Backend Channel

We assume that the cryptographic primitives used for the key agreement (e.g., ECDH) and the authenticated and encrypted channel (e.g., AES, etc.) are secure. Authenticating the app-to-backend channel to protect it against MITM attacks is one of the main objectives of the Protecting Touch architectures.

In both protocols, the key material used for authentication is only stored on the NFC tag. In PT1 that key material can only be decrypted by the user's phone. Similarly, in PT2, only a phone that proves to be in possession of the session token may access the key material on the tag. Thus, an adversary would need to have control of both devices in order to take over the communication between the app and the backend.

If the MITM attack happens during key enrollment of PT1 or PT2, the adversary would need to be in possession of the enrollment password $p_{enroll}$. By using an appropriate password-authenticated key-agreement protocol (such as SRP [24]), brute-force attacks on the password can be prevented during key-agreement. To further protect the key enrollment against MITM attacks, we propose to use an out-of-band channel. A hash over the shared secret $k_s$ of the initial communication establishment is thereby sent from the server to the client. The user needs to provide additional verification (such as a bank card and PIN for the mobile banking use case) in the used out-of-band channel. The adversary would, therefore, also need to be in control of the out-of-band channel or have the verification details of the user in order to successfully conduct an attack during enrollment.

Our proposed architectures PT1 and PT2 also provide forward secrecy for the communication between app and backend. If the session key $k_s$ is leaked in PT1, the adversary would be able to decrypt the session where this key is used. However, as the session key is derived from authenticated key-agreement using the one-time key $k_{OTK}$ as shared authentication key, it is not possible to generate the next session key $k'_s$ with this knowledge. Neither can a leaked $k_s$ be used to decrypt previously recorded sessions when using Diffie-Hellman protocol with ephemeral keys. Thus, this provides forward secrecy as it generates new keys every time a channel is established. The one-time key $k_{OTK}$ is read from the tag and used to protect against MITM attacks. An adversary who has access to the session key would only be able to generate this one-time key $k'_{OTK}$ of the very next session. In order to eavesdrop the communication of that next session, the attacker would also need to actively interfere during the key-agreement.

In PT2 the session key for the communication between the app and the backend is generated directly on the tag, derived from a long-term shared secret $k_{LT}$. This derivation is done by encrypting $k_{LT}$ with an ephemeral session key established between the tag and the backend ($k_{s,tag}$). This effectively generates a deterministic random session key seeded by $k_{LT}$ and the ephemeral key $k_{s,tag}$. Since $k_{s,tag}$ is randomly and independently generated, this also implies that each new session key is independent of all previous session keys. Hence, forward secrecy is provided in PT2. Forward secrecy would be compromised if the long-term tag authentication key $k_{tag}$ together with the long-term shared secret $k_{LT}$ stored on the tag itself are compromised. As a result, it is necessary that this long-term secret ($k_{tag}$ and $k_{LT}$) never leaves the tag (or the backend) in clear-text.

### *4.5. Comparison of PT1 and PT2*

With PT1 and PT2 we developed protocols for both ends of the spectrum of available low-cost NFC tags. Both protocols have a different trade-off between availability, usability, and security.

While PT1 can be implemented with any NFC tag that has a few hundred bytes of memory, PT2 requires a specific NFC tag product (MIFARE DESFire). Nevertheless, MIFARE DESFire also has good availability at low cost. NFC tags suitable for PT1 are available from USD 0.50 while tags suitable for PT2 start at USD 3.00 (based on prices for one piece from Tagstand (nfctags.tagstand.com) and nfc-tag.de (www.nfc-tag.de). In comparison, the dedicated NFC-based authentication token YubiKey NEO is currently sold at USD 50.00.

Tags used for PT1 can easily be cloned by an attacker with physical access or with the capability to eavesdrop on the NFC communication whereas PT2 provides good protection against cloning of tags. Even though such a cloned tag in PT1 is of limited use (cf. Section 4.4.1), this makes PT1 less secure than PT2. Moreover, since there is no reversible write-protection for NFC tags, PT1 is susceptible to denial-of-service through maliciously or accidentally overwriting the tag memory (e.g., by other apps).

With regard to user-experience (specifically the duration that users need to keep an NFC tag continuously attached to their smartphone), PT1 has the advantage that the tag needs to be read only once at the beginning of a protocol cycle and written only once at the end of the protocol cycle. Both operations are fast (e.g., the write operation for our OTK record (approx. 160 bytes) takes, depending on the tag hardware, between 100 and 300 ms in our prototype) and by using a double-tap mechanism (one tap before and one tap after negotiating the new one-time key with the backend server) the user interactions can be decoupled from network latency.

Moreover, the HMAC over the OTK data allows the app to pre-authenticate tags offline before connecting to its backend server. This could also be used to implement locking of functionality inside the app that is less critical and does not need communication with the backend server.

With PT2, the app transparently proxies parts of the authentication phase between the tag and the backend server. Consequently, the tag needs to be continuously attached to the smartphone during that phase of PT2. We measured a typical duration between 200 and 500 ms for this phase over UMTS network and over LTE when the connection to our backend server was previously established. However, we found that it may take up to 3 seconds to initially establish a connection over UMTS.

Our current implementation of PT1 and PT2 considers only one app per NFC tag. This means that users need to carry a separate tag for each app. However, our protocols can easily be extended to multi-application scenarios. In PT2, each app can have its own DESFire application on the tag. Thus, multiple apps would not interfere with each other. In PT1, each app could add its own OTK record to the NDEF message on the tag. This comes at the price that all apps also read all the OTK records of other apps, which results in similar risks as tag cloning.

## 5. Conclusions and Future Work

We assessed different types of existing NFC tag technologies with regard to their usability as tokens for authentication in smartphone apps. Based on the requirements of an exemplary mobile banking app, we created the Protecting Touch (PT) architectures, two concepts for a user-friendly secure authentication mechanism for mobile apps and their backend systems based on NFC tags. As opposed to existing approaches to authentication with NFC-compatible tokens, our approach uses key material obtained from the tags to secure the whole communication.

We analyzed the two protocols PT1 and PT2 with regard to their security properties and created prototypes to verify that both can be implemented for recent Android devices. The two protocols were designed to leverage the features of both ends of the spectrum of available NFC tag hardware resulting in different trade-offs in terms of availability, security and usability. While PT1 works with virtually any NFC tag, tags are widely available at lowest cost and tags can be pre-authenticated offline, it suffers from a limited, but remaining, risk of tag cloning. PT2, on the other hand, provides good protection against tag cloning. However, this comes at the price of requiring real-time communication between the backend and the tag over the mobile network. Tags are slightly more expensive (though still low-cost compared to other solutions) and also have good availability.

Finally, we mainly base our assumptions that NFC tags provide a better trade-off between security and usability than passwords on mobile devices and that our solution is, hence, more user-friendly on various studies criticizing passwords. Future work, e.g., in the form of a user study, needs to quantitatively assess if carrying an NFC tag is indeed more user-friendly than choosing, remembering and typing strong passwords.

**Acknowledgments:** The first author would like to thank Banco do Brasil S/A for funding the international collaboration at Johannes Kepler University Linz. A special note of thanks to the Institute of Networks and Security for hosting and supporting this author. Moreover, this work has been carried out within the scope of the Josef Ressel Center for User-Friendly Secure Mobile Environments ("u'smile"), funded by the Christian Doppler Gesellschaft, A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, NXP Semiconductors Austria GmbH and Österreichische Staatsdruckerei GmbH.

**Author Contributions:** Fernando Kaway Carvalho Ota, Michael Roland and Michael Hölzl designed and evaluated the protocols; René Mayrhofer and Aleardo Manacero supervised the design of the protocol; Michael Roland, Fernando Kaway Carvalho Ota and Michael Hölzl wrote the paper.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Yan, J.; Blackwell, A.; Anderson, R.; Grant, A. Password Memorability and Security: Empirical Results. *IEEE Secur. Priv.* **2004**, *2*, 25–31.
2. Greene, K.K.; Gallagher, M.A.; Stanton, B.C.; Lee, P.Y. I Can't Type That! P@$$w0rd Entry on Mobile Devices. In *Human Aspects of Information Security, Privacy, and Trust (HAS 2014)*; LNCS; Springer: Cham, Switzerland, 2014; Volume 8533.
3. Schartner, P.; Bürger, S. *Attacking mTAN-Applications Like e-Banking and Mobile Signatures*; Technical Report, TR-syssec-11-01; University of Klagenfurt: Klagenfurt, Austria, 2011. Available online: https://syssec.at/Publikationen/TR1101.pdf (accessed on 3 July 2017).
4. Konoth, R.K.; van der Veen, V.; Bos, H. How Anywhere Computing Just Killed Your Phone-Based Two-Factor Authentication. In *Financial Cryptography and Data Security (FC 2016);* LNCS; Springer: Berlin, Germany, 2016; Volume 9603.
5. FIDO Alliance. FIDO NFC Protocol Specification v1.0. 2015. Implementation Draft. Available online: https://fidoalliance.org/specs/fido-u2f-nfc-protocol-id-20150514.pdf (accessed on 3 July 2017).
6. FIDO Alliance. Universal 2nd Factor (U2F) Overview. 2015. Proposed Standard. Available online: https://fidoalliance.org/specs/fido-u2f-overview-ps-20150514.pdf (accessed on 3 July 2017).
7. Yubico. YubiKey NEO – Premium Strong Two-Factor Authentication for Secure Logins. Available online: https://www.yubico.com/products/yubikey-hardware/yubikey-neo/ (accessed on 3 July 2017).
8. León-Coca, J.M.; Reina, D.G.; Toral, S.L.; Barrero, F.; Bessis, N. Authentication Systems Using ID Cards over NFC Links: The Spanish Experience Using DNIe. *Procedia Comput. Sci.* **2013**, *31*, 91–98.
9. Horsch, M.; Braun, J.; Wiesmaier, A. *Mobile eID Application for the German Identity Card*; Technical Report TUD-CS-2011-2918; TU Darmstadt: Darmstadt, Germany, 2011. Available online: http://www.cdc.informatik.tu-darmstadt.de/reports/TR/Mobile_eID_app_for_the_German_ID_card.pdf (accessed on 3 July 2017).
10. WISeKey. VaultIC150/150D/152. Available online: https://www.wisekey.com/vaultic/secure-solutions/vaultic150-150d-152/ (accessed on 3 July 2017).
11. Urien, P. Introducing TLS-PSK authentication for EMV devices. In Proceedings of the International Symposium on Collaborative Technologies and Systems (CTS 2010), Chicago, IL, USA, 17–21 May 2010; pp. 371–377.
12. Elenkov, N. Dissecting Lollipop's Smart Lock. Android Explorations Blog. 2014. Available online: http://nelenkov.blogspot.com/2014/12/dissecting-lollipops-smart-lock.html (accessed on 3 July 2017).
13. Duc, D.N.; Lee, H.; Konidala, D.M.; Kim, K. Open issues in RFID security. In Proceedings of the International Conference for Internet Technology and Secured Transactions (ICITST 2009), London, UK, 9–12 November 2009; pp. 1–5.

14. NFC Forum. Type 1 Tag Operation, ver. 1.2. 2014. Technical Specification. Available online: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/tag-type-technical-specifications/ (accessed on 3 July 2017).
15. NFC Forum. Type 2 Tag Operation, ver. 1.2. 2014. Technical Specification. Available online: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/tag-type-technical-specifications/ (accessed on 3 July 2017).
16. NFC Forum. Type 3 Tag Operation, ver. 1.2. 2014. Technical Specification. Available online: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/tag-type-technical-specifications/ (accessed on 3 July 2017).
17. NFC Forum. Type 4 Tag Operation, ver. 2.0. 2011. Technical Specification. Available online: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/tag-type-technical-specifications/ (accessed on 3 July 2017).
18. NFC Forum. Type 5 Tag Operation, ver. 1.0. 2015. Technical Specification. Available online: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/nfc-forum-technical-specifications/ (accessed on 3 July 2017).
19. NFC Forum. NFC Data Exchange Format (NDEF), ver. 1.0. 2006. Technical Specification. Available online: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/data-exchange-format-technical-specification/ (accessed on 3 July 2017).
20. Murdoch, S.J.; Drimer, S.; Anderson, R.; Bond, M. Chip and PIN is Broken. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, 16–19 May 2010; pp. 433–446.
21. Shin, S.; Kobara, K.; Imai, H. Leakage-Resilient Authenticated Key Establishment Protocols. In Proceedings of the Advances in Cryptology (ASIACRYPT 2003), 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, 30 November–4 December 2003; Springer: Berlin, Germany, 2003; Volume 2894, pp. 155–172.
22. Sahaa, M.; Roy Chowdhurya, D. Provably secure key establishment protocol using one-way functions. *J. Discret. Math. Sci. Cryptogr.* **2009**, *12*, 139–158.
23. M'Raihi, D.; Bellare, M.; Hoornaert, F.; Naccache, D.; Ranen, O. *HOTP: An HMAC-Based One-Time Password Algorithm*; RFC 4226; December 2005. Available online: https://tools.ietf.org/html/rfc4226 (accessed on 3 July 2017).
24. Wu, T. The Secure Remote Password Protocol. In Proceedings of the Network and Distributed System Security Symposium (NDSS 1998), San Diego, CA, USA, March 1998. Available online: http://www.isoc.org/isoc/conferences/ndss/98/wu.pdf (accessed on 3 July 2017).
25. Oswald, D.; Paar, C. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In Proceedings of the Cryptographic Hardware and Embedded Systems (CHES 2011), Nara, Japan, 28 September–1 October 2011; LNCS; Springer: Berlin, Germany, 2011; Volume 6917.