VeinoCert: Binding an Object to an Owner

Serge Vaudenay

EPFL, Lausanne, Switzerland

Abstract. We define a protocol by which we can recognize if a person is the owner of an object. The object can, for instance, be an official document such as a diploma. In our model, the object has an attached RFID chip. The owner is enrolled when the document is created and the chip is attached. Later on, public verifying terminals can verify if a person is the enrolled owner by means of biometric recognition. Hence, the terminal must scan both the chip and the person. As an implementation demonstrator, we use fingervein biometry. Our system can also be used for access control to an online repository to get more information and services related to the object. We require strong security and privacy levels such as: a secure owner recognition and access limited to the legitimate owner holding the right document, the principle of least privilege, and no storage of biometric data at rest. Our solution relies on an inexpensive off-the-shelf RFID chip.

1 Introduction

Showing that a claimant owns an official document is quite frequent in daily life. As an example, such document can be a diploma, a training certificate, a driving license, or even a vaccine certificate. Let us call it a certificate for the moment to avoid confusion with an ID document. In such cases, the legitimate owner joins some education program, some training, some driving tests, or some vaccination campaign. The owner who completes the requirements is typically identified with an official identity document, and the identity is copied onto the issued certificate. Later on, a claimant can show their ID document and the certificate. Ownership of the certificate is verified by identifying the person as on the ID document and comparing the printed identity on both the ID document and the certificate.

This approach has several obvious problems. First of all, identifying a person based on an ID document is often poorly done in a manual manner, if any: this typically consists of comparing a small facial picture which was taken several year ago with a person with potentially new haircut and glasses. The picture was printed a long time ago and printing degraded as many things happened to the ID document during this time. It is not even sure that verifiers always care about verifying the picture or are even trained for that. Second, this implies trust in the ID document architecture and the respective issuing authority. There are different security levels in the ID document types and across regions. Some ID issuers could actually act maliciously against the interest of the certificate

application. Third, this actually reduces the recognition of the certificate owner to the recognition of the ID document owner by doubling the problem: now, we must recognize the ID document owner when we issue the certificate and when we verify the ownership of the certificate. Finally, providing an ID is a privacy loss. We argue that claiming ownership does not necessarily imply to provide an identity. The problem is only to recognize that the person who enrolled as an owner and the person who claims ownership are the same. Indeed, identities are used across applications and this creates privacy concerns.

We propose to eliminate the need for an identity document in the loop. Typically, the certificate is attached to an RFID chip. This chip is configured upon the creation of the certificate and when the owner of the certificate is enrolled. Enrollment happens by capturing a biometric model of the owner. Later on, recognition of the owner is done using biometric recognition. The chip is used to help to recover the biometric model in order to verify the matching with a biometric probe.

Essentially, the model is stored in the database of a biometric server. However, privacy concerns impose to prevent storing biometric data at rest. Instead, the biometric model is stored encrypted in the server, with a decryption key which is stored in the chip (and only in the chip). Biometric matching is done on the biometric server side. This server should actually be the only element in the architecture which could see the biometric data. To avoid cloning a chip, the chip should identify itself to a chip server. For privacy reasons, we do no want the chip server to be able to identify a user and we do not want the biometric server to be able to identify a chip.

It is becoming common to have specific repositories for certificates, as it gives an online service which can be used to verify non-revocation, to provide more metadata, to provide an official digital signature, or to offer more services in general. Typically, access to the repository is done using some code, link, or QR code which is printed on the certificate. Obviously, security and privacy with this technique is weak since anyone who saw (or guessed) the printed access information is getting some permanent and transferable access to the repository. In our system, we can have a more fine-grained access control to the repository. Essentially, we can ensure that access is limited in time and also restricted to passing the owner recognition.

The *certificate* can actually be a much more general object for which the notion of *owner* is defined. It could actually be an access badge or a new ID document. Hence, we call it a *document* in the rest of the paper.

As an example for implementation, we used fingervein biometry. We built up on tools developed for the architectures of BioID [8] and BioLocker [12]. We use a biometric scanner which is trusted and encrypting the biometric images with the embedded public key of the biometric server. We also use simple off-the-shelf RFID tags which have basic authentication mechanisms like HOTP [19] but insufficient storage to store biometric templates.

Our system builds up on an existing "RUDI" infrastructure for documents D: we assume an issuer I, users (owners and holders) U, and a repository R.

We add "CATS": chips C, an issuing authority A (working together with I), verifying terminals T, and two servers S_C and S_U for the chips and the users. The legitimate owner U of D is defined by A who also sets up C. Our goal is twofold: to enable T to verify that the holder of D is actually the owner U (owner recognition) and remote access control of the owner to a separate service provider R (access control).

Use-cases. One application could be the verification of diploma ownership. A student would enroll with biometry, graduate, be given a (physical) diploma with a chip, and later on be able to prove that they are the owner of this document. In this application, we need ownership recognition, but the remote access control to a repository R would also be helpful as well. Access to R would provide a digital copy of the diploma with a digital signature from the university, which helps in detecting falsification. It can also be used to check possible revocation of a diploma, or to provide more information such as a grade sheet. With a solution based on a static QR code, a malicious terminal can always replay whatever QR code that was scanned before and get another access to R without the consent of the owner, or even sell what was scanned. Our approach based on a chip is more privacy-preserving as it enables fine-grained access control.

Another application (which we hope will never happen again) would be to prove the ownership of a COVID vaccine certificate to enter in a restaurant. It avoids having to reveal an identity, a date of birth, or a nationality, which the restaurant would otherwise be able to sell or to use for discrimination.

We can also use our protocol for identity documents such as passports or driver licenses with cheaper chips (but online servers).

Related Work. Some projects such as W3C verifiable credentials [2] also apply on proving that a person holds a diploma. In their case, they prove that a diploma (which is not shown) was given to a person of a given ID. This application does not apply to a physical document nor to a physical person. The problem of recognizing if a holder corresponds to the ID is left. In our case, we bind a physical document to a physical person.

The biometric passport (MRTD) or mobile driver license (mDL) [1] has more similarities. In these applications, there is a device with a powerful and expensive chip with Javacard (while our chip is inexpensive) and no need for a remote server (while we use an online server). Furthermore, these applications have a limited template protection. Essentially, either the reference template leaks to the verifier, or we need to limit access to the chip to "authorized verifiers" (meaning we need a central authority to grant access and the chip to reliably verify access credentials and possible revocations), or we hope that the matching would be done by the chip, thus need a stronger trust assumption in that chip.

Some other project such as Private Eyes [26] focus on the root problem of authenticating a user on a remote server, without specifying how to use the output (like grant access to another service R). They offer an elegant and very simple and efficient protocol, where the reference template is shared between the chip and the server. However, their protocol restricts to some biometric technique

which is based on the Hamming metric, which works well for iris recognition but would not work for fingervein recognition (our protocol is agnostic). They need some large storage on the chip. Typically, a template share takes 256B, which is beyond the storage capacity of the chips that we use. Their protocol is stateful as the stored shares keep updating. This implies some reliability concerns: if it happens that the probed template is too far from the reference template (false negative can happen), then the chip and the server end up being desynchronized and can no longer authenticate the user. Even if false negatives are rare, a malicious user could inject errors during the transmission to break synchronization. Our protocol is stateless and the storage on the chip does not need update. There is no synchronization problem. Finally, their protocol (like ours) would need specific protections against hill climbing attacks. Otherwise, after corrupting the chip, an adversary can interact with the server with many requests and fully decrypt the reference template.

Our contribution. We design an architecture allowing to recognize if a person is the registered owner of a registered document and to perform access control to services about the document. It builds on the top of an existing architecture for registered documents and online access to a digital repository. We register and recognize the owner by using privacy-preserving biometric techniques in conjunction with RFID technology. For this, we construct a key hierarchy to prevent unauthorized access to private data. Recognizing the owner also comes with access control functionalities to the digital repository. Our system is efficient and offers pragmatic solutions.

We implemented a proof-of-concept.

Our RFID chip stores little information (typically: 64B) which is static and is non-programmable. The chip plays two roles: it provides a time-dependent authenticator which plays a role in remote access control; it also proves that the holder consents to have ownership verified.

Several details appear only in the appendix of this paper.

2 Architecture Overview

Our notations are summarized in Table 1.

2.1 Participants

Our architecture involves several participants (as depicted on Figure 1).

- (D) official documents
- (R) document repositories
- (I) document issuing authorities who print D and write in R (involved for document creation only)
- (U) (human) document owner (there is one owner per document)
- (S_C) one chip server which maintains a database

Table 1. Notations

D, R, I, U, S_C, S_U, C, A, T	participants in our protocol
ek_{SC}, dk_{SC}	encryption key pair of S_C (encryption and decryption)
sk_{SC}, vk_{SC}	signature key pair of S_C (signature and verification)
sk_A, vk_A	signature key pair of A (signature and verification)
k_{SC}, k'_{SC}	symmetric keys of S_C
k_{SU}, k'_{SU}	symmetric keys of S_U
rid_C	record identifier in the chip database
rid_U	record identifier in the biometric database
eerec_C	doubly-encrypted record in the chip database
erec_C	encrypted record in the chip database
rec_C	record in the chip database
last	latest access time and results for a chip record
ts	authentication challenge for C
$\mathsf{ts}_{\mathsf{old}}$	last value of ts in a successful authentication
flag_U	flag to indicate if U ownership recognition is used
flag_R	flag to indicate if R access control is used
flags	allowed combination of $(\mathsf{flag}_U, \mathsf{flag}_R)$
otk_C	one-time key to encrypt rec_C
otk_U	one-time key to encrypt bio_U
ak_C	authentication key of C
eid_U	encrypted identifier for U
com_A, r_A	commitment by A and opening value
com_S, r_S	commitment by S_C and opening value
σ_A	signature by A
ebio_U	encrypted biometric template of U
bio_U	biometric template of U
p_U	identifier hash of U
id_C	identifier of C
sn	serial number of D
slot	time slot to access an entry in R
tok	access token to R
G_1, G_2, G_3	three generators of a Diffie-Hellman group
H	hash function
PRF	pseudorandom function
AEEnc, AEDec	authenticated encryption (encryption and decryption)
OTEnc, OTDec	one-time encryption (encryption and decryption)
MAC	message authentication code
Sign, Verify	signature algorithms (signature and verification)
PoK	proof of knowledge

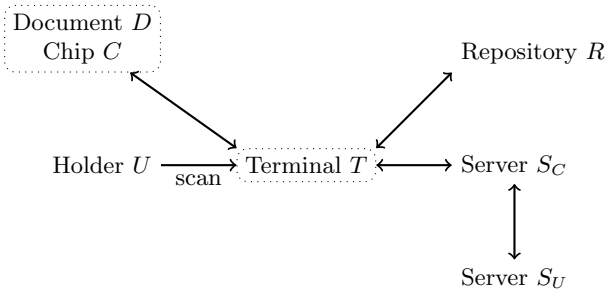

Fig. 1. Authentication Architecture (without I and A)

- (S_U) one biometric server which maintains a database
- (C) IC chips attached to the document (one chip per document; one document per chip)
- (A) issuing authorities who configure C and S by scanning U (involved for registration only)
- (T) verifying terminals

We call the RUDI architecture, the one using only owners U, documents D, an issuer I, and possibly a repository R. Our aim to to build upon the RUDI architecture by adding CATS: chips C, an authority A, terminals T, and servers S.

Document *D*. The document D has a serial number sn which may be printed on it. This number must be unique (we cannot have two existing documents with the same serial number). The document has several attributes which can be printed and/or stored in a repository. How the document is structured and what attributes are defined is independent from our system.

Repository *R*. A repository R stores a copy of some documents D, their serial number sn, and an access policy to define who can read which attribute in the document D. Indeed, D has several attributes. Some could be public. Some others could be accessed only by the holder (i.e., scanning the chip is enough to get access). Some others could be accessed only by the owner (i.e., after biometric recognition). Some could be accessed by subscribers.

Access to some attributes could be logged or not (for instance to let the owner know). There can be several repositories to store the same document. Ideally, it should be up to the owner to decide which repository to use and which access policy to use. However, the chip C points to a single R which is defined at enrollment.

We assume secure communication to repositories (e.g. with TLS [22]). The design of R is independent from our system. In our system, we only need to add in R some access control verification mechanisms.

Issuer I. At the beginning, the issuer I builds the official document D.

Owner/Holder U. The owner/holder U is a person, which we model as a biometric source which can be *scanned*. Whenever U is scanned to register a biometric model, a new identifier id_U is privately generated. It is only used at the issuance of a document. This identifier is independent from the identity of U. This identifier could be fresh for every document issuance (this would avoid linking documents with same owners) but a user can decide to reuse the same identifier and the biometric model for several documents. Hence, it is up to U to decide to remember id_U. There is no way to recover it otherwise, as it is stored nowhere.

Server S_C. The chip server S_C has the authority to grant access to a repository for a specific time slot to a document related to a specific chip C. However, S_C knows neither the document nor the owner. The chip server S_C maintains a database where each chip corresponds to an entry in the database. It contains information limited to the chip and access control. The chip server has two key pairs (so two public keys). One is used by anyone to communicate to S_C securely. The other is used to authenticate tokens which have been issued by S_C.

Server S_U. The biometric server S_U is an oracle telling if a given biometric probe matches the biometric model of a specific enrolled owner. The biometric server S_U maintains a database where each enrolled owner corresponds to an entry in the database. It contains the encrypted biometric models. The biometric server has a key pair (so one public key) to allow anyone to communicate to S_U securely.

Chip C. The IC chip C is associated to a unique identifier id_C. It should prove itself to the chip server S_C and send information about C and U to the servers. Then, the terminal T could send a biometric probe bio' and get a 0/1 result.

Access to S_C by C are limited in number or in time. (We call it the rate limit.) This is in order to prevent hill climbing attacks where an adversary would reconstruct the biometric model by making many queries. After a holder has been recognized as being the owner and was granted access to R, we do not anticipate applications requiring to do another recognition associated to the same chip. Hence, we could block access for 15 minutes after a positive authentication, as no more authentication is needed, and also after 10 consecutive negative authentications, which could be a sign of an attack. So, the record entry for the chip on S_C should keep a field last indicating the access attempts and their results during the 15 last minutes.

Authority A. The participant A has authority to attribute an owner U to a document D (either by deciding who is the owner or by carefully verifying

if a person should be the legitimate owner). At enrollment of an owner U, the authority A takes the identifier id_U chosen by U, captures a biometric model bio_U of U, and adds a new record in the database of S_U. At registration of a document D with a new chip C, the authority A gets id_U from U, recognizes U using S_U and a biometric probe bio', takes a fresh identifier id_C, sends information to S_C to create a new entry, and stores information on C.

Terminal T. The terminal T is interacting with all other participants. It can optically read the document, interact with the chip, prompt the user to scan their biometry, query the server, and query the repository.

Biometric scanner. In this document, an implicit part of A and T is a biometric scanner which can extract biometric captures of a user U. This biometric scanner could directly encrypt the captured biometric information to be decrypted by S_U. We think of this biometric scanner to be compatible with previously proposed infrastructures such as BioID [8] and BioLocker [12].

2.2 Security and Privacy Objectives

The goal of our protocol is to address the following threats.

1. (Ownership theft) A user convinces T that they are registered as the owner U of C although they are not.
2. (Access violation) T gets access to R for a service related to a chip C without approval by the holder U.
3. (Non-consented ownership verification) T interacted with the server to check the ownership of a chip C without having scanned C recently.
4. (Privacy breach) An adversary gets sensitive information such as a reference biometric template, or the matching between a template and a document, or the equality of owners of chips.

Informally, the goal of the protocol is to make S_C assess on whether U is the registered owner of D and to grant access to some restricted attributes through R. Access to R is limited to some time slot denoted by slot.

In our security model, I, A, S_C, S_U, and R are *normally* trusted to preserve the security of the infrastructure. We assume a secure distribution of the public keys vk_{SC} and vk_A to R and of ek_{SU} to biometric scanners. We assume secure communication channels $A \leftrightarrow S_C$, $A \leftrightarrow S_U$, $S_C \leftrightarrow S_U$, and $T \leftrightarrow R$. (This could be done by using TLS [22].) We assume that R and S_C are loosely synchronized on the notion of current time.

However, we relax these assumptions in privacy analysis. We consider three different privacy models. Servers can be honest-but-curious (i.e. they do follow protocols but would like to learn information). Another model is when their cryptographic secret keys are well protected (e.g. by specific secure hardware) but their entire storage (namely: the databases) are not. Yet another model is when they are fully malicious but receive no external information.

Terminals T are untrusted by default. To prevent collecting biometric templates, we should however assume either T to be honest or the scanners to encrypt scans with the public key of S_U.

We further assume that data acquisition by scanners is trusted. An adversary cannot replicate the biometrics *on* the sensor, i.e. run a spoofing attack between the physical world and the sensor. However, spoofing between the sensor and the transmission is possible (but addressed). This is a malicious replication of the biometrics provided *by* the sensor. (We discuss it in Appendix D.2).

We assume that chips C are normally tamper-resistant. So, they are uncloneable. Some C may however be corrupted to be cloned or reprogrammed.

The document D may be tampered. However, our system can be used to defeat such attack: a counterfeit D could be detected by providing a digital copy of D on R.

We assume that the biometric method is secure: if we make a model by scanning U and we present a probe which matches the model, then it must be the case that the probe was scanned from U, except for rare false matching cases. However, an adversary may try to produce many synthetic probes until one is accepted as a false matching case. (However, our system is blocking multiple attempts.)

2.3 Results

We list our security results below. They are formally analyzed in Appendix A.1 and A.2.

Security of ownership recognition. If A, S_C, S_U, and T are honest and T is presented a chip C and a holder U', while A registered C with an owner U, ownership recognition passes if and only if $U = U'$, assuming that biometry is secure, i.e. except in rare false matching cases. This result does not depend on RFID security: C could be corrupted.

Security of consent and access control to R. If A, S_C, S_U, and R are honest and R is presented a valid access token for sn during a time slot slot, while sn is the serial number of a document D registered by A with a chip C and an owner U, then it must be the case that at the beginning of slot, S_C was presented

- (consent) a valid authenticator obtained from C before and that neither this authenticator nor any more recent one was presented to S_C before (assuming that C was not tampered);
- (access control) and a biometric scan of U (assuming that biometry is secure).

Depending on RFID security, this results shows that we need the latest access authenticator of C and a biometric scan of U to obtain the access rights. We

could further guarantee that the biometric scan is fresh and/or recent as an extension of our system, following the extension proposed in Appendix D.2.

We can make this access control feature fit the Chen-Vaudenay model [13]. The setup phases sets the ak_C key in the chip and the key pair for SU, then $\mathsf{esk} = \mathsf{ek}_{SU}$, $\mathsf{psk} = (\mathsf{ek}_{SU}, \mathsf{ak}_C)$, $\mathsf{csk} = (\mathsf{dk}_{SU}, \mathsf{ak}_C)$. Enrolling sets em to $\mathsf{Enc}_{\mathsf{ek}_{SU}}(0\|\mathsf{bio})$. Probing sets pm to $(\mathsf{ebio}', \mathsf{ts}, \mathsf{resp})$. We obtain $\{\mathsf{esk}, \mathcal{O}_{\mathcal{B}}, \mathcal{O}_{\mathsf{Probe}}\}$-UF security: an adversary who knows ek_{SU}, who can get templates from the user (with $\mathcal{O}_{\mathcal{B}}$), and who can get $(\mathsf{ts}, \mathsf{resp})$ pairs without being allowed to submit them for matching cannot forge some em which will match the user.

Privacy. We assume an honest U who was registered by A as owning a document D which is attached to a chip C. For privacy, we guarantee

- (honest-but curious servers) that only the scanners and S_U see biometric templates of U, except if one of them is corrupted;
- (server database leakage) that if A, S_C, and S_U are honest but the databases leak, beyond identifying which record corresponds to C and U, no private information leaks (except the result of attempts to get access during the last few minutes before leakage);
- (full server corruption) that if registration was honestly done, then after S_C and S_U become corrupted, they cannot see any information about D, C, or U without any additional input (i.e. until a new authentication with C or a new chip registration with the same id_U).

In the Chen-Vaudenay model [13], enrollment and probing just use public-key encryption and their result directly gives $\{\mathsf{esk}, \mathsf{psk}, \mathsf{em}, \mathcal{O}_{\mathsf{pm}}\}$-IND security: given the encrypted biometric model em in the enrollement of either user U_0 or user U_1 and both templated generators, the generator of encrypted templates pm for the enrolled user, the public key ek_{SU}, the chip secret ak_C, the adversary cannot guess which user was enrolled.

3 Not Covered in this Document

As already stated, we build on the top of an existing RUDI infrastructure and we make a series of security assumptions for the additional CATS. These assumptions are application-dependent and must be carefully addressed. We list below what is not covered in this document.

- What to put in R.
- The structure of R and D.
- The structure of sn and how to determine which R and A to use.
- The interaction between U and R through T at the application level.
- The authorization mechanism for A to act as an authority.
- The authorization mechanism for T to access to S_C.
- The distribution of the public keys ek_{SC}, vk_{SC}, vk_A to terminals T (by means of a PKI).

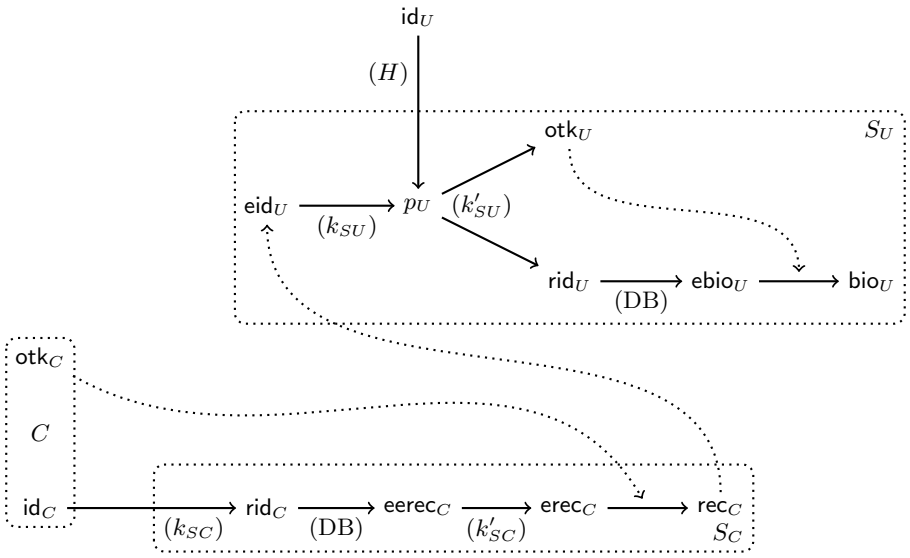

Fig. 2. Key Hierarchy

- The distribution of the public keys vk_{SC} and vk_A to repositories R.
- How servers are protecting their secret keys (e.g. using trusted hardware).
- How to secure the communications $A \longleftrightarrow S_C$, $T \longleftrightarrow R$, $S_C \longleftrightarrow S_U$.
- How the biometric scanner works and liveliness detection.
- Which biometric matching techniques are used.

4 The Protocols

We are going to specify our three protocols.

1. **Biometric Registration.** A user U selects an identifier id_U (which could be random) to register to A with biometry. A interacts with S_U.
2. **Chip Registration.** A user U with registered id_U registers to A a chip C attached to a document D. A interacts with S_U and S_C.
3. **Authentication.** A user U authenticates to T with a chip C and biometry. T interacts with S_C and later accesses to R for services.

The protocol allows T to verify that a user U is the owner of D coming with C and/or to grant access to the repository R where more services about D are provided. The latter authentication protocol has two option flags. The flag flag_U indicates that T wants to proceed to the ownership verification, and therefore use biometry. The flag flag_R indicates that T wants to access to R so needs access control. The protocol simplifies when some flags are down (for instance, T needs no biometric scanner if flag_U is down) but offers less functionality.

4.1 Keys and Databases

We define in more details the key hierarchy, the databases, and the storages. The hierarchy is depicted on Figure 2.

Server S_C. The chip server S_C holds two symmetric keys k_{SC} and k'_{SC} and two key pairs. A key pair $(\mathsf{ek}_{SC}, \mathsf{dk}_{SC})$ is used to enable anyone to communicate privately with S_C, where ek_{SC} is public. A key pair $(\mathsf{vk}_{SC}, \mathsf{sk}_{SC})$ is used for authenticating tokens from S_C, where vk_{SC} is public. It is used to sign tokens issued by S_C, to be verified by a repository R or a terminal T.

The server S_C maintains the chip database. The chip database contains one record per identifier rid_C. Each record has form

$$[\ \mathsf{rid}_C\ ,\ \mathsf{eerec}_C\ ,\ \mathsf{last}\ ,\ \mathsf{ts}_{\mathsf{old}}\ ,\ \mathsf{flags}\ ]$$

where last is a field which is used to monitor recent access to this record and possibly block it if too often used. The value $\mathsf{ts}_{\mathsf{old}}$ is used if the chip only allows one-time key authentication like HOTP [19] or TOTP [20], to check what was the lastly used challenge and check that the new one (chosen by the chip) is coherent (i.e. increasing, typically). The flags flag_U and flag_R respectively indicate if the record is to be used for U ownership recognition and R access control. flags indicate which combination of $(\mathsf{flag}_U, \mathsf{flag}_R)$ can be used for this record. eerec_C is the encryption of erec_C using k'_{SC}. erec_C is the encryption of rec_C using a one-time key otk_C (which is provided by C). The structure of rec_C is as follows.

$$\mathsf{rec}_C = [\ \mathsf{ak}_C\ ,\ \mathsf{eid}_U\ ,\ \mathsf{com}_A\ ,\ \sigma_A\ ]$$

The key ak_C is the HOTP chip authentication key. It is used to authenticate C to S_C. The commit value com_A and the signature σ_A are provided by A at set up. The eid_U is an encrypted identifier which is understood by S_U.

The key k_{SC} is used to derive the record identifier rid_C from id_C, the chip identifier. The key k'_{SC} is used to decrypt eerec_C into erec_C. Namely, we set

$$\begin{aligned} \mathsf{rid}_C &= \mathsf{PRF}_{k_{SC}}(\mathsf{id}_C) \\ \mathsf{eerec}_C &= \mathsf{AEEnc}_{k'_{SC}}(\mathsf{erec}_C) \\ \mathsf{erec}_C &= \mathsf{OTEnc}_{\mathsf{otk}_C}(\mathsf{rec}_C) \end{aligned}$$

Server S_U. The biometric server S_U holds a key pair $(\mathsf{ek}_{SU}, \mathsf{dk}_{SU})$ to enable anyone to communicate privately with S_U, where ek_{SU} is public, a symmetric encryption key k_{SU}, and a symmetric PRF key k'_{SU}.

The server S_U maintains the owner database. It contains one record per identifier rid_U. Each record has form

$$[\ \mathsf{rid}_U\ ,\ \mathsf{ebio}_U\ ]$$

where ebio_U is the encrypted biometric template. The key k_{SU} is used to decrypt the owner identifier eid_U into a pseudonym (ID hash) p_U. The key k'_{SU} is used

to derive a record identifier rid_U and a key otk_U from p_U. The one-time key otk_U is used to decrypt ebio_U into bio_U. Namely, we set

$$\begin{aligned} p_U &= \mathsf{AEDec}_{k_{SU}}(\mathsf{eid}_U) \\ (\mathsf{rid}_U \| \mathsf{otk}_U) &= \mathsf{PRF}_{k'_{SU}}(p_U) \\ \mathsf{ebio}_U &= \mathsf{OTEnc}_{\mathsf{otk}_U}(\mathsf{bio}_U) \end{aligned}$$

S_U may also implement protection against ebio'_U replay attacks (as discussed in Appendix D.2).

Chip C. Among the above quantities, the chip C only stores what follows:

$$[\ \mathsf{id}_C\ ,\ \mathsf{otk}_C\ ,\ \mathsf{ak}_C\ ,\ \mathsf{sn}\ ,\ r_A\ ,\ \mathsf{flags}\ ]$$

The value id_C identifies C to S_C. The key ak_C is used to authenticate C to S_C. The key otk_C is provided to S_C to allow S_C to read rec_C. Values sn and r_A are used to generate the access token to get access to R.

Owner U. The owner U only needs to remember id_U in order to register documents under a registered biometric identity. This identifier is randomized into p_U by a hash function.

$$p_U = H(\mathsf{id}_U)$$

Unless the user wants to reuse the biometric information, the identifier can be forgotten. Actually, id_U could be generated/used/deleted by A directly when registering an owner and a document at the same time.

Authority A. The authority A has a key pair $(\mathsf{vk}_A, \mathsf{sk}_A)$ which is used to bind sn to C. We assume that R and T know vk_A and vk_{SC}.

Terminal T. T selects an ephemeral symmetric k_T to be sent to S_C. It is used by S_C to answer to T privately.

4.2 The Authentication Protocol

The authentication protocol sets two flags flag_U and flag_R to determine if we need U ownership recognition and R access control, respectively. The record of the chip in the S_C database must allow this combination. At a high level, the authentication protocol runs as follows (see Figure 3).

1. T activates C.
2. C answers by providing id_C, otk_C, sn, r_A, flags and the response authenticator $(\mathsf{ts}, \mathsf{resp})$ (based on ak_C: such that $\mathsf{resp} = \mathsf{MAC}_{\mathsf{ak}_C}(\mathsf{ts})$).
   [The above works in the case of an HOTP [19]-type authenticator where the challenge ts is chosen by the claimant. In the HOTP case, ts should match the one of the previous authentication session with a random but small increase.]

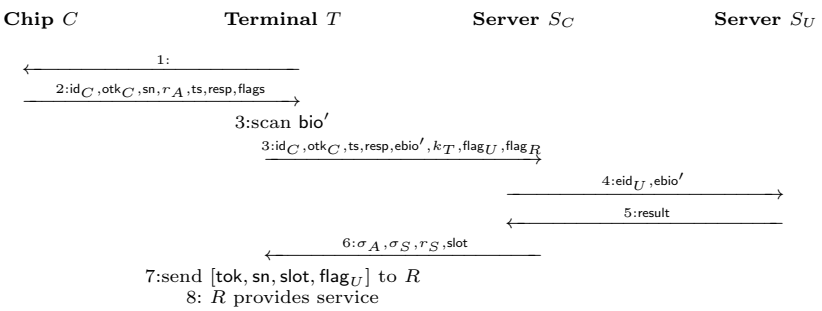

Fig. 3. Authentication Protocol (with Ownership Recognition and Access Control)

3. T performs the following steps: T checks that flags allows the combination of flag_U and flag_R T scans a biometric probe bio' of U (abort if not). T computes $\mathsf{ebio}' = \mathsf{Enc}_{\mathsf{ek}_{SU}}(1 \| \mathsf{bio}')$ (this is normally done by the trusted scanner). T picks a random k_T. T sends $\mathsf{Enc}_{\mathsf{ek}_{SC}}(\mathsf{id}_C, \mathsf{otk}_C, \mathsf{ts}, \mathsf{resp}, \mathsf{ebio}', k_T, \mathsf{flag}_U, \mathsf{flag}_R)$ to S_C. (If flag_U is false, bio' and ebio' are not computed and there is no biometric scan.)
4. S_C performs the following steps: S_C decrypts the query to get $\mathsf{id}_C, \mathsf{otk}_C, \mathsf{ts}, \mathsf{resp}, \mathsf{ebio}', k_T, \mathsf{flag}_U, \mathsf{flag}_R$. S_C derives rid_C from id_C. S_C retrieves the record $[\mathsf{eerec}_C, \mathsf{last}, \mathsf{ts}_{\mathsf{old}}, \mathsf{flags}]$ from database with key rid_C. S_C cleans up last from outdated information. If flag_U is raised, S_C verifies the rate limit in the record using last (abort if not). S_C checks that flags allows flag_U and flag_R (abort if not). S_C decrypts eerec_C to get erec_C. S_C decrypts erec_C to get rec_C thus $\mathsf{ak}_C, \mathsf{eid}_U, \mathsf{com}_A, \sigma_A$. S_C verifies $(\mathsf{ts}, \mathsf{resp})$ using ak_C (abort if not). S_C verifies that $\mathsf{ts} - \mathsf{ts}_{\mathsf{old}}$ is positive and small, following HOTP [19] (abort if not). S_C updates $\mathsf{ts}_{\mathsf{old}}$ in the database. If flag_U is raised, S_C passes eid_U and ebio' to S_U.
5. (When flag_U is raised only) S_U performs the following steps: S_U decrypts eid_U to get p_U. S_U derives rid_U and otk_U from p_U. S_U retrieves the record from database with key rid_U to get ebio_U. S_U decrypts ebio_U using otk_U to get bio_U. S_U decrypts ebio' to get bio'. S_U runs a matching algorithm between bio_U and bio'. S_U returns the result to S_C. S_U closes the session and discards temporary values.
6. S_C performs the following steps: If flag_U is raised, S_C logs result in the database for rate limiting by updating last in the database. If flag_U is raised and the result is negative, S_C returns $\mathsf{AEEnc}_{k_T}(\mathsf{result})$. Otherwise, If flag_R is raised, S_C picks r_S, prepares σ_S (see the token-based access to R below), and returns $\mathsf{AEEnc}_{k_T}(\mathsf{result}, \sigma_A, \sigma_S, r_S, \mathsf{slot})$. Otherwise, S_C returns $\mathsf{AEEnc}_{k_T}(\mathsf{result}, \sigma_A)$. S_C closes the session and discards temporary values.
7. T performs the following steps: T decrypts the response using k_T and gets result, and $\sigma_A, \sigma_S, r_S, \mathsf{slot}$ if positive. If flag_U is raised and result is negative, abort. If flag_R is raised, T prepares tok using σ_A, r_A, σ_S, and r_S (see below). For owner recognition (flag_U raised): T verifies $(\mathsf{tok}, \mathsf{sn}, \mathsf{slot})$ as explained

below (if flag_R is low, only σ_A is verified), that sn is the serial number of D, and that vk_A is the key of the right authority A. For access to R (flag_R raised): T sends $(\mathsf{tok}, \mathsf{sn}, \mathsf{slot}, \mathsf{flag}_U)$ to R. T discards temporary values.

8. (When flag_R is raised only) R verifies and provides access to the sn record depending on the set up policy, unless tok is invalid or slot has expired (see below).

Then, T can get from R the private attributes of D and services during the time slot.

Token-based access to *R*. We describe here how A computed σ_A and r_A (during chip registration), how S_C computes σ_S and r_S, how T computes tok, and how R verifies the access authorization for T.

This is a simple (partially) unlinkable proof system.

We assume there are public random generators G_1, G_2, G_3 in a group where the decisional Diffie-Hellman problem is hard [10]. (Typically: a random elliptic curve.) We assume a trusted setup. No malicious participant should know the discrete logarithm of any G_i.

At enrollment, A sets a commitment to sn and $(\mathsf{id}_C, \mathsf{otk}_C)$ with a random r_A and signs it:

$$\mathsf{com}_A = \mathsf{sn}.G_1 + H(\mathsf{id}_C, \mathsf{otk}_C).G_2 + r_A.G_3 \quad , \quad \sigma_A = \mathsf{Sign}_{\mathsf{sk}_A}(\mathsf{com}_A)$$

We use here a Pedersen commitment [21].

S_C sets a commitment to $(\mathsf{id}_C, \mathsf{otk}_C)$ with a random r_S, signs it (with slot, com_A, and flag_U), and gives (σ_S, r_S) to T:

$$\mathsf{com}_S = H(\mathsf{id}_C, \mathsf{otk}_C).G_2 + r_S.G_3 \quad , \quad \sigma_S = \mathsf{Sign}_{\mathsf{sk}_S}(\mathsf{slot}, \mathsf{com}_A, \mathsf{com}_S, \mathsf{flag}_U)$$

The proof token is defined by providing the commitments, the signatures, and a zero-knowledge proof of knowledge of the openings and that $h = H(\mathsf{id}_C, \mathsf{otk}_C)$ is the same in both commitments:

$$\begin{aligned}
\mathsf{com}_A &= \mathsf{sn}.G_1 + H(\mathsf{id}_C, \mathsf{otk}_C).G_2 + r_A.G_3 \\
\mathsf{com}_S &= H(\mathsf{id}_C, \mathsf{otk}_C).G_2 + r_S.G_3 \\
\mathsf{tok} &= (\mathsf{com}_A, \sigma_A, \mathsf{com}_S, \sigma_S, \pi) \\
\pi &= \mathsf{PoK}\left\{ (r_A - r_S, h, r_S);\ \begin{array}{r} \mathsf{com}_A - \mathsf{sn}.G_1 - \mathsf{com}_S = (r_A - r_S).G_3 \\ \mathsf{com}_S = h.G_2 + r_S.G_3 \end{array} \right\}
\end{aligned}$$

To build π, set $y_1 = \mathsf{com}_A - \mathsf{sn}.G_1 - \mathsf{com}_S$ and $y_2 = \mathsf{com}_S$, pick k_1, k_2, k_3 at random, compute $C_1 = k_1.G_3$ and $C_2 = k_2.G_2 + k_3.G_3$, compute $e = H(G_1, G_2, G_3, y_1, y_2, C_1, C_2)$, compute $z_1 = e.(r_A - r_S) + k_1$, $z_2 = e.H(\mathsf{id}_C, \mathsf{otk}_C) + k_2$, $z_3 = e.r_S + k_3$, and set $\pi = (e, z_1, z_2, z_3)$. This is a generalized Schnorr proof [23,24] with Fiat-Shamir extension [14] for non-interaction.

T and R can verify σ_A, σ_S, π.

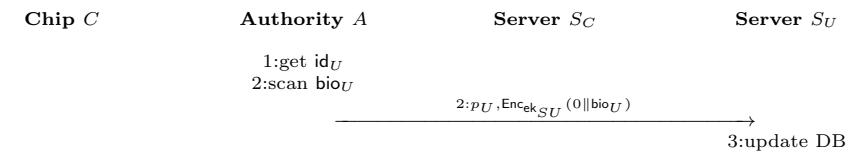

Fig. 4. Biometric Registration Protocol

The verification of σ_A and σ_S are usual signature verifications with the public keys vk_A and vk_{SC}.

$$\mathsf{Verify}_{\mathsf{vk}_A}(\sigma_A, \mathsf{com}_A) \quad , \quad \mathsf{Verify}_{\mathsf{vk}_{SC}}(\sigma_S, (\mathsf{slot}, \mathsf{com}_A, \mathsf{com}_S, \mathsf{flag}_U))$$

To verify π, set $y_1 = \mathsf{com}_A - \mathsf{sn}.G_1 - \mathsf{com}_S$ and $y_2 = \mathsf{com}_S$, compute $C_1 = z_1.G_3 - e.y_1$ and $C_2 = z_2.G_2 + z_3.G_3 - e.y_2$, and check

$$e = H(G_1, G_2, G_3, y_1, y_2, C_1, C_2)$$

4.3 The Registration Protocols

We assume that the issuer I of an existing document D already defined the content on R in the RUDI architecture. By extending the content in R and services, R must determine which authority A can grant access to it. Hence, the identifier of A must also be stored on R in the entry for D. The authority A is assumed to have the authority to define the owner of D or to have means to verify that U is the owner of D. Hence, the registration protocol is only between A, U, C, S_C, and S_U.

Registration of U and C can be done separately. By registering, a user U must choose a private identifier id_U. The user can register several times and get several identifiers. This identifier is only needed to register a new chip C. It can be forgotten after the last C is registered. Actually, registering U and one or several C can be done at the same time, in which case id_U can be set to random and forgotten immediately. This can be done by A without any effort from U.

For the selection of id_U, we require uniqueness in the S_U system. It is further recommended to have a high entropy to prevent random guessing in the system. The impact of guessing id_U is that a malicious S_U would be able to see bio_U without any access request from U.

Biometric registration. At a high level, the biometric registration protocol works as follows (see Figure 4).

1. U selects id_U and tells A.

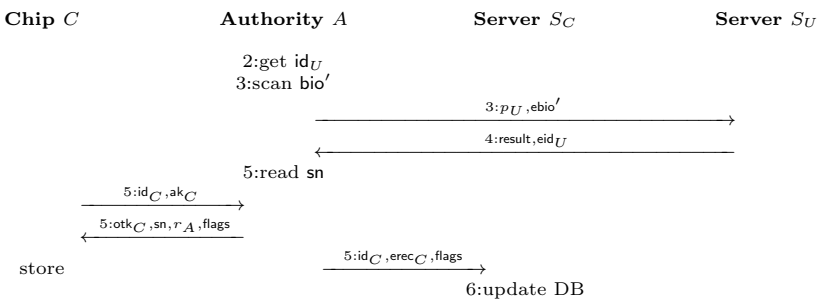

Fig. 5. Chip Registration Protocol

2. A performs the following steps: A derives $p_U = H(\mathsf{id}_U)$. A checks with S_U that p_U is available (i.e. not already used; abort otherwise). A scans a biometric model bio_U of U and encrypt it (this is done by the trusted scanner). A sends (in a private manner) to S_U the values of p_U and $\mathsf{Enc}_{\mathsf{ek}_{SU}}(0\|\mathsf{bio}_U)$. A closes the session and discards temporary values.
3. S_U performs the following steps. S_U verifies the request comes from an authorized A. S_U derives rid_U and otk_U from p_U. S_U decrypts to get bio_U. S_U encrypts bio_U into ebio_U using otk_U. S_U updates the user database with a new entry

$$[\ \mathsf{rid}_U\ ,\ \mathsf{ebio}_U\ ]$$

S_U closes the session and discards temporary values.

Chip registration. At a high level, the chip registration protocol works as follows (see Figure 5).

1. A reads D and decides whether U should be the owner of D (abort if not).
2. U gives id_U to A.
3. A performs the following steps: A derives $p_U = H(\mathsf{id}_U)$. A scans a biometric probe of U and encrypts it to ebio' (this is done by the trusted scanner). A sends (in a private manner) to S_U the values of p_U and ebio'.
4. S_U performs the following steps S_U verifies the request comes from an authorized A. S_U decrypts ebio' to get bio'. S_U derives otk_U and rid_U from p_U. S_U gets ebio_U from the database with entry rid_U. S_U decrypts ebio_U with otk_U to get the model bio_U. S_U does the matching of bio_U with bio'. If it matches, S_U sets $\mathsf{eid}_U = \mathsf{AEEnc}_{k_{SU}}(p_U)$. S_U tells (privately) A the result of the matching and eid_U. S_U closes the session and discards temporary values.
5. If it did not match, abort. Otherwise, A performs the following steps: A reads sn from D. A reads id_C from C as defined by the manufacturer. A sets up a random ak_C (alternately, A reads it from C). A picks r_A and sets the value of com_A and σ_A based on sn, id_C, otk_C. A decides on authorized flags. A configures C to store otk_C, ak_C, sn, r_A, and flags, and locks the read

access to ak_C on C. A sets $\mathsf{rec}_C = [\mathsf{ak}_C, \mathsf{eid}_U, \mathsf{com}_A, \sigma_A]$. A sets up a random otk_C. A encrypts rec_C using otk_C to get $\mathsf{erec}_C = \mathsf{OTEnc}_{\mathsf{otk}_C}(\mathsf{rec}_C)$. A sends to S_C (in a private manner) id_C, erec_C, and flags. A closes the session and discards temporary values.

6. S_C performs the following steps: S_C verifies the request comes from an authorized A. S_C derives rid_C from id_C. S_C sets last and $\mathsf{ts}_{\mathsf{old}}$ to empty. S_C encrypts erec_C using k'_{SC} into eerec_C. S_C updates the chip database with a new entry

$$[\ \mathsf{rid}_C\ ,\ \mathsf{eerec}_C\ ,\ \mathsf{last}\ ,\ \mathsf{ts}_{\mathsf{old}}\ ,\ \mathsf{flags}\ ]$$

S_C closes the session and discards temporary values.

Table 2. Implementation round-trip time (100 iterations)

protocol	operation	average	standard deviation
biometric registration	Steps 2–3 without biometric scan	968.39ms	93.26ms
chip registration	Steps 3–6 without biometric/chip scan	1183.47ms	25.11ms
authentication	Steps 3–8 without biometric scan	1155.48ms	26.15ms

5 Implementation

We detail our implementation here.

The one-time encryption OTEnc is AES in a CTR mode using a zero-nonce [5]. The AEEnc is an authenticated encryption: it is AES-GCM-SIV with a zero-nonce [15]. The hash function H is SHA256 [3]. The function PRF is HKDF [16]. The public key cryptosystem is ECIES [25] or HPKE [9]. The signature scheme is ECDSA on P256 [4].

The user U should select id_U of large entropy to avoid offline password search after corruption of S_U. Actually, id_U is a private identifier which is needed only to register U and to register C immediately after. If lost, U could register again. Using SHA256, p_U is of 32B.

The size of σ_A is of 64 bytes, but C is using S_C as a repository to avoid storing it on the chip. The size of r_A is normally of 32 bytes, but is reduced down to 16B using a pseudorandom generator.

The storage of the chip C is $(\mathsf{id}_C, \mathsf{otk}_C, \mathsf{ak}_C, \mathsf{sn}, r_A, \mathsf{flags})$. However, id_C and ak_C may be "free" of storage need as they are already required by the firmware. The key otk_C is of 16B. With sn of 8B and r_A of 16B, the total is of 40B, plus the "free" keys: with id_C of 8B and ak_C of 16B, this is 64B in total. (We did not count the 4 bits of flags.)

With the same computation, the database in S_C occupies 221B per record. The database of S_U is larger. The size of the records is dominated by the biometric information for which we allocate 30KB.

We implemented our protocol. We used the EM 4425 RFID tag[1] and the F1 Venoscanner[2]. The terminals and servers were implemented on Intel Core Ultra 7 155H with 32GB RAM, running Ubuntu 24.04.2 LTS. The terminals and servers were implemented in dockers. When running 100 times, we observe round-trip times around the second, as detailed in Table 2.

We also implemented our terminal/authority on Android to have a portable version, and the app-less version which is presented in Appendix D.1.

6 Discussions

Analysis, possible attacks, protections, and extensions. The discussions about the security analysis, some possible attacks, some possible protections, and some direct extensions of our protocol have been moved to the appendix of this paper.

Scalability. Scalability is not an issue in our protocol. Only the servers are impacted by the scale since their database grows linearly with the number of registered documents and only a database query will be affected by this size.

Revocation. To revoke a document D, we can just update its entry in the repository R. To revoke a chip C, we can remove its entry in S_C if we have id_C or rid_C. We can also smash C with a hammer or use the kill options by the manufacturer. Finally, to revoke the owner U of a chip C, we can recover eid_U by using C and remove the entry in S_U.

Post-quantum extension. To reach post-quantum security in the user ownership functionality, we only need to replace the cryptosystem by a post-quantum secure one such as ML-KEM [6]. To so the same in the access control functionality, we must replace the signature scheme, the commitment scheme, and the proof-of-knowledge by post-quantum secure ones. We can suggest ML-DSA [7] for signature. For commitment or zero-knowledge proofs, we can use the results by Lyubashevsky, Nguyen, and Plançon [17] and the LaZer library [18].

Privacy. Our design provides private information on a need-to-know basis. The server S_U is running the matching algorithm but does not store biometric data in clear and cannot decrypt without interacting with C. Given that it starts with NFC, we believe this implies the holder's consent. In addition to this, S_U does not know C, U, nor D.

7 Conclusion

We developed an architecture which allows to recognize if a person is the registered owner of a registered document. For this, we use biometry and RFID

[1] `https://www.emmicroelectronic.com/product/nfc-high-frequency-ics/em-echo-v`

[2] `https://globalid.swiss`

technologies. Our architecture is optimized to be efficient, secure, and privacy-preserving. Our protocols use simple stateless query-response client-server connections. The storage in the RFID chip is minimal. We build on existing "RUDI" infrastructure. Finally, our approach enables new services with access control policies.

Acknowledgements. We thank Piotr Kulpinski, Leonhard Koblitz, Zhicheng Dong, Charles Rousset and Alejandro Romero for their great contributions to the implementation of our system and for their comments. This project was supported by GlobalID.

References

1. Mobile Driver's License Functional Needs White Paper. Version 0.9. AAMVA. 2019. https://www.aamva.org/topics/mobile-driver-license
2. Verifiable Credentials Overview. W3C Group Note 09 January 2025. World Wide Web Consortium. https://www.w3.org/TR/vc-overview/
3. Secure Hash Standard (SHS). *Federal Information Processing Standards* Publication #180-4. U.S. Department of Commerce, National Institute of Standards and Technology, 2015.
4. Digital Signature Standard (DSS). *Federal Information Processing Standards* Publication #186-5. U.S. Department of Commerce, National Institute of Standards and Technology, 2023.
5. Advanced Encryption Standard (AES). *Federal Information Processing Standards* Publication #197. U.S. Department of Commerce, National Institute of Standards and Technology, 2001.
6. Module-Lattice-Based Key-Encapsulation Mechanism Standard. *Federal Information Processing Standards* publication #203. U.S. Department of Commerce, National Institute of Standards and Technology, 2024. https://doi.org/10.6028/NIST.FIPS.203
7. Module-Lattice-Based Digital Signature Standard. *Federal Information Processing Standards* publication #204. U.S. Department of Commerce, National Institute of Standards and Technology, 2024. https://doi.org/10.6028/NIST.FIPS.204
8. Fatih Balli, F. Betül Durak, Serge Vaudenay. BioID: a Privacy-Friendly Identity Document. In *Workshop on Security and Trust Management STM'19*, Luxembourg, Lecture Notes in Computer Science 11738, pp. 53–70, Springer-Verlag, 2019. https://doi.org/10.1007/978-3-030-31511-5_4
   Eprint 2019/894. https://eprint.iacr.org/2019/894
9. Richard L. Barnes, Karthik Bhargavan, Benjamin Lipp, Christopher A. Wood. Hybrid Public Key Encryption. RFC 9180. IETF. 2022.
10. Dan Boneh. The Decision Diffie-Hellman problem. In *Algorithmic Number Theory ANTS'98*, Portland OR, USA, Lecture Notes in Computer Science 1423, pp. 48–63, Springer-Verlag, 1998. https://doi.org/10.1007/BFb0054851
11. Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, vol. 13 (7), pp. 422–426, 1970. https://doi.org/10.1145/362686.362692

12. F. Betül Durak, Loïs Huguenin-Dumittan, Serge Vaudenay. BioLocker: A Practical Biometric Authentication Mechanism Based on 3D Fingervein. In *Applied Cryptography and Network Security ACNS'20*, Rome, Italy, Lecture Notes in Computer Science 12147, pp. 62–80, Springer-Verlag, 2020. https://doi.org/10.1007/978-3-030-57878-7_4
Eprint 2020/453. https://eprint.iacr.org/2020/453
13. Keng-Yu Chen, Serge Vaudenay. The Cryptographic Layer of Biometric Authentication. To appear in Applied Cryptography and Network Security ACNS'26, LNCS, Springer-Verlag.
14. Amos Fiat, Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology CRYPTO'86*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 263, pp. 186–194, Springer-Verlag, 1987.
15. Shay Gueron, Adam Langley, Yehuda Lindell. AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption. RFC 8452. IETF. 2019.
16. Hugo Krawczyk, Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869. IETF. 2010.
17. Vadim Lyubashevsky, Ngoc Khanh Nguyen, Maxime Plançon. Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General. In *Advances in Cryptology CRYPTO'22*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 13507–13510, pp. 71–101 vol.2, Springer-Verlag, 2022.
Eprint 2022/284. https://eprint.iacr.org/2022/284
18. Vadim Lyubashevsky, Gregor Seiler, Patrick Steuer. The LaZer Library: Lattice-Based Zero Knowledge and Succinct Proofs for Quantum-Safe Privacy. In *ACM SIGSAC Conference on Computer and Communications Security CCS'24*, Salt Lake City UT, USA, pp. 3125–3137, ACM, 2024. https://doi.org/10.1145/3658644.3690330
Eprint 2024/1846. https://eprint.iacr.org/2024/1846
19. David M'Raihi, Mihir Bellare, Frank Hoornaert, David Naccache, Ohad Ranen. HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226. IETF. 2005.
20. David M'Raihi, Salah Machani, Mingliang Pei, Johan Rydell. TOTP: Time-Based One-Time Password Algorithm. RFC 6238. IETF. 2011.
21. Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology CRYPTO'91*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 576, pp. 129–140, Springer-Verlag, 1992.
22. Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. IETF. 2018.
23. Claus P. Schnorr. Efficient Identification and Signature for Smart Cards. In *Advances in Cryptology CRYPTO'89*, Santa Barbara, California, U.S.A., Lecture Notes in Computer Science 435, pp. 235–251, Springer-Verlag, 1990.
24. Claus P. Schnorr. Efficient Identification and Signature for Smart Cards. *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.
25. Victor Shoup. ISO/IEC 18033-2: Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric Ciphers. ISO/IEC. 2004.
26. Ewa Syta, Michael J. Fischer, David Wolinsky, Abraham Silberschatz, Gina Gallegos-García, Bryan Ford. Private Eyes: Secure Remote Biometric Authentication. In *Security and Cryptography SECRYPT'15 and E-Business and Telecommunications ICETE'15*, Colmar, France, vol. 4, pp. 243–250, SciTePress, 2015. https://doi.org/10.5220/0005539602430250

A Analysis

A.1 Security Results

We study here the security properties of our system, as announced in Section 2.2.

Since A has the authority to decide who is the owner of a document, we assume that A is honest in security analysis. Assuming that the commitment is binding, that the signature is unforgeable, and that π is sound, whenever the authentication protocol returns a valid $(\mathsf{tok}, \mathsf{sn}, \mathsf{slot})$, this implies a valid signature σ_A on a commitment to sn and some id_C, so that A honestly registered a document with serial number sn. This honest registration was done with a document, a chip, and an owner which we denote by D, C, U.

As soon as C has been scanned by an adversary, a malicious S_C has all means to claim that a holder U' of D was recognized as the owner of D (as seen in Appendix B). Hence, we assume that S_C is honest in security analysis.

Similarly, S_U has freedom on returning result to S_C (as seen in Appendix B) we assume that S_U is honest in security analysis.

This is why we assume that A, S_C, and S_U are honest in the security analysis.

Owner recognition. We consider a terminal T who is shown D' with a chip C' and a holder U', and that after running the protocol, the output is a match for sn, the serial number of D registered with C. As owner recognition is providing results to T only, we only care for the honest T case. If $D' \neq D$ but D' has the serial number sn, it means that D' is a forgery (with $C' = C$ or not). It is not the purpose of ownership recognition to detect it. This could rather be done with R services, by comparing D' with the online copy of D. Hence, we assume that A, S_C, S_U, and T are honest and that $D' = D$.

Due to the soundness of π, com_A and com_S commit to the same $H(\mathsf{id}_C, \mathsf{otk}_C)$. Assuming collision resistance for H, this indicates that they commit to the same id_C. Assuming the unforgeability of the signature, this means that id_C used during registration and owner recognition are the same, hence use the same ak_C and eid_U, following rec_C in the database. Furthermore, com_A binds to sn. Due to honest registration, the registration of id_C was precisely for the document of serial number sn, which is D and has owner U.

Since T is honest, U' is compared to U. Assuming that biometry is reliable, the result shows that $U = U'$. Hence, U' is really the owner of D.

Consent and access control. We now consider a repository R who receives an access token from a terminal T for a document sn during a time slot slot, with signatures from A and from S_C.

Due to the unforgeability of the signature and the honesty of S_C, it must be the case that σ_S was signed at the beginning of slot. Hence, the token is recent.

The same argument as above shows that S_C signed after having received from T a valid authenticator corresponding to C and some biometric scan of U. We can say a bit more: that neither the authenticator nor any more recent

authenticator by C was presented to S_C before, and, unless C has been corrupted or that HOTP is insecure, the authenticator was collected by scanning C. If we use an extra protection (which is presented in Appendix D.2), we can assess that the biometric scan is also recent so it looks legitimate to grant access to T.

A.2 Privacy Results

We assume a honest U who was registered by a honest A as owning a document D which is attached to a non-corrupted chip C.

Access to biometric templates As long as scanners and S_U are honest, they are the only participants who can see biometric templates in clear, and this happens only when they need it. Either because they capture it (for scanners), or because they need to run the matching algorithm (for S_U). This is due to the security of encryption with ek_{SU}.

We have even more guaranties when corruption happens, as detailed below.

Impact of leakages in databases. We assume that A, S_C, and S_U are honest but that the databases leaked.

On the biometric database, records are totally indistinguishable from random so leak no information, due to the security of encryption.

On the chip database, rid_C and eerec_C are indistinguishable from random, due to the security of PRF and or encryption. Hence, only last and $\mathsf{ts}_{\mathsf{old}}$ leak information. Potentially, an adversary can identify which database entry corresponds to a given chip C, and just learn the result of the recent (in the last few minutes before the leakage of the database) access control requests. We do not take it as a privacy concern.

In this model, the risk is that the adversary manages to harvest a tuple from C. They could then decrypt the databases entries and recover the biometric templates. Hence, in the case of a database leakage, the chips should be destroyed.

Corrupted servers. We assume that A honestly registered a document D with a chip C and an owner U. Clearly, S_C and S_U receive no information about D (not even sn).

We now assume that S_C and S_U honestly register C and U. However, they later became malicious.

If the entropy of id_U is low, S_U can guess it and derive rid_U and otk_U to be able to obtain bio_U, which is a privacy concern.

We assume that id_U and id_C have a high entropy and that S_C and S_U receive no external information (such as scans of C). In that case, the database entries are still indistinguishable from random, except last and $\mathsf{ts}_{\mathsf{old}}$ which leak little information.

B Possible Attacks

Corrupted *R*. A corrupted repository sees the content of the digital copy of the document D but has no link to bio_U. The problem of access by R to the documents already exists in the RUDI architecture, when we do not use our CATS system. We could fix it (as discussed in the Appendix D.3) but we believe that this extension is out of the scope of our system.

Leakage from *C*. The chip C is talkative and can "answer to strangers". When activated, C yields a tuple $(\mathsf{id}_C, \mathsf{otk}_C, \mathsf{sn}, r_A, \mathsf{ts}, \mathsf{resp})$. While id_C, otk_C, sn, and r_A are static, ts and resp change all the time.

Alone, this enables an adversary to track the presence of C, identified by id_C. This also leaks that sn is an existing serial number. To protect against this, we could make sure that C only activates through NFC range to reduce the easiness to detect C from far away.

The tuple can also be used by the adversary to impersonate C to S_C, only once, and until C authenticates for real to S_C as the authenticator $(\mathsf{ts}, \mathsf{resp})$ from the tuple would then be rejected. However, the adversary still needs a biometric scan of the owner to get full access to R (i.e. access with flag_U raised). If an adversary gets a recent tuple from C and the biometric scan of the owner, our protocol can no longer identify illegitimate access request.

Corrupted *C*. The problem is more severe if the tamper-resistance of C is broken and ak_C leaks. In that case, C can be cloned or impersonated at any time to create fresh $(\mathsf{ts}, \mathsf{resp})$ authenticators. However, the adversary still needs a biometric scan of U to have full access to R.

Corrupted *T*. A corrupted T will see the content of D, will get a leaked tuple from C, will get ebio' from the scanner scanning U, and will get $(\sigma_A, \sigma_S, r_S)$ from S_C.

Getting σ_A, r_A, σ_S, and r_S together allows to get access to R during a specific time slot, but this is actually the purpose of our system. To get access for another time would require to obtain a new (σ_S, r_S) from S_C which would assume getting a new $(\mathsf{ts}, \mathsf{resp})$ pair from C. What could be a problem is if the adversary steals C and gets more access to R without the consent of the owner, by replaying ebio'. It is debatable whether getting access to R by using C and an old ebio' is a feature or a threat, depending on the application. If this is a concern, S_U should implement protection against replay attacks of ebio' (as discussed in the Appendix D.2) and enforce ebio' freshness.

Corrupted PKI. The PKI is mostly responsible for distributing ek_{SC}, vk_{SC}, vk_A to terminals T. Other public keys (vk_A and vk_{SC}) are distributed to repositories R which are corporate actors and may have stronger means to get public keys.

A corrupted PKI may distribute a rogue ek_{SC} to a target T. As an effect, the adversary can decrypt what is sent by T to S_C, i.e. id_C, otk_C, ts, resp, and ebio', k_T. This situation is very similar to the corrupted T case, except that the adversary does not get r_A so cannot forge an access token. Getting ebio' is a more severe problem as it may be reused if the adversary can get a tuple from C and play the role of a corrupted T.

A corrupted PKI may distribute rogue vk_{SC} or vk_A to T. Without corruption of A or S_C, and without breaking the security of the $A \leftrightarrow S_C$ or $T \leftrightarrow S_C$ links, giving wrong keys is useless as S_C would give the correct σ_A and σ_S anyway.

Corrupted S_C. A malicious chip server S_C can grant access to R for a document of serial number sn to a user U' who has got the right chip C of the corresponding document D, even though U' is not the owner U of D. For this, S_C just follows the protocol without interacting with S_U and does as if S_U accepted. For this attack, access to C is needed.

The only missing information for S_C to run the attack without C is

$$(\mathsf{id}_C, \mathsf{otk}_C, \mathsf{sn}, r_A)$$

If a corrupted S_C managed to get (to harvest) this from C, they can prepare (σ_S, r_S) for whatever slot, then the token tok as T would do, and get access to R. Hence, harvesting tuples from chips is a concern when used by a corrupted S_C.

A corrupted S_C cannot decrypt the record of C in the database without having received otk_C from C.

Collusion of S_C and R. With this scheme, a collusion between S_C and R can associate sn and $(\mathsf{id}_C, \mathsf{otk}_C)$, which is a privacy breach. Avoiding this would be expensive: no matter how the proof token is made, watching the traffic to R during the allowed time slot reveals potential associations. If there is a single query during this slot, the association is clear. To avoid this we would need long time slots and delayed requests to R, which would harm usability.

Corrupted S_U. Similarly, a corrupted S_U can pretend that a malicious U' is the owner U of a document D attached to a chip if C is available for the attack.

A corrupted S_U can also map every eid_U to bio_U and to the submitted bio'. However, S_U does not know which document or chip is related to eid_U, and the only information about the owner U is given as a hash p_U of a one-time pseudonym id_U. Hence, it is recommended that U should not use a pseudonym which reveals an identity.

A malicious S_U could also try to guess some id_U to be able to decrypt some random bio_U in the database. However, we feel that this attack would recover a limited number of bio_U which will not be much more than what S_U is legitimately decrypting. If this is a concern, the entropy of id_U must be high enough to prevent that. (We recall that id_U may be needed only once immediately after registration. So, having a high-entropy identity is making a lot of sense.)

Corrupted A. The authority A has the authority to bind a chip C to a document sn. The binding is sealed in the chip database with com_A, so when S_C issues their authorization, it cannot be used for another sn. However, A could register a dummy document D' with a simulated chip C' and any person U', and seal the serial number sn of a target document D so that A would get access to the service about D on R.

R should verify from the structure of sn if σ_A is signed by the A who has authority to register this serial number. This would prevent A from getting access to documents registered by another A.

If getting service access to documents that A registered is a concern, our system should be extended as discussed in Appendix D.3.

Leaking S_C. We assume protection against leakage of the secret keys (typically, using a trusted hardware). What can leak from the storage of S_C is the database. When an adversary sees id_C, otk_C, ts, resp from C, the only link to the corresponding record $(\mathsf{rid}_C, \mathsf{eerec}_C, \mathsf{last}, \mathsf{ts}_{\mathsf{old}})$ is through last and $\mathsf{ts}_{\mathsf{old}}$. However, recovering the record of C only provides information which cannot be decrypted.

Leaking S_U. We assume protection against leakage of the secret keys (typically, using a trusted hardware). What can leak from the storage of S_C is the database. However, no otk_U can be recovered without k'_{SU} which is well protected. So no ebio_U can be decrypted. Furthermore, there is no way to link some seen eid_U, id_U, or ebio_U. However, the adversary can link eid_U with ebio' by impersonating S_C to S_U and link p_U to ebio' by impersonating A to S_U.

Curious S_C. A curious S_C who suspects that two records rid_C and $\mathsf{rid}_{C'}$ have the same owner U, although maybe registered twice in S_U, can try to verify this guess. As soon as S_C obtains what is needed to decrypt the record $\mathsf{rid}_{C'}$ (by scanning C' or by having illegally kept a previous requests for it), S_C can obtain $\mathsf{eid}_{U'}$. If ebio' was kept from a previous request for C, S_C can query S_U with $(\mathsf{eid}_{U'}, \mathsf{ebio}')$. The results shows if U and U' are the same person or not.

Curious S_U. A curious S_U who suspects that two records rid_U and $\mathsf{rid}_{U'}$ come from the same person can try to verify this guess. As soon as S_U obtains eid_U and $\mathsf{eid}_{U'}$ (from access request or from registration), S_U gets what is needed to decrypt the records rid_U and $\mathsf{rid}_{U'}$, S_U can obtain bio_U and $\mathsf{bio}_{U'}$, and match them. The results shows if U and U' are the same person or not.

C Protections

Anti-cloning. The chip C stores a secret ak_C which never goes out. It is only visible by S_C. So, unless S_C is corrupted (either otk_C was harvested or corruption was in place when S_C was accessing ak_C so that the corrupted S_C can decrypt the record of C), and assuming that C is tamper-resistant, no cloning of C is possible. As C authenticates to S_C based on ak_C, the chip cannot be impersonated.

Anti-hill climbing. A malicious T having got the authenticator of a chip can use it to interact with S_C during a time period. The malicious T can try many synthetic bio'_U until one is accepted, then apply small variations on this accepted template to reconstruct every bit of bio_U. We protect against this attack by blocking access temporarily after too many negative answers, and blocking temporarily access after any positive answer. This considerably slows down the attack to make it impractical.

Note that hill-climbing can be improved by using several chip C using the same owner U with same id_U. So, in general, it is not a good idea to reuse id_U.

Hill-climbing could also be done by a corrupted A. For this, specific audit tools on S_U should be implemented, to raise an alert or temporarily block A after too many non-matching queries.

D Possible Extensions

D.1 Simple Access Control to R

There are situations where we need neither ownership recognition nor access control. We only want to get access to R with limited functionality. For instance, to check the revocation status of D on R or just a digital copy to be sure that D was not forged.

App-less variant (no biometry, no chip authentication). In this situation, we can extend our system by having C to use URL substitution. Essentially, just scanning the RFID tag with a regular smartphone would provide the URL of R. The URL would contain additional references such as sn. As the chip C becomes passive, it could even be replaced by a QR code in that case. We need no dedicated app on T for that.

D.2 Adding Security on the Probe Freshness

At the moment, anyone who got some encrypted biometric template, may it be A or T after scanning, or even S_C who received an access request, can try to maliciously replay it. By scanning C, this can be used to get an access to R.

There are several directions to protect against this. First of all, we could implement a protection against replay attacks by keeping record of previous queries. For instance, S_U could insert ebio_U in a Bloom filter [11] to be stored in the record of rid_U.

This protection is not enough to make sure that a probe is recent. For this, we could modify the scanners so that they encrypt a timestamp together with the biometric template. One problem is that scanners may not have any notion of time and would rely on external clocks. They could at least check that the timestamp they are given increased. They could also interact with a trusted clock service as proposed in BioID [8].

Another option is to add a communication path from S_U to the scanner, so that S_U can send a nonce and make the scanner encrypt it with the biometric template.

D.3 Protecting against a Curious Repository

So far, R can see whatever R is supposed to provide to clients. A malicious A can also get access to a honestly registered sn by registering it again maliciously so that they can get an authorization from S_C at any time.

Our system can be extended by storing an additional unlock key in either C or U (as a password in that case). This would prevent a malicious A from access. Additionally, we can use a decryption key inside C to read the stored content in R. This would prevent a curious R from access.