FICP: Federated Identity Card Provider
An Identity Provider to secure identity ownership.

Micael Pedrosa[1]
Universidade de Aveiro, Aveiro, 3810-193, Portugal

***Abstract*— FICP is a protocol proposal for a distributed Identity Provider (IdP) that aims to reduce the risk of identity forgery by the IdP itself. The identity is forced into a digital signature (the FI-Card) close to the object of identification, assuming that only the identity owner is a trustful identity manager. This makes it impossible to forge the identity, as long the underling protocol doesn't leak vital information to intermediate parties. The FI-Card also introduces new challenges and innovation opportunities related with card cancellation and renovation that will be addressed here.**

I. Introduction

An Identity Provider (IdP) is a system that manages identity information while providing authentication services to relying party applications within a federation or distributed network. Today it's very frequent to have applications from different organizations using a common digital IdP [1]. E.g. Facebook and Google as Identity Provider's, use interoperable protocols that can bring Single Sign-On (SSO) functionalities to external applications like Spotify. This makes identity management (and other associated features, e.g Access Control) much easier for application developers. However, from a corporate point of view, the true owner of the identity is not the user. All the needed requirements for a successful log-in is in the IdP, and so, the IdP is capable of impersonating the user identity if it also controls the SSO protocol. While this is not of major importance in day to day internet applications, it's a considerable concern for corporate applications, regarding corporate espionage [2].

An example of a normal login flow and a forged flow is in Figure 1. If a Spotify user tries to login using their Facebook account, the normal flow a) will redirect the browser agent to the Facebook authentication domain, and again to the application domain on a successful login. In the forged flow b), a successful administrator login will redirect to a forgery form page, where the account to be impersonated can be inserted here. The IdP can also disable all kind of audit events to the original account, hiding the forged activity. Although these events can also be generated in the application side, this is rarely implemented.

To circumvent this lack of trust, when organizations need SSO features normally they use local deploys, yet these don't promote cooperation between organizations. The FICP protocol will increase trust in the IdP and SSO system, so that different organizations can cooperate by using the same identities and at the same time reducing the risk of corporate espionage. The FICP will define data structures and communication protocols that aims to put the identity ownership closer to the end user, by spreading the IdP roles to several different mechanisms and network points, minimizing the identity forgery capabilities by the IdP. The goal is to provide those features maintaining the requirements of anti-theft, anti-fraud and privacy as also avoiding recurrent IdP deploys, and instead provide it as a global service.

II. Background

Novel attempts to remove the strong trust in IdP like SlashID [3], adopts user passwords to encrypt and store critical information in the IdP. Although this facilitates the evolution of the account, by changing passwords in case the account is compromised, people are notoriously poor at achieving sufficient entropy to produce satisfactory passwords. Strong passwords are required to avoid dictionary attacks or other sophisticated methods of password guessing [4] and to resist to the increasing computer power that IdP corporations have access to. Also, the SlashID protocol as defined, demands direct connection between the identity owner and the application service, sending unencrypted information that could compromise the protocol if intercepted by the IdP or any other message broker. The FICP circumvent the use of user generated passwords with asymmetric cryptographic keys, and can transmit messages through any middleware without compromising the authentication process.

Implementations of federated SSO protocols like SAML2, RADIUS and OpenID variants mostly rely on a central IdP to manage identities. An example of a RADIUS architecture is the Eduroam [5] used by universities. Students identities are still owned by the university, but since there is a fair amount of trust between those organizations, there is no need to avoid identity forgery from the IdP. OpenID Connect can be considered a modern version of SAML2, and since it doesn't define the method of authentication, it's even possible to use Smart Card authentication schemes [6]. Although Smart Cards do provide identity ownership in a similar manner as the FICP, there is no seamless solution to recover from a lost or stolen card. The FICP protocol is able to recover using a form of consensus [7] from a trusted quorum, that we call Trusted Links. Other methods of key recovering from biometric data [8] are available, however biometric data can also be stolen and is not revocable, locking the identity to a unique and immutable key. The FICP can use any method to generate keys and evolve to a different key if needed.

The Public Key Infrastructure (PKI) is also similar to a virtual ID card using X.509 certificates that needs a Certificate Authority (CA). This option was abandoned for new

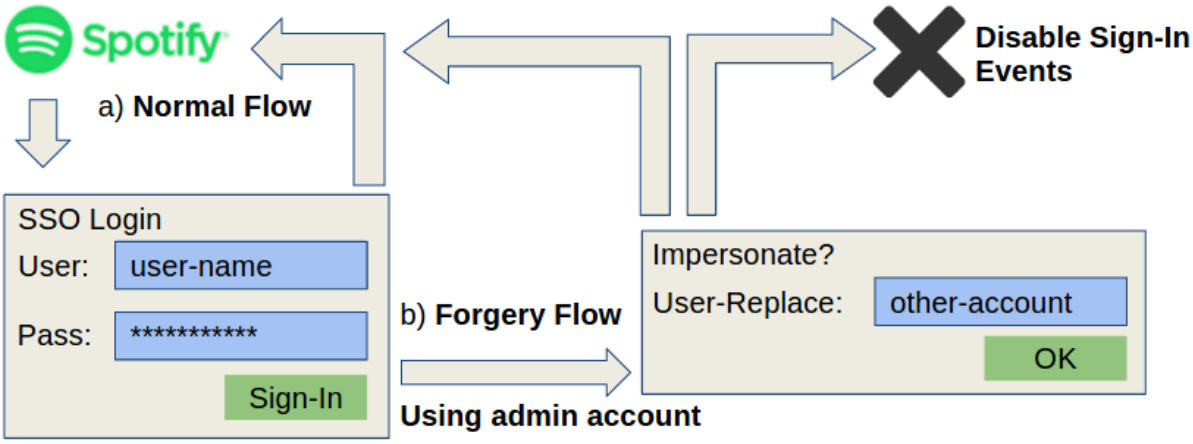

Fig. 1. SSO login flow. Representing the normal login flow a) and the possible identity forgery flow b).

deploys because of the too many moving parts and reported hight costs [9] of maintenance. The FICP avoids a central IdP and a Certificate Authority (CA), giving preference to consensus algorithms, Out-Of-Band provisioning methods [10] and two-factor authentication (2FA).

III. Architecture

In Figure 2 are identified the main components in the FICP protocol. The **FI-Card** is where the identity is protected from external components, this is comprised of a pair of private and public asymmetric keys. These keys can be stored in a cell phone software or even in a smart card. We support elliptic curves [11] to generate the key pairs, although other cryptographic methods can be adequate. The **FI-Gateway** acts as a message broker and replaces the role of the IdP, it has all the required information to validate cryptographic signatures, and it stores the most recent copies of all CardChain structures (explained in Section III-A). The **FI-Application** is the application (e.g. Spotify) requiring the FICP protocol for the authentication process. Finally a set of **FI-Trusted Links** can be used to cancel and recover from a lost or stolen FI-Card, these can be trusted external corporations, friends or any other software also using the FICP protocol.

All components are identified by an UUID corresponding to a CardChain. However a CardChain is only valid when registered in a FI-Gateway making the CardChain fork uniquely identifiable between multiple gateways. The gateway can also require additional information on the CardBlock $info$ in order to comply with internal restrictions of the database, e.g. a unique alias for the identity.

A. The CardChain

The most important requirements (and also where the complexity lies) of the FICP protocol are the Cancel/Recover and Evolve phases. The **CardChain** data structure in Figure 3 is essential for these requirements, and gathers some ideas from the BlockChain technology [12], in the sense that an append only block-chain is produced and linked

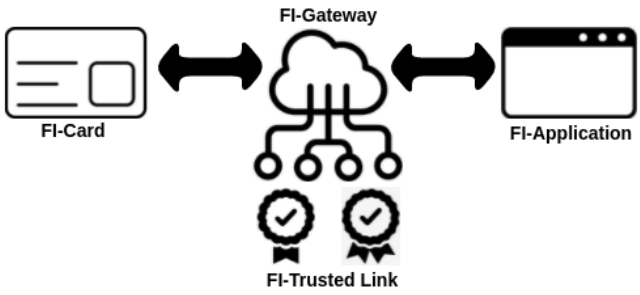

Fig. 2. Main components in the FICP protocol.

with digital signatures. The CardChain is stored in the FI-Gateway and is composed by sequential **CardBlock's** linked with **CRLink's** (Cancel and Recover links). The chain is identified by the UUID, that is the public key from the first generated CardBlock. A CardBlock contains the UUID and the public key for the block, any public information that the identity owner wants to share, plus an header with the cryptographic suite used to sign the block. The trusted links (**T-Link** explained in Section III-C) are required to evolve the chain. The T-Link's are references to identities also with an asymmetric key pair, in this way capable of generating signed Cancel and Recover blocks. When a set of correctly signed and verified Cancel blocks are generated (for each corresponding T-Link) that completes the pre-established requirements (e.g. Cancel blocks requires half of the T-Link's), the cancel status is established and the CardChain evolves to the inactive state. When a CardBlock is canceled the FI-Card can generate and submit new candidates to the FI-Gateway, only then the signed Recover CRLink's can complete the chain to the new candidate and activate the CardChain with the new CardBlock. The whole structure will provide an append only history of the evolved cards. The current account identity is grant by the active CardBlock, the header of the CardChain.

The serialized CardBlock structure when used by a transport protocol is a base-64 encoded string containing:

- 4 bytes indicating the JSON structure $size$.

- $size$ bytes with the JSON structure: {$uuid$: string, key: string, $header$: object, $info$: object, $links$: T-Link[]} for the already mentioned CardBlock structure.
- last remaining bytes with the signature of the JSON structure.

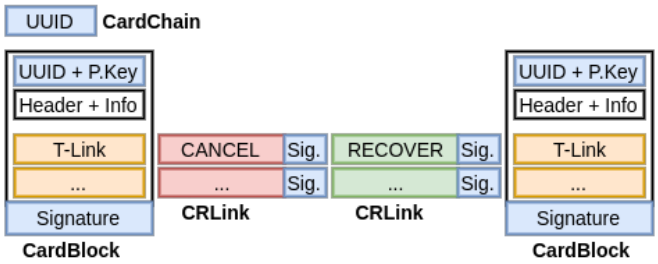

Fig. 3. A CardChain data structure evolving to the next CardBlock.

B. Chain Forks

When a CardBlock private-key is compromised it's possible to create a fork with the same UUID in a different FI-Gateway. Any synchronization protocol between gateways should be aware of this issue, but since this is out of scope of the FICP protocol, all FI-Application's must assume that UUID's of different gateways are independent identities. This effectively implies that an identity will be uniquely identified in a domain with the address <uuid>@<idp-key>. Note that the Evolve phase described in Section IV is not considered a complete synchronization protocol between gateways, but it can be used to construct one.

C. Trusted Links

The CRLink's detailed in Figure 4 are generated by FI-Trusted Link entities. The T-Link structure needs the public key and address of the FI-Trusted Link so that any CRLink signature can be verified in the FI-Gateway. The T-Link's are pre-established when the CardBlock is created, meaning that the identity owner will only trust those links to sign Cancel and Recover CRLink blocks. Any signed CRLink has the full information to correctly position it self in the CardChain, mainly the type of link (Cancel or Recover), the T-Link public key, the previous and next public keys of the corresponding CardBlock's (Next P.Key is only for Recover links). Any self signed Cancel CRLink is enough for the CardBlock cancellation. In that regard the FI-Card is considered an implicit T-Link that can only be used for the Cancel procedure.

The pre-established requirements to accept a set of CR-Link's that completes the Cancel or Recover actions are defined in the CardBlock information. The generation process of the CRLink's is out of scope of the FICP protocol, the only requirement is to obey the Cancel and Recover protocol phases. This abstraction can encompass several use cases like: trusting in a friends list to help the owner recover the card, trusting only in the card emitter (e.g. country governments), set a specific link as mandatory (the CRLink from the entity is mandatory) or authoritative (the CRLink from the entity override other links), discriminate if the T-Link is to be used only in Cancel or Recover procedures, etc. In a sense, it's possible to implement common IdP flows with the correct CRLink requirements, making the FICP a more general approach for an authentication protocol.

The serialized CRLink structure when used by a transport protocol is similar to the CardBlock, where the only difference is in the content of the JSON structure.

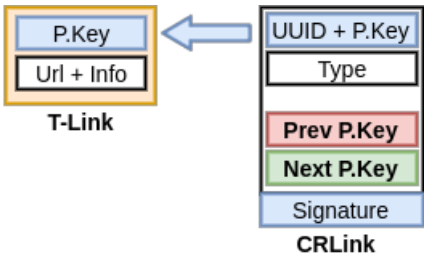

Fig. 4. A Trusted Links structure of the CardBlock and the respective CRLink.

IV. Protocol

This protocol proposal is at the application layer in the OSI model [13], and is abstracted from any transport layer. All payloads are JSON structures and base-64 encoded binaries, that will simplify any transport layer integration in the future. The protocol is composed by several phases: **Challenge, Register, Subscribe, Search, Login, Cancel/Recover and Evolve** interacting with the components in Section III. Components will be identified in short as: FI-Card (**FIC**), FI-Gateway (**G**), FI-Application (**A**), FI-Tusted Link (**TL**) and (**ANY**) for any of those.

Each protocol phase is defined in the next sections, incorporating: **Message Sequence**, **Acknowledge Errors**, **Body Structures** and possible **Attack Vectors**. Every message structure is defined using TypeScript [14] interfaces, having the same header fields and a body JSON object with the structure of: {ver: string, $type$: string, cmd: string, id: number, $from$: string, to: string, $body$: <body structure>}, where ver is the protocol version ("1.0" for this publication), $type$ and cmd is the identifier of the message that is used in Message Sequence definitions (on the next sections), id is the session message number provided by an internal counter of the message emitter (where the same id represents a group of messages corresponding to a session of the same protocol phase), $from$ and to are the CardChain UUID used for the routing addresses (the to field is not needed when the destination is the gateway), and $body$ as defined in the Body Structures. Even when there is no explicit **ack** for a message flow, errors should always be expected and are implicit in the Message Sequence definition.

A set of generic errors are defined here, and can be sent as a response to any request:

Unknown → {$code$ = -1 and $error$ = "Unknown error!"} For all internal server errors or other unknown errors.

Timeout → {$code$: -2, $error$: "Timeout error!"} When a reply exceeds the timeout threshold.

Signature $\rightarrow$ {$code$ = -3 and $error$ = "Signature error!"} When a correct signature is required, and the provided one is incorrect.

No-Chain $\rightarrow$ {$code$ = -4 and $error$ = "Non existent card-chain!"} When there is no card-chain referenced in the message.

Chain-Active $\rightarrow$ {$code$ = -5 and $error$ = "CardChain is active, when inactive state is required!"} The card-chain has the current card-block in active state when the protocol phase requires to be inactive, e.g when registering a candidate.

Chain-Inactive $\rightarrow$ {$code$ = -6 and $error$ = "CardChain is inactive, when active state is required!"} The card-chain has the current card-block in inactive state when the protocol phase requires to be active, e.g when subscribing to a channel.

Custom error messages for the sequence are defined in the Acknowledge Errors. The **ack** message is always sent in the inverse direction of the original message that originated the error, where the $body$ format is: {$code$:<error code or 0 if OK>, $error$: <error message if code != 0>}. The Ack message doesn't have the $type$ header field, it's always expected to be a reply concluding the session phase.

A. Challenge

The challenge is a procedure used to authenticate a component, it can be embedded in other protocol phases or requested at any moment between components. Messages are normally not directly sent between components, but routed through the gateway.

Message Sequence:
ANY [**req-cha** / Request Challenge] $\longrightarrow$ ANY
ANY [**rpl-cha** / Reply Challenge] $\longleftarrow$ ANY

Body Structures:

req-cha $\rightarrow$ {$secret$: string, key: string, $mode$: map} The $secret$ containing an encrypted random string. The header to field is used to reference the CardChain and the key to identify the CardBlock to be used when encrypting the $secret$. The $mode$ will specify the encryption scheme and all the necessary information to decrypt the $secret$ using the encryption scheme, e.g. Elliptic Curve Diffie-Hellman public-key (ECDH), the initialization vector (IV), etc.

rpl-cha $\rightarrow$ {$sigc$: string} Returns the signed challenge in the $sigc$ field. The signature method is implicit from the previous selected CardBlock.

Encryption Modes:

The **req-cha** message can support several encryption modes depending on the FI-Card implementation software. The only mandatory mode for the protocol version "1.0" is the **ECDH-AES-CBC** mode. This mode requires the fields: $suite$ = "ecdh-aes-cbc" in the $mode$ map, the $curve$ for the Elliptic Curve Diffie-Hellman algorithm [15], the $from$ header field with the base-64 encoded public-key, that in association with the private-key of the selected CardBlock will generate the symmetric key for the AES [16]. As also the initialization vector iv for the AES-CBC operation mode. Reusing the same key on the $from$ field allows for the user to permanently authorize any challenges for that key without confirming the challenge sentence every time, however this should be used with care because automatic authorization on applications can compromise security. Encrypt-then-MAC [17] schemes are not needed, since compromising the $secret$ integrity will fail to complete the challenge.

Acknowledge Errors:

req-cha $\rightarrow$ {$code$: 1, $error$: "Out of sync, needs to evolve."} Sent when the key field corresponds to an outdated CardBlock. Meaning that the CardChain has a different CardBlock as the header. The application needs to evolve in order to correctly use the chain.

req-cha $\rightarrow$ {$code$: 2, $error$: "Unsupported encryption mode."} Sent if the FI-Card software doesn't support the encryption mode provided in the $mode$ field.

Attack Vectors: Challenge requests can be sent by any one. Overloading the FI-Card with many requests can confuse the user that needs to complete and authorize the challenge. To minimize this, the **req-cha** message encrypts the challenge into the $secret$ field. E.g. when firing the login procedure via the FI-Application, the user can write a random sentence in the login form and then confirm if it's the same sentence when it appears on the FI-Card software. This provides an actual Out-of-Band authentication mode without requiring to memorize a passwords.

B. Register

Any time a component wants to register a new CardChain or a CardBlock candidate in the FI-Gateway it will send the **req-reg** message, and receive the corresponding **ack** with the field ok = true in a successful registration. Candidates are only accepted for an inactive CardChain.

Message Sequence:
ANY [**req-reg** / Request Register] $\longrightarrow$ G
ANY [**ack** / Acknowledge] $\longleftarrow$ G

Body Structures:

req-reg $\rightarrow$ {$type$: "new"|"cand", $card$: CardBlock} The $type$ field will discriminate if the request if for a new CardBlock or a candidate. The CardBlock structure is defined in Section III-A.

Acknowledge Errors:

req-reg $\rightarrow$ {$code$: 201, $error$: "Already existent card-block in the card-chain."} Sent when a CardBlock with the same uuid and public-key already exists in the gateway.

Attack Vectors: Any component can try to fork a canceled CardBlock with different T-Link's. **G** can easily reject this

with error $code$ = 201 if it has the original CardBlock. However if **G** is a different one, it has no information to reject the request. Any CardChain synchronization method between FI-Gateway's is not in the scope of this protocol, and any future developments on this should be aware of the fork issue. The alternative is to treat every gateway as an independent identity domain.

C. Subscribe

Any component that needs a keep alive connection to receive routed messages from the FI-Gateway (in push mode), must first subscribe and confirm it's address. The **req-sub** message will send the CardChain UUID, that will be the component address. To avoid creating fake routes a simple acknowledge is not enough, as so **G** will request a challenge **req-cha** containing random data that needs to be signed by the active private-key, and sent with a **rpl-cha** in a limited time-frame.

Message Sequence:
ANY [**req-sub** / Request Subscribe] ⟶ G
ANY [**req-cha** / Request Challenge] ⟵ G
ANY [**rpl-cha** / Reply Challenge] ⟶ G

Body Structures:
req-sub → {} The body is empty or non-existent, since the $from$ address available in the header will be used for the routing address.

Acknowledge Errors:
No custom errors for the subscription phase.

Attack Vectors: A possible Men-In-The-Middle [18] attack is not really a security issue since we already consider **G** as a MITM component, any additional in between entities can at most create DoS, but can not gather any important information from intercepted messages.

D. Search

The search capability is important for applications external to the FI-Gateway to known the existence and status of CardBlock's. E.g. when recovering from a lost FI-Card the CardBlock candidate is created from an existing inactive CardBlock. The identity owner can search for their card in the gateway (providing the name) and generate a new candidate.

Message Sequence:
ANY [**req-sch** / Request Search] ⟶ G
ANY [**rpl-sch** / Reply Search] ⟵ G

Body Structures:
req-sch → {$query$: string} Search for CardBlock's using a formated $query$ string. The format is a sequence of terms in the form of <field>:<regex value> that can be connected by and and or operators, e.g: "name:Matias* and birthdate:1981-01-*". The usable fields are: all public fields in the Info header and the uuid.

rpl-sch → {$result$: map[]} The result is list containing information about the matched cards with the mandatory fields: the $uuid$ of the CardChain, the $active$ field signaling if the chain is in the active state and the $last$ CardBlock public-key on the chain.

Acknowledge Errors:
req-sch → {$code$: 401, $error$: "Invalid query format: <hints>"} Sent when the query format is not correct, providing hints in the places that are not compliant.

Attack Vectors: There are no security concerns about the search process. The process is read only and all the information stored in the CardChain is public.

E. Login

The login process is a simple challenge request **req-cha**, emitted by the FI-Application to the FI-Card address routed by the FI-Gateway. The login succeeds when the challenge is completed and authorized on the FI-Card software from some process or user. When an identity is used for the first time, the application must authenticate the FI-Card by showing the UUID in the login form for the user to confirm.

Message Sequence:
A [**req-cha** / Request Challenge] ⟶ FIC
A [**rpl-cha** / Reply Challenge] ⟵ FIC

Body Structures and **Acknowledge Errors** are already defined in the Challenge phase.

Attack Vectors: To avoid the already mentioned CardChain forks from different FI-Gateway's, the FI-Application should maintain a CardChain database for every gateway domain. Also application challenges can be requested by any one at any time. Filters can be applied on the decrypted $secret$ to discard possible challenge request attacks.

F. Cancel and Recover

As defined in Section III-C the FIC is an implicit FI-Trusted Link that can directly cancel the card using the Cancel **req-cr-lnk** message. When the FIC is compromised, both the cancellation and recovering processes are provided by a quorum of FI-Trusted Link's. When a required number of **req-cr-lnk** messages from the TL nodes are gathered, the respective CardBlock is canceled or renovated. Optionally, Cancel/Recover requests can be made explicitly with **req-cr** messages. This is useful for automatic procedures, but should be avoided as much as possible since it's a door for some attack vectors.

Message Sequence:
TL [**req-cr** / Request Cancel-Recover] ⟵ G
FIC|TL [**req-cr-lnk** / Request CRLink] ⟶ G
FIC|TL [**ack** / Acknowledge] ⟵ G

Body Structures:

req-cr → {$type$: "cnl"|"rec", $uuid$: string, $prev$: string, $next$?: string, $secret$: string, key: string, $mode$: map} The $type$ field will signal if it's a Cancel or Recover request. The $uuid$ and the $prev$ base-64 encoded public-key identifies the previous canceled CardBlock. The $next$ public-key identifies the CardBlock candidate, and is only required for recovering. This message is only needed for an explicit Cancel/Recover request (explicit mode). The fields $secret$, key, $mode$ are used in the same way as in the challenge process.

req-cr-lnk → {lnk: CRLink} The CRLink structure is defined in Section III-C. This message doesn't demand an explicit **req-cr**.

Acknowledge Errors:

req-cr-lnk → {$code$: 301, $error$: "No card-block or t-link found."} Sent when there is not valid CardBlock or T-Link in the CardBlock.

Attack Vectors: When an FI-Card is compromised the cancellation process should not be delayed, in the same manner as for a Credit Card. The private key stored in the FI-Card must be protected with a strong PIN, enough to give a secure time-frame for the cancellation process. Once a CardBlock is canceled any one can try to register alternative candidates and overflow the **TL** with Recover **req-cr** messages, mixing attack requests with valid ones and confuse the **TL** (that can be a real person). To minimize this, all **req-cr** can be ignored and use the **TL** only in implicit mode, or use the optional field $secret$ (described in the Challenge phase) to authenticate (authentication mode).

G. Evolve

Evolve is a request for a CardChain synchronization. When any application external to the FI-Gateway is not in sync with the CardChain of the gateway, a synchronization is requested with **req-evl**.

Message Sequence:

ANY [**req-evl** / Request Evolve] ⟶ G
ANY [**rpl-evl** / Reply Evolve] ⟵ G

Body Structures:

req-evl → {$uuid$: string, $start$: string } Request all CardBlock's and CRLink's from the $uuid$ CardChain, starting at the CardBlock with the $start$ public-key.

rpl-evl → {$chain$: {$card$: CardBlock, $links$: CRLink[]}[], $sign$: string, $mode$: map} Returns an ordered list of CardBlock's and the respective CRLink's from the selected CardChain. The $chain$ is signed with the private-key of the gateway in the $sign$ field with the signature mode indicated in $mode$. This version requires in the $mode$ the fields $curve$ = "secp384r1" and $suite$ = "SHA3-256withECDSA".

Acknowledge Errors:

req-evl → {$code$: 501, $error$: "Starting at an undefined card-block."} When there is no CardBlock at the $start$ position of the CardChain.

Attack Vectors: Any CardChain information is public, but forks can exist between different gateways. Other components may require to correctly identify the correct fork, since the identity is actually composed of <uuid>@<idp-key>. **G** will sign the chain to authenticate the origin of the information.

V. Proof of concept

A proof of concept that validates the protocol phases and the needed steps to compile and reproduce the experiment is available at `https://github.com/shumy/federated-identity-card`. The software simplifies the authentication process by doing automatic authorization for the application/gateway challenges. This is a dangerous process, and should not be used in a production version. The project contains the core api in the src folder and the proof of concepts application in the $app - example$ folder. The application simulates an in-memory asynchronous FI-Gateway avoiding any implementations over a transport protocol. An in-memory client api implementation of the **IBroker** interface is also available, used by other components for message transmission to the gateway. The broker and gateway simulations implement full message serialization and deserialization, avoiding any restrictions on future transport protocol integrations.

The UI for the test application has 4 tabs that simulates simplified versions of the following components: the FI-Card, the FI-Application and a quorum of two Trusted Links. Every tab has the possible action buttons and input text fields to interact with, as also a read-only text area that will log important information about the performed action.

A. Testing Simulation

This section describes how to use the proof of concept application to perform a full round-trip of the described protocol phases. Every step corresponds to an action button on the respective tab:

FI-Card → Register/Subscribe This action will create a CardBlock with the necessary Elliptic Curve keys and the selected user name inserted in the input text field. The card is registered in the FI-Gateway with the generated UUID followed by the subscription phase. This process is automatically done to the other components when the application opens.

FI-Application → Login Insert the last created user name in the input text field and fire the action. This will perform a search for the respective card in the FI-Gateway if it's not already available in the local database. If the card is found, an evolve phase is requested to retrieve all the card-chain information. A challenge phase is finally requested against

the respective address. Note that, the challenge authorization at the FI-Card is done automatically, but this can be changed in a different implementation. The log area should report "—Login OK—".

FI-TL-1 and FI-TL-2 → Cancel Canceling the card is performed at the Trusted Links. Insert the user name in the first input text field and fire the action. The search and evolve phases are also used here to collect the initial card-chain information. The action must be performed in both FI-TL tabs to effectively cancel the card.

FI-Application → Login Try to login again in the application. Note that the login fails with the ACK message "CardChain is inactive, when active state is required!", informing that the card-chain (or account) is disabled due to cancellation.

FI-Card → Send Candidate This action will create a new CardBlock and register this as a candidate for the disabled card-chain at the gateway. The log should report the same UUID and a public-key corresponding to a different key pair. Copy this new key to the clipboard.

FI-TL-1 and FI-TL-2 → Recover Paste the clipboard key in the second text input field and fire the action. The action must be performed in both FI-TL tabs to effectively recover the card-chain to this new candidate.

FI-Application → Login Try to login again in the application. The action fails, but this time with the ACK message "Out of sync, needs to evolve.", meaning that the FI-Application local database is not up to date and doesn't have the recovered card.

FI-Application → Evolve This action will pull the card-chain including the recovered card. This action can be performed automatically, reacting to the respective ACK error message.

FI-Application → Login The login is successful using the recovered card.

The Cancel/Recover process can be tested indefinitely by evolving the FI-TL databases and registering new candidates with the FI-Card. Additional log details can be seen in the output console, including serialized messages transmitted by the in-memory broker.

VI. Conclusions

In this article it's proposed a protocol to secure identity cards based on asymmetric cryptographic keys. Securing the identity is achieved by sharing only the necessary information with the relying parties. Private keys are never transmitted in the protocol and no passwords are required for the authentication process, escaping the burden of password management. The card-chain structure shows that it's possible to evolve the account to a different key pair when the card is lost. Although this is not a standard procedure at the moment, the lack of a proper solution can be the reason for that and it can be a critical feature when adopting a software for smart-card management. Governments and banks can apply these managements features on identity and credit cards, accelerating the cancellation and renovation process, or even provide end users with an API to build custom software.

For this version the evolution of the card-chain is only available for the FI-Card software, implying a limitation where other components are not protected against compromised or lost cards. This was deliberately taken out on this first version so that complexity could not compromise security. Extensions to the protocol can be proposed later and security experts reviews should be performed before major adoption.

References

[1] S. S. Shim, G. Bhalla, and V. Pendyala, "Federated identity management," *Computer*, vol. 38, no. 12, pp. 120–122, 2005.

[2] M. Chan, "Corporate espionage and workplace trust/distrust," *Journal of Business Ethics*, vol. 42, no. 1, pp. 45–58, 2003.

[3] G. Elahi, Z. Lieber, and E. Yu, "Trade-off analysis of identity management systems with an untrusted identity provider," in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*. IEEE, 2008, pp. 661–666.

[4] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 2009, pp. 391–405.

[5] L. Florio and K. Wierenga, "Eduroam, providing mobility for roaming users," in *Proceedings of the EUNIS 2005 Conference, Manchester*, 2005.

[6] P. Urien, "Convergent identity: Seamless openid services for 3g dongles using ssl enabled usim smart cards," in *Consumer Communications and Networking Conference (CCNC), 2011 IEEE*. IEEE, 2011, pp. 830–831.

[7] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[8] B. K. Sy and A. P. K. Krishnan, "Generation of cryptographic keys from personal biometrics: An illustration based on fingerprints," in *New Trends and Developments in Biometrics*. InTech, 2012.

[9] S. Berkovits, S. Chokhani, J. A. Furlong, J. A. Geiter, and J. C. Guild, "Public key infrastructure study," NATIONAL INST OF STANDARDS AND TECHNOLOGY GAITHERSBURG MD, Tech. Rep., 1994.

[10] R. Kainda, I. Flechais, and A. Roscoe, "Usability and security of out-of-band channels in secure device pairing protocols," in *Proceedings of the 5th Symposium on Usable Privacy and Security*. ACM, 2009, p. 11.

[11] I. Blake, G. Seroussi, and N. Smart, *Elliptic curves in cryptography*. Cambridge university press, 1999, vol. 265.

[12] I. Bashir, *Mastering Blockchain*. Packt Publishing Ltd, 2017.

[13] N. Briscoe, "Understanding the osi 7-layer model," *PC Network Advisor*, vol. 120, no. 2, 2000.

[14] G. Bierman, M. Abadi, and M. Torgersen, "Understanding typescript," in *European Conference on Object-Oriented Programming*. Springer, 2014, pp. 257–281.

[15] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," in *Towards a quarter-century of public key cryptography*. Springer, 2000, pp. 103–123.

[16] T. Jamil, "The rijndael algorithm," *IEEE potentials*, vol. 23, no. 2, pp. 36–38, 2004.

[17] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *Advances in CryptologyASIACRYPT 2000*, pp. 531–545, 2000.

[18] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 78–81, 2009.