Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Lightweight Offline Access Control for Smart Cars

Gian-Luca Frei, Fedor Gamper, Prof. Dr. Annett Laube
Bern University of Applied Sciences TI -ICTM

Presenter

Gian-Luca Frei

- Bachelors' studies in computer science with a specialization in IT-Security at Bern University of Applied Sciences.
- Has done research on modern cryptographic protocols for which he has received the ISSS Excellence Award 2019.
- Is now working in the security industry. Topics include *Inter-banking Payment Protocols, Covid-19 Tracing, E-banking Security, Web Application Security.*

Overview

1. Introduction
1. Example Use-Case
2. Features

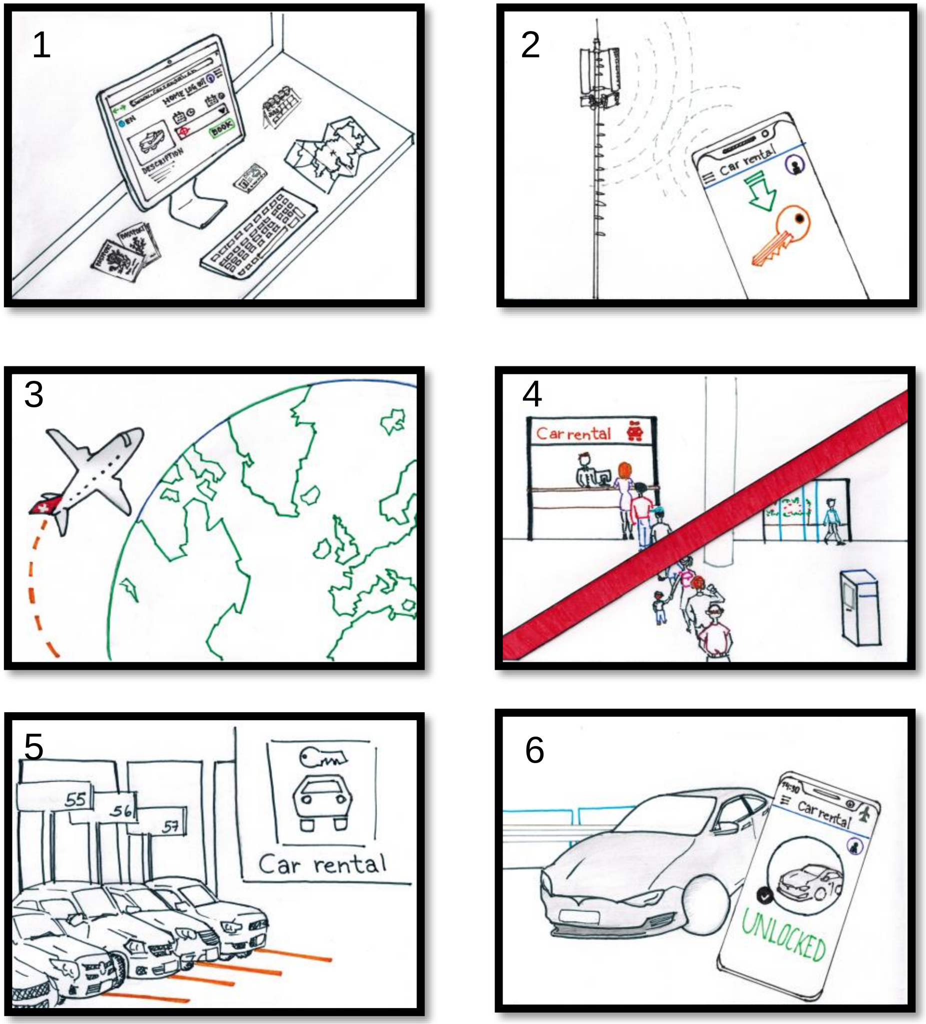
2. Protocol
1. Principle
2. Public Key Recovery
3. Authentication Mechanism
4. Protocol Phases

3. Prototype
1. Prototype
2. Performance

Introduction

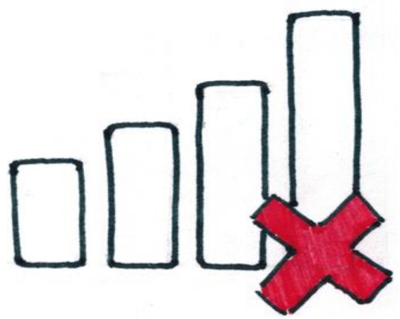
1. Example Use-Case
2. Features

Example Use Case

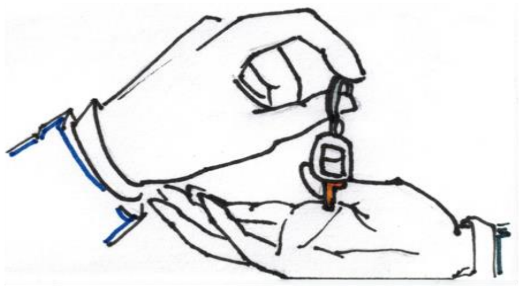


One example use case of the protocol is a international car rental company that wants to enables it’s customers to open the rental cars with their smartphones. In this case, the customer could make the booking online (1), download the digital key for the vehicle in advance (2), travel to the destination (3), avoid long queues (4), and directly go to the car that is waiting (5). Next, he can use the digital key to unlock the vehicle (6).

Features



offline

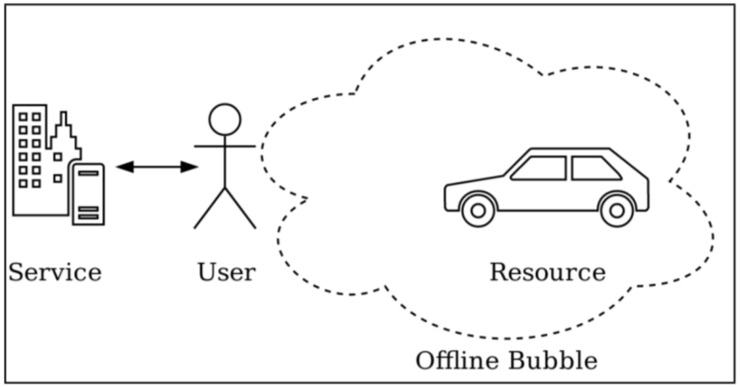


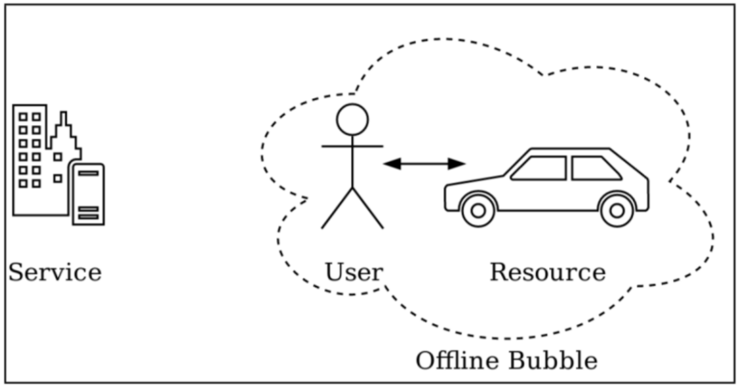
delegable

lightweight

The protocol's main features are the following: First, it requires no online connection to open cars. Therefore, it is suitable for applications where the cars and the users have no network connection. (For instance, the traveling customer). Second, it enables users to delegate their access rights to other users. Third, the protocol is designed for low-bandwidth channels like Bluetooth Low Energy and transports around 210 bytes per car access.

Offline Bubble





Designing access control solutions is quite easy if the cars have a stable network connection. The only parts needed are an authentication mechanism and a server that the car can query to check if a user is allowed to access it. However, maintaining a constant network connection is often not possible or not desirable because of the higher costs involved. Moreover, developing a protocol whereby all steps can be done with no network connection would not be meaningful in a world where most smartphones almost always have an internet connection. Therefore, we use the following networking model: The users are most of the time online and need a network connection for registration and to make bookings. Later, a user receives a credential that enables him to make use of his access rights in an offline fashion. This means that if he opens a car, he needs no network connection, nor does the car need a network connection. To illustrate this, imagine the car is located in an underground car park, where no cellular reception is available. In that case, the carsharing provider cannot communicate with the car, whereas a user can cross into the offline zone by entering the underground car park. Most carsharing providers allow their users to make spontaneous bookings over their smartphones. This means access control rules can change quickly. Therefore, the user needs to receive the credentials to authenticate and authorize himself outside before entering the underground car park.

Protocol Overview

1. Principle
2. Public Key Recovery
3. Signature Validation with Public Key Recovery
4. Possible Attacks

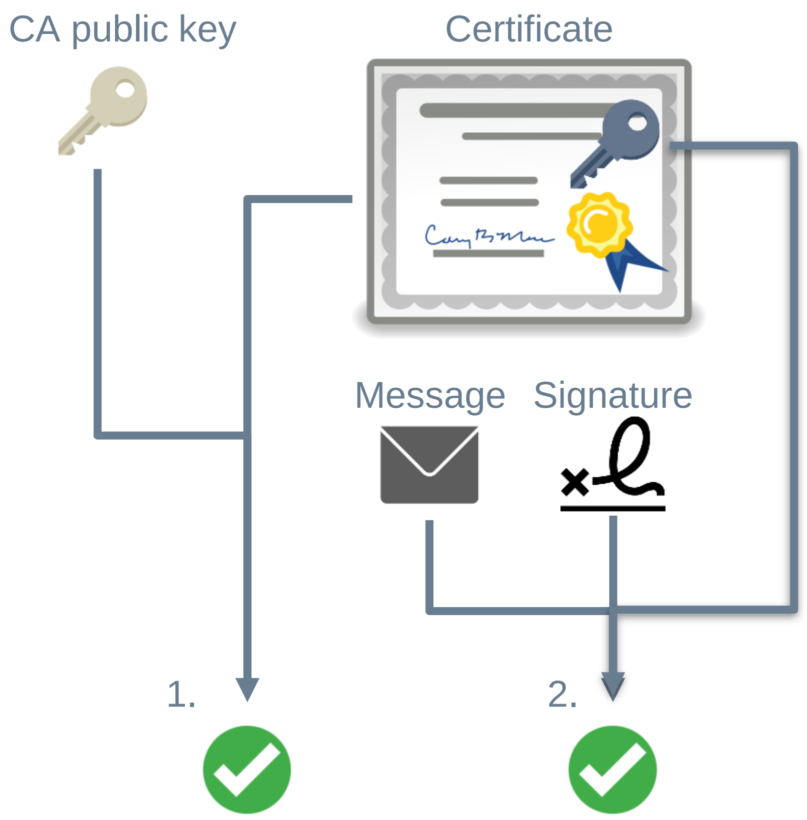
Principle

- The protocol is based on the principle of authorization tokens and a strong authentication mechanism with public key certificates. Each user has a device containing a unique private key used for authentication and a public key certificate.
- Further, each user has one or multiple authentication tokens, which are digitally signed messages that link access rights to a specific user. These tokens are independent of the private key used for authentication and can be shared between multiple devices of one user. For example, if a user changes his smartphone, then he needs to onboard his new phone to generate a new private key and get a new public key certificate, and then he can copy his authorization tokens to his new device.
- The user can then create an access request on his device. Each access request is authenticated with the private key and linked to the user with the public key certificate.

Normal Signature Verification with Certificates

This graphic shows how a signature verification with standard certificates works:

1. Validate the certificate with the public key of the certificate authority
2. Check if the message was signed with the public key from the certificate



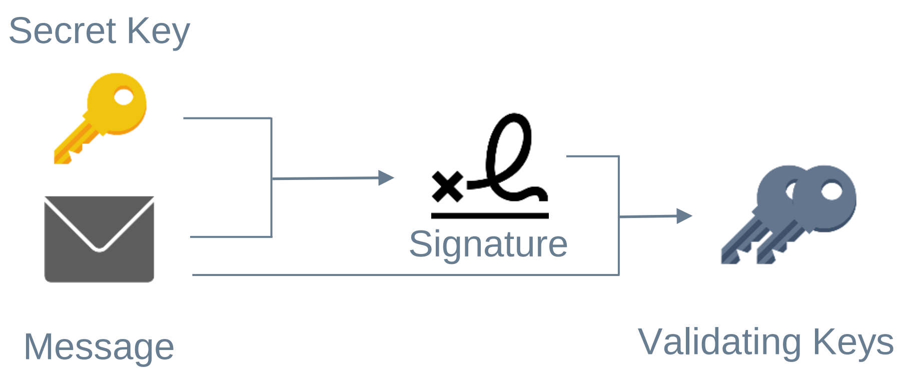
Normal Certificates

$$Cert = \begin{pmatrix} username \\ validity \\ publicKey \\ Sign_{CA}(H(username, validity, publicKey)) \end{pmatrix}$$

Normal certificates contain the name of the subject (username), its validity period, the public key of the subject as well as a signature of the other values. (In practice, certificates contain many more metadata values such as CA name. However, those are not relevant to understand the protocol.)

Public Key Recovery

To minimize the message sizes, the presented protocol uses public key recovery.



Public Key Recovery:

- Idea: Recover validating public-key from message and signature
- Possible with ECDSA and Schnorr-Signatures
- Multiple validating keys
- Additional recovery parameter in signature to identify the ’right’ key (2 bit)

Public Key Recovery with ECDSA

The protocol is independent on the cryptographic algorithm that is used as long as it allows public key recovery. The following overview explains the public key recovery process with ECDSA:

Signature Generation:

To sign a message m the private key sk is needed. Because m is normally not an element of the finite field F, a mapping algorithm is needed. This is done by hashing the message and singing the resulting scalar e = hash(m).

Select a random scalar $0 < k < q$

Compute the curve point $R = kA$

Let r the x coordinate of R

Compute $s = k^{-1}(e + sk * r) \mod q$

The signature is the pair of the skalars (s, r)

Verification:

To verify if a signature (r,s) is valid for a message m (respectively hash e = hash(m)) and a public key pk, the verifier computes:

$u_1 = s^{-1}e \mod q$

$u_2 = s^{-1}r \mod q$

$P = u_1 A + u_2 pk$

$r' = x_P$, where x_P is the x coordinate of P

And tests if $r' \stackrel{?}{=} r \mod q$

Public Key Recovery with ECDSA

Public Key Recovery:
It is not obvious, but given a hash e and signature (r, s), a public key pk0 can be computed for which the signature will be valid. The r part of the signature is the x coordinate of the point R. Therefore, two points R1 and R2 can be computed which have the same x coordinate as R.
This works by rearranging the formula of the elliptic curve y 2 = x 3 + ax + b mod q to y1,2 = √ x 3 + ax + b mod q. There are many algorithms for calculating this type of equations.
For both of resulting points R1 = (x, y1) and R2 = (x, y2) we can compute a public key for which the signature is valid pk1 = r −1 (sR1 − eA) and pk2 = r −1 (sR2 − eA). (This is simplified and only valid for elliptic curves with co-factor 1)
Why this works, can be seen when validating the signature with the recovered public key pk1.

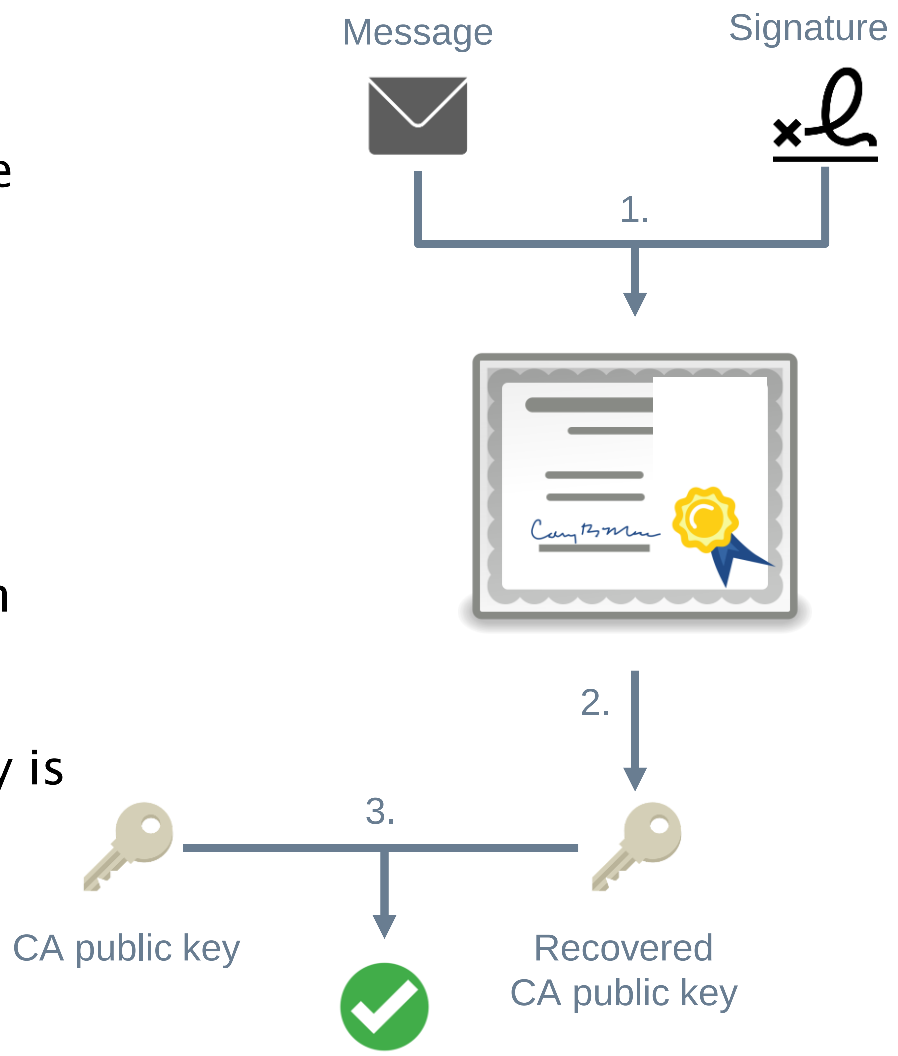
$$P = u_1 A + u_2 pk_1$$
$$P = s^{-1} eA + s^{-1} r(r^{-1}(sR_1 - eA))$$
$$P = s^{-1} eA + s^{-1}(sR_1 - eA)$$
$$P = s^{-1}(eA + sR_1 - eA)$$
$$P = s^{-1}(sR_1)$$
$$P = R_1 \text{ has the x coordinate } r$$

For the second validating key pk2 it is the same. The signer could add a small recovery parameter j to the signature. This small number (0 <= j < 4) allows the verifier to determine which is the right public key. (Used in the prototype)

Signature Verification with Public Key Recovery

This graphic shows how a signature verification with the minimal certificates from the protocol:

1. Recover verifying key from message and signature.
2. Recover a public key which validates the certificate.
3. Check if this recovered key is the public key of the CA.



Minimal Certificates

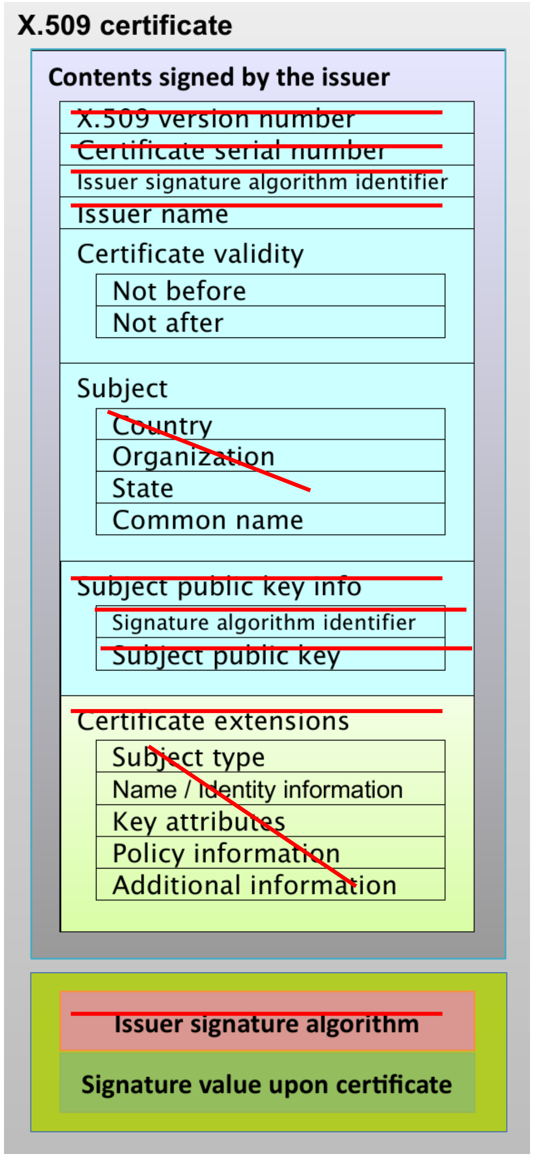
The minimal certificates used by the protocol are signed in the same way as standard certificates but don't contain the subject's public key. This key can be recovered from the signature that is generated later.

$$Cert = \begin{pmatrix} username \\ validity \\ publicKey \\ Sign_{CA}(H(username, validity, publicKey)) \end{pmatrix}$$

More Improvments

Comprared to X509 certificates contain many more fields that are all not used for the cretificates in the protocol

- Many fields are not needed in a closed system
- Subject public key can be reconstructed from authentication signature
- Issuer can be derived with public key recovery



Authentication

- Public key certificate
- Instead of challenge response the challenge is the current time
  - Advantage: Only a half roundtrip

Authorization

- Authorization tokens are digitally signed messages which bind access rights to a specific user
- Valid for limited time-period
- Different trust roots for certificates and authorization tokens

Protocol Details

Detailed description in the paper

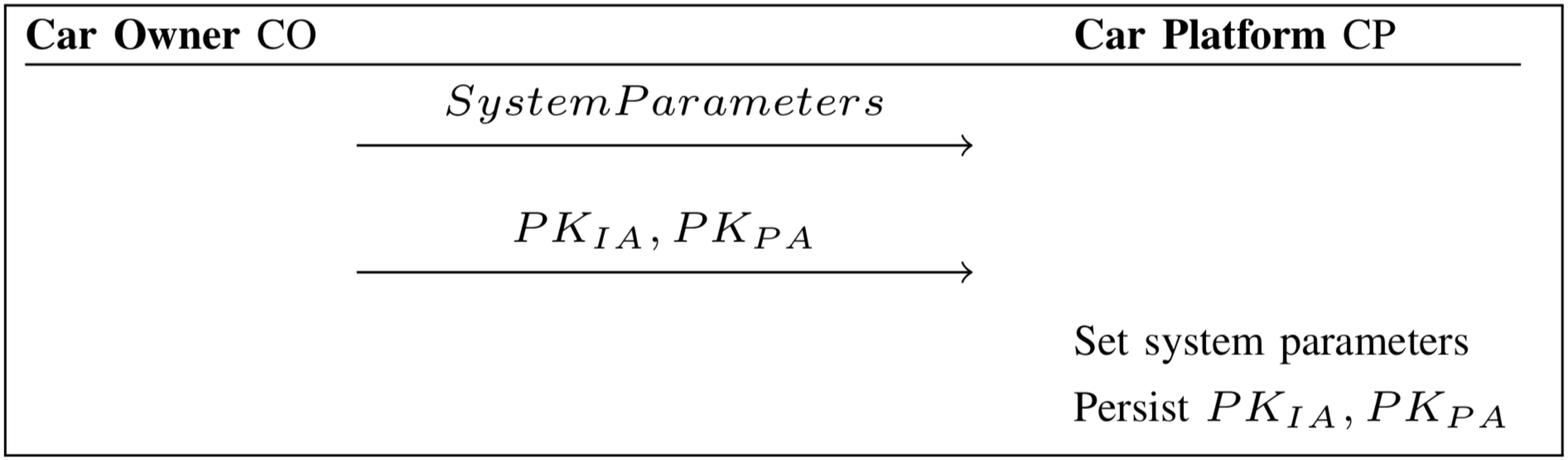


Figure 2. Car Initialization Sequence Diagram

Protocol Details

Detailed description in the paper

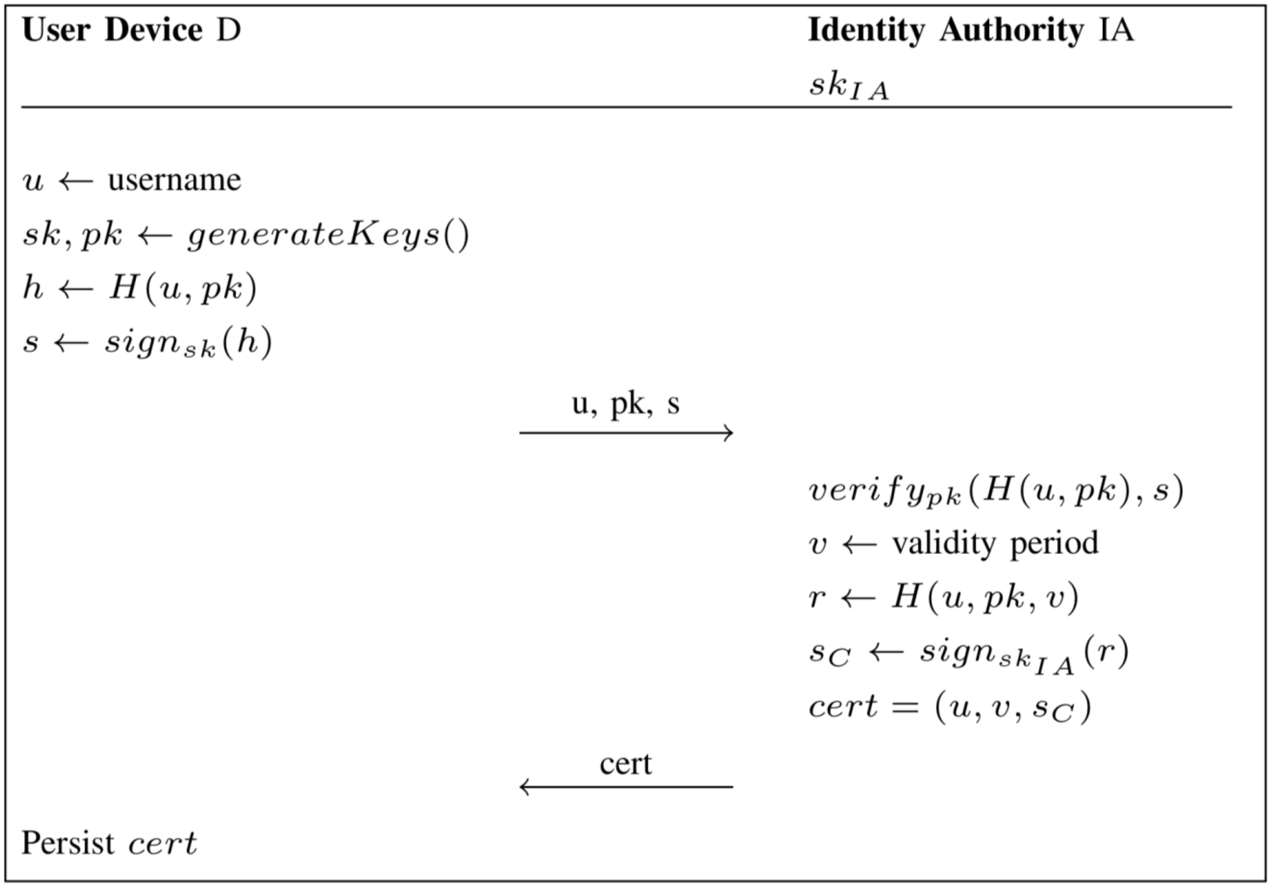


Figure 3. Device Onboarding Sequence Diagram

Protocol Details

Detailed description in the paper

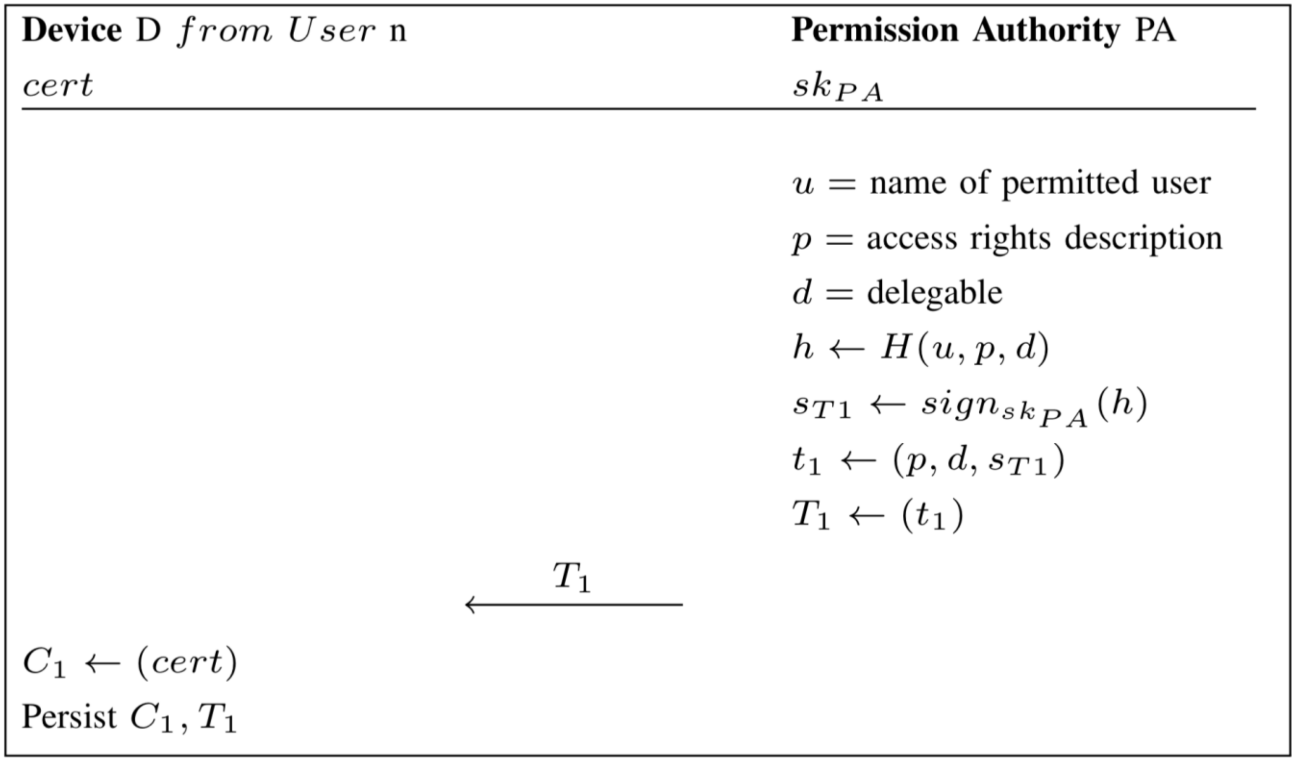


Figure 4. Root Token Issuance Sequence Diagram

Protocol Details

Detailed description in the paper

Delegate Device D		**Receiver Device** D'
(from user $n-1$)		(from user n)
$sk_{u_{n-1}}, T_{n-1}, C_{n-1}$		$cert_n$
$h_n \leftarrow H(u_n, p_n, d_n)$		
$s_{Tn} \leftarrow sign_{sk_{u_{n-1}}}(h_n)$		
$t_n \leftarrow (p_n, d_n, s_{Tn})$		
$T_n \leftarrow T_{n-1} \| (t_n)$		
	$\xrightarrow{C_{n-1}, T_n}$	
		$C_n \leftarrow C_{n-1} \| (cert_n)$
		persist C_n, T_n

Figure 5. Token Delegation Sequence Diagram

Protocol Details

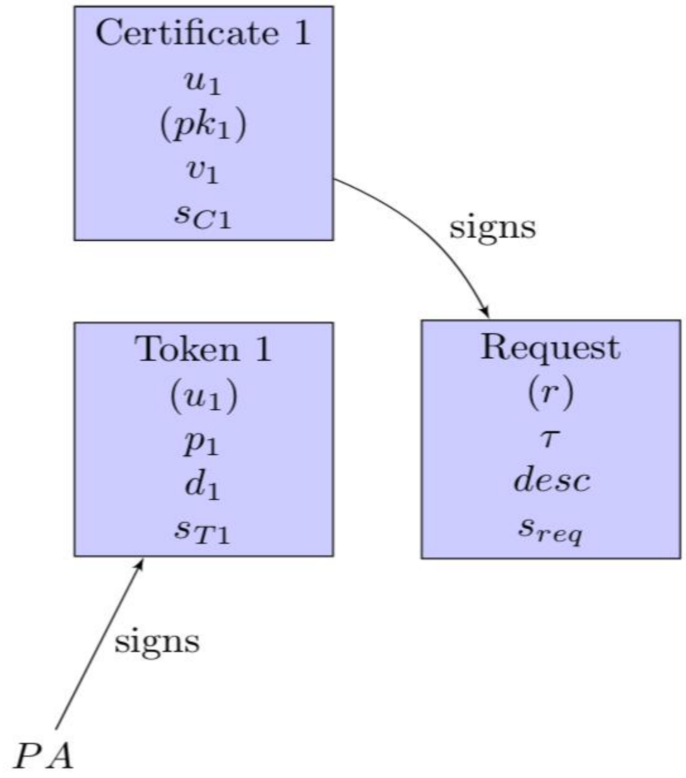
Detailed description in the paper

User Device D $from\ User$ n		**Car Platform** CP
u_n, sk_n, T, C		PK_{IA}, PK_{PA}
$r \leftarrow$ car name		
$\tau \leftarrow currentTime()$		
$desc \leftarrow$ additional information		
$s_{req} \leftarrow sign_{sk_n}(H(r, \tau, desc))$		
$req = (\tau, T, C, desc, s_{req})$		
	$\xrightarrow{req}$	
		$validateRequest(req)$
		$carAccess(desc)$

Figure 7. Access Request Sequence Diagram

Overview

This graphic shows how the different token and certificates are linked to the request. Values in parenthesis mean that the value is not stored in the message itself, but later reconstructed when validated.



Overview

This graphic illustrates the chain of public key certificates and access tokens that are used when digital keys are delegated to another person. In this example user 1 delegated his rights to user 2, which again delegated to user 3.

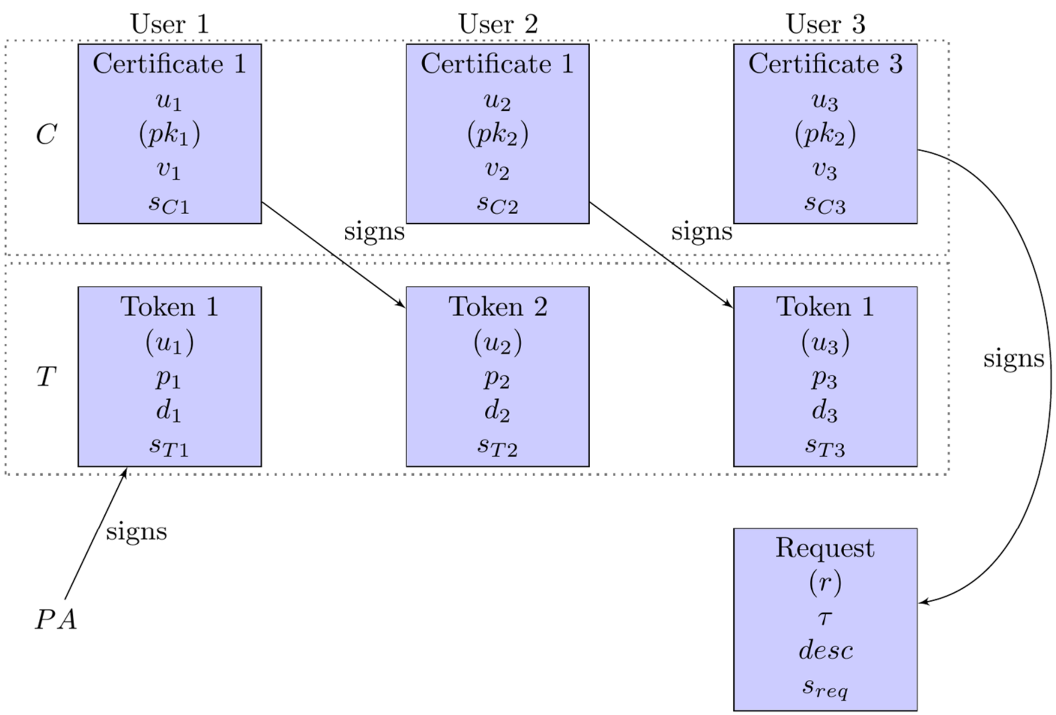


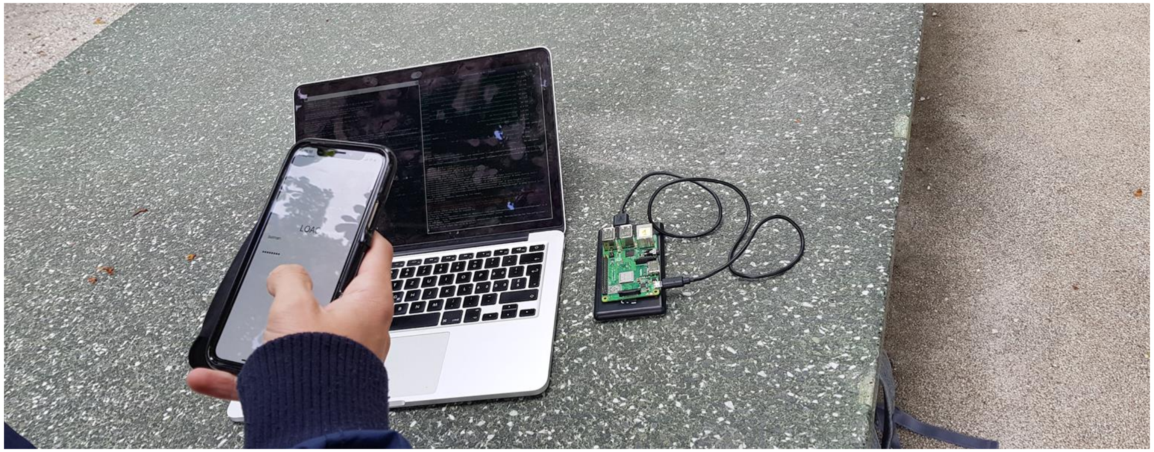
Figure 2.6: Access Request Example

Prototype

1. Prototype Overview
2. Demo
3. Performance
4. Conclusion

Protoytpe

To test the performance in a real scenario, a prototypic system was developed with a smartphone app that communicated over Bluetooth Low Energy to a Raspberry Pi that simulates a car. Additionally, a web application was developed that issues access tokens and public key certificates. Instead of a full car, a coffee machine was used as a replacement. Main reason: The authors of the protocol like coffee.



Prototype Performance

	Transmission Size	Transmission Time	Computation Time	Total Time
No Delegation	259 Bytes	247 ms	248 ms	495 ms
One Delegation	433 Bytes	282 ms	284 ms	566 ms
Two Delegations	608 Bytes	331 ms	401 ms	732 ms

The total time for the request gets longer with more delegations. However, <1s is still practicable for most applications. The computation time for validates takes almost as long as the time for transmission. This is because the public key recovery operation with ECDSA is very expensive. However, it must be noted that the prototypic NodeJs implementation of the algorithm could be replaced with a much faster native implementation. In that case, the computation time would be almost negligible.