QuantumCharge: Post-Quantum Cryptography for Electric Vehicle Charging – preprint –

Dustin Kern[1], Christoph Krauß[1], Timm Lauser[1], Nouri Alnahawi[1], Alexander Wiesmaier[1], and Ruben Niederhagen[2,3]

[1] Darmstadt University of Applied Sciences, Germany
{firstname.lastname}@h-da.de
[2] University of Southern Denmark, Odense, Denmark
[3] Academia Sinica, Taipei, Taiwan
ruben@polycephaly.org

Abstract. ISO 15118 enables charging and billing of Electric Vehicles (EVs) without user interaction by using locally installed cryptographic credentials that must be secure over the long lifetime of vehicles. In the dawn of quantum computers, Post-Quantum Cryptography (PQC) needs to be integrated into the EV charging infrastructure. In this paper, we propose QuantumCharge, a PQC extension for ISO 15118, which includes concepts for migration, crypto-agility, verifiable security, and the use of PQC-enabled hardware security modules. Our prototypical implementation and the practical evaluation demonstrate the feasibility, and our formal analysis shows the security of QuantumCharge, which thus paves the way for secure EV charging infrastructures of the future.

Keywords: Post-Quantum Cryptography, Electric Vehicle Charging, Formal Security Verification, ISO 15118, Hardware Security Module

1 Introduction

One of the driving technologies of the 21st century is going to be quantum computing. However, while quantum computing promises great impact, e.g., in the fields of chemical and biological engineering, artificial intelligence, financial services, and complex manufacturing [15], it also poses a severe threat to our current IT security. While current prototypes of quantum computers are not yet large and stable enough to pose an immediate threat, experts predict that in the upcoming years practical attacks are becoming more and more likely [46]. For instance, Shor's algorithm [56] provided that it is executed on a sufficiently large and stable quantum computer, can break the asymmetric cryptography that is currently in wide use — RSA, DSA, DH, and ECC. This will affect all of our current IT infrastructures including automotive protocols.

In recent years, the term Post-Quantum Cryptography (PQC) has been established for the next generation of cryptography providing protection against crypt-analytic attacks aided by large quantum computers. Much effort is taken to develop and integrate PQC [3], the most prominent of which is the ongoing standardization effort [48] of the National Institute of Standards and Technology (NIST) where the first candidates for standardization were stipulated in July 2022. However, PQC's requirements regarding computational power, storage, and memory pose challenges for its use in resource-restricted devices [9].

With the long lifespan of entities in the e-mobility context, such as 10+ years for Charge Points (CPs) [58] or up to 35 years for Electric Vehicles (EVs) [35], considerations of PQC and crypto-agility are critical in order to maintain security. One of the most important e-mobility standards is ISO 15118 [34,35], which defines a Plug-and-Charge (PnC) protocol enabling charging and billing of EVs based on cryptographic credentials installed directly in the car making RFID cards or apps obsolete. An integration of PQC schemes into ISO 15118, however, is not straightforward. First, several entities defined in this standard are embedded devices and as such subject to strict resource restrictions hampering the integration of PQC. Second, ISO 15118 is not crypto-agile, i.e., it does not provide a mechanism to add and agree upon new cryptographic schemes, which requires changes to the standard when introducing PQC. Instead, it specifies hard requirements for the used cryptography (such as specific algorithms and parameters or limited execution times and data sizes) due to interoperability reasons.

Related work. An overview of the challenges and the current state of migrating applications and communication protocols to PQC is provided in [3]. Different platforms and use cases impose different requirements w.r.t. the choice of a suitable PQC scheme and parameters as discussed in [51,9]. PQC schemes are benchmarked and tailored to specific hardware and software platforms in [12,20]. Other works focus on special requirements for embedded/resource-constrained devices [42,5] or integrating PQC in micro-controllers [50,30]. Efforts for integrating PQC in prominent protocols include IKEv2 [59] and VPN [32,43]. There is also work on integration/migration strategies [9,22] and crypto-agility [2,41]. The integration into the industrial protocol Open Platform Communications Unified Architecture (OPC UA) has been discussed in [53] for cyber-physical systems.

A major focus has been given to PQC for Transport Layer Security (TLS): Several experiments and approaches are presented in the literature for integrating PQC into TLS including benchmarks [57,52], hybrid approaches [60,19], and investigations for embedded systems [16,18]. The Open Quantum Safe (OQS) project provides a collection of implementations of several PQC schemes in the library *liboqs* and integration into several popular Internet protocols including forks of the BoringSSL and OpenSSL libraries with the integration of PQC using liboqs [61].

The suitability of NIST candidates for Vehicle to Vehicle communication is assessed in [13] and [29] provides a light-weight identity-based two-party Authenticated Key Exchange for the Internet of vehicles. A case study of PQC for secure communication within a vehicle is given in [17] and a concrete implementation is evaluated for the protocol Lightweight Authentication for Secure Automotive Networks (LASAN) in [54]. Optimizations for PQC in automotive systems are provided in [24]. [65] provides an analysis of NIST PQC candidates for usage in Hardware Security Modules (HSMs) and proposes new sets of hardware accelerators for the future generation of the automotive HSMs.

To the best of our knowledge, no related work focuses on PQC security in EV charging protocols. However, some papers investigate the use of HSMs with ISO 15118. In [26], an approach is presented that secures an EV's PnC credentials, which are per ISO 15118 definition generated in the backend, by importing them into the EV's HSM (specifically a Trusted Platform Module (TPM) 2.0). Notably, the approach of [26] has been integrated into the current draft of the upcoming second edition of the standard, ISO 15118-20 [35]. However, the generation of private credentials in the backend results in a needlessly high risk of backend leaks. This issue is addressed in [25,27], where private keys are generated within the EV's HSM (specifically a TPM 2.0) and the EV only request a corresponding certificate from the backend. Critical aspects that are not considered in this context include: *(i)* crypto-agility, *(ii)* PQC-support including migration, and *(iii)* formal security proofs.

Our contribution. We propose QuantumCharge, an ISO 15118 extension for integrating PQC into the EV charging and billing process, with the following eight main contributions: *(i)* We analyze the cryptographic algorithms used in the standards w.r.t. their security against attacks from quantum computers and propose alternatives following NIST. *(ii)* We propose a migration concept to address the issue of existing legacy PnC entities that cannot be updated to support PQC. *(iii)* We propose a crypto-agility concept by supporting multiple PQC algorithms and extending the ISO 15118 process to support algorithm negotiation. *(iv)* We include changes to the protocol that enable symbolic proofs of the desired strong security properties. *(v)* We propose a concept for integrating PQC-enabled HSMs to ensure secure key generation, storage, and usage as well as other security services such as secure boot. *(vi)* We provide an implementation and performance evaluation showing the feasibility of our approach and compliance with ISO 15118 limitations such as timeouts. *(vii)* We perform a formal analysis of the central aspects of our protocol extension using the Tamarin prover to show the security of QuantumCharge. *(viii)* We release the adapted ISO 15118 implementation and Tamarin models as open source (cf. Secs. 6 and 7).

Structure of the paper. The remainder of the paper is structured as follows. Sec. 2 provides background on e-mobility and PQC on embedded systems. Sec. 3 introduces our assumed system and attacker model and Sec. 4 the requirements for QuantumCharge. Sec. 5 describes our QuantumCharge concept. Our formal security evaluation is described in Sec. 6 and the implementation and practical evaluation in Sec. 7. Finally, we conclude the paper in Sec. 8.

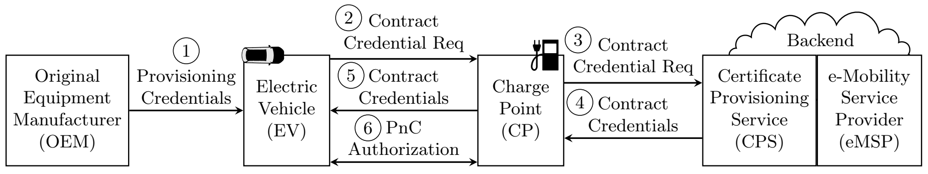
Original Equipment Manufacturer (OEM)
1 Provisioning Credentials
Electric Vehicle (EV)
2 Contract Credential Req
5 Contract Credentials
6 PnC Authorization
Charge Point (CP)
3 Contract Credential Req
4 Contract Credentials
Backend
Certificate Provisioning Service (CPS)
e-Mobility Service Provider (eMSP)

Fig. 1. E-mobility architecture.

2 Background

In this section, we introduce background on e-mobility and relevant standards as well as PQC and its performance on embedded systems.

2.1 E-Mobility

Fig. 1 shows a simplified e-mobility architecture. It adopts the definitions of the ISO 15118 standard [33,34] for entities and processes related to PnC authentication for EV charging. The entities include the EV, the CP, and several PnC backend systems. For credential management, only the EV's Original Equipment Manufacturer (OEM), the e-Mobility Service Provider (eMSP) backend, and the Certificate Provisioning Service (CPS) are relevant. The EV user has a contract with the eMSP, which enables PnC authorization and billing based on cryptographic credentials. The CPS establishes trust in the credentials provided by an eMSP for an EV. The EV stores the required credentials and establishes a PnC session with the CP. The CP (managed by a Charge Point Operator; not shown) enables data transfer to the backend and authorizes the EV to use the charging service.

The ISO 15118 communication between EV and CP uses a TLS channel with unilateral authentication of the CP. The EV authenticates itself inside the TLS channel using a challenge-response protocol. In the upcoming second edition, ISO 15118-20 [35], TLS uses mutual authentication with a vehicle certificate installed by the OEM in addition to the challenge-response-based EV authentication within the TLS channel. ISO 15118-20 further includes more modern TLS cipher suites, the option for TPM-based credential protection, and backward compatibility. The communication between CPs and the backend uses different protocols, e.g., Open Charge Point Protocol (OCPP) [49], and is usually secured with TLS. We omit the details for the sake of simplicity. In the following, we describe the relevant steps for using PnC with a focus on the related application-layer security mechanisms.

EV Preparation. During production, the OEM generates the EV's provisioning credentials (Step 1 in Fig. 1) in the OEM backend and installs them in the EV. The credentials consist of a unique long-term identity called Provisioning Certificate Identifier (PCID), the key pair $\{PC_{pk}; PC_{sk}\}$, and the respective X.509 *OEM Provisioning Certificate* PC_{Cert} (which includes PCID and PC_{pk}).

In case the second edition of ISO 15118 is supported, the vehicle certificate and the corresponding private key are additionally generated and installed in the EV.

Generation of Contract Credentials. To enroll the EV for PnC charging services, the owner hands over the PCID to the eMSP when concluding a charging contract. The eMSP then generates contract credentials to enable the EV to authenticate itself against the CP. The eMSP generates a key pair $\{CC_{pk}; CC_{sk}\}$ and the X.509 *Contract Certificate* CC_{Cert} that contains a unique e-Mobility Account Identifier (eMAID) linked to the charging contract.

Contract Credentials Installation/Update. An EV's contract certificate CC_{Cert} and the respective key pair are installed when connecting the EV to a CP for the first time. First, a TLS channel is established (not shown in Fig. 1). In Edition 1 of ISO 15118, only the CP authenticates itself in the TLS handshake by using the private key corresponding to the Supply Equipment Communication Controller (SECC) certificate. The additional EV authentication is only supported in Edition 2 based on the vehicle certificate. After the TLS channel has been established, the EV sends a request to the CP to install the contract credentials (Step 2). The request contains PC_{Cert} and is signed with PC_{sk}. The CP forwards the request over the CPS to the eMSP (Step 3). The eMSP encrypts the private key CC_{sk} with PC_{pk} (from PC_{Cert})[4]. The CPS signs the response and sends it (Step 4) over the CP to the EV (Step 5). The EV verifies the authenticity of the received data, decrypts CC_{sk}, and stores the credentials for later use. Credential updates use a similar process, except that old contract credentials are used for signing/encryption instead of the provisioning credentials.

Contract Credentials Usage. To start charging, the customers plug the charging cable of their EV into a CP, and the charging process is automatically handled with ISO 15118 without further customer interaction. Again, a TLS channel is established first. Then, the EV uses the contract credentials to authenticate its concluded contract against the CP, which authorizes the charging session (Step 6). The EV provides its eMAID and contract certificate CC_{Cert} to the CP. The CP verifies the validity of the credentials and sends a random nonce to the EV, which in turn signs the nonce with the private key CC_{sk} and returns the signature to the CP. If the authentication succeeds, the CP activates charging. Afterwards, the EV can periodically sign meter readings with its private key CC_{sk} to confirm the status of the charging session (see [34], Section 8.4.3.13). Before billing, signatures regarding the metering may be verified by eMSPs.

2.2 Post-Quantum Cryptography on Embedded Systems

In 2016, NIST started a PQC standardization process [48] that aims to standardize schemes for key encapsulation and signing, which were selected from five

[4] The encryption is done symmetrically using Advanced Encryption Standard (AES) for which a symmetric session key is generated with an ephemeral-static Elliptic Curve Diffie Hellman (ECDH) key exchange using PC_{pk} as the static part.

main families of PQC algorithms: code-based, hash-based, isogeny-based, lattice-based, and multivariate schemes. We focus on signatures (with key generation in the HSM, cf. Sec. 5). After several rounds, three signature-scheme candidates were selected for standardization in July 2022 [47]: The two lattice-based schemes CRYSTALS-Dilithium [7] and FALCON [23] as well as the hash-based scheme SPHINCS+ [6].

Notably, IETF already has standardized the "stateful" hash-based PQC signature schemes XMSS [31] and LMS [44]. Compared to the "stateless" algorithm SPHINCS+, handling of the state in XMSS and LMS requires additional precautions along with existing key-handling practices. While in a vehicle equipped with a TPM the handling of a state can be realized fairly safely and easily (e.g., using a monotonic counter in the TPM), more effort may be required for backend servers without TPM or HSM — e.g., backup and workload-sharing strategies need to be adapted to take state management into account. Due to these additional requirements specific to stateful schemes, we do not investigate XMSS and LMS in further detail. However, since SPHINCS+ generally requires more computational resources and has larger signatures than XMSS and LMS, SPHINCS+ is a "worst case" hash-based signature scheme and the results for SPHINCS+ also apply as an upper limit to XMSS and LMS.[5]

Performance benchmarks on an embedded Arm Cortex-A53 platform[6] indicate that, in general, Dilithium stands out for its efficient key generation, signing, and verification [42]. FALCON is less efficient for signing and verification, yet still acceptable; however, key generation times are rather long. SPHINCS+ is considered the least efficient, with acceptable key generation and verification but extremely long signing times. Benchmarks of the pqm4 project [36,37], targeting a 32-bit Arm Cortex-M4 micro-controller, confirm the findings in [42] on the suitability of said PQC candidates. These results show that there are PQC schemes that can be used on embedded platforms and hence are promising candidates for the integration into EV charging protocols.

3 System and Attacker Model

Our system model assumes the e-mobility architecture with the entities and communication relations shown in Fig. 1. We require that EVs supporting QuantumCharge are equipped with an HSM supporting the required PQC algorithms (cf. the analysis in [65]) which can be used to locally generate keys in the EV

[5] In addition, XMSS and LMS require the selection of parameters that define the maximum number of signatures per public-private key pair. These parameters can be chosen large enough to support the total number of expected charging operations during the lifetime of a vehicle. Given a maximum lifespan of 35 years for an electric vehicle and at most two charging operations per day, a maximum of 2^{20} signatures would provide a significant margin. Nevertheless, procedures for re-keying must be put in place when using XMSS and LMS for this application.

[6] Arm Cortex-A53 platforms are increasingly used in more powerful automotive Electronic Control Units (ECUs), e.g., see Renesas product range [55].

(similar to [25,27]). Legacy EVs do not support QuantumCharge. Similarly, CPs and backend systems may or may not support PQC and QuantumCharge. Thus, we have several use cases ranging from all entities supporting QuantumCharge to no one supporting QuantumCharge.

The security of the PnC communication is of high importance as a successful attack can cause financial damages and may even harm power grid stability (cf. [1,67,38]). Thus, there are high incentives for attackers to compromise the security of the charging process [8]. In our attacker model, we distinguish attackers based on their area of influence. We consider the following four kinds of attackers with access to quantum computers. Moreover, we consider that cryptographic algorithms might become insecure during the lifecycle of vehicles and charging infrastructure, requiring cryptographic agility to be in place.

Compromised EV. We consider that the attacker has complete control over the charging vehicle. This encompasses remote attackers compromising the car as well as physical attacks. For example, the attacker could be the car owner or someone with temporary access in a car-sharing scenario. Moreover, car owners might be incentivized to manipulate the billing information to conduct charging fraud. Such an attacker might replace components of the car or temper with the firmware of ECUs, but does not have the capabilities to perform physical attacks on HSMs.

Compromised CP. We consider an attacker with complete control over the CP. As the billing information is generated by the CP, manipulating this data is difficult to prevent in case of a compromised CP. However, the attacker might try to manipulate, replay, or forge any data that a CP receives/forwards (e.g., metering confirmations generated by vehicles[7]).

Compromised Network. A network attacker has complete control over a network node between the CP and backend systems or between the CP and the EV. For example, a malicious device could have been installed within the charging cable or plug or directly behind the CP, allowing for manipulation of the sent and received communication. However, the attacker has no control over EV, CP, or backend components. As we aim for backward compatibility, downgrade attacks have to be considered with regard to this attacker (cf. Sec. 6).

Leaked Backend Data. An attacker might be able to extract information from backend systems. We assume that top-level secret keys, such as certificate signing keys, are sufficiently protected, but individual EV keys might not. Thus, private keys of EVs should not be stored in the backend.

4 Requirements

We define the requirements for security (RS) and feasibility (RF) as follows:

[7] While ISO 15118 allows an EV to confirm meter values via a signature (called meter receipt), using this signature for billing purposes is subject to local regulation [34].

RS_1 *Secure key storage:* Private keys must be stored in a protected and secure environment to prevent their leakage.

RS_2 *Secure cryptographic operations:* Cryptographic algorithms must be executed within a secure, tamper-resistant, and separated environment.

RS_3 *Key usage authorization:* Key usage must be limited to authenticated and trusted software to prevent unauthorized access.

RS_4 *Secure key provisioning:* Private keys must be generated in a secure environment and must never leave it.

RS_5 *PQC support:* Cryptographic algorithms used for security-critical functions must resist attacks by quantum computers.

RS_6 *Crypto-agility:* The system must provide a mechanism to update or replace outdated or broken cryptographic algorithms securely.

RS_7 *Secure Credential Installation*: Bilateral authentication of the data sent between CPS and EV for credential installation must be guaranteed.

RS_8 *Secure Charge Authorization*: The authenticity of charge authorization requests received by the CP must be guaranteed.

RS_9 *Charge Data Authenticity*: The authenticity of received charge data as attested by the EV must be guaranteed towards the eMSP.

RF_1 *Minimal overhead.* The system should keep extra communication and computational overhead within the constraints of existing standards.

RF_2 *Easy integration.* The system should not alter the message flow of existing protocols and only introduce minor changes to message content.

RF_3 *Continued operation.* The system should offer backward compatibility with regard to actors that do not support PQC.

5 Security Concept

In this section, we describe our analysis of cryptographic algorithms which need to be replaced with PQC algorithms, the general approach of QuantumCharge, and detail the integration of QuantumCharge into the PnC architecture.

5.1 Analysis of PnC Cryptographic Algorithms

We focus our analysis of the specified cryptographic algorithms on both editions of ISO 15118 and relevant processes as described in Sec. 2.1. The application-specific protocols of ISO 15118 (discussed in the following) operate inside a TLS channel between the EV and the CP as well as the CP and the backend. Since there already exists exhaustive work on the migration of TLS to PQC (e.g., [57,52,60,19,16,18]) and prototype PQC-TLS libraries (e.g., [61]), we consider TLS out of scope for this paper and assume a post-quantum secure underlying TLS channel in the following. Instead, we focus on application-layer security mechanisms, which need to be post-quantum secure independently of the individual TLS channels (e.g., credential installation between EV and the backend).

PKI and Signatures. ISO 15118 defines a (rather complex) Public Key Infrastructure (PKI) as the basis for credential installation and charge authorization. Public/private key pairs are used in the OEM provisioning and contract credentials. In addition, eMSPs, CPSs, and CPs are also equipped with public/private key pairs and corresponding certificates. All certificates and key pairs use Elliptic Curve Cryptography (ECC). Edition 1 uses the `secp256r1` curve and ECDSA with SHA-256 as the hash function. Edition 2 uses `secp521r1` and ECDSA with SHA-512 or `Curve448` and EdDSA with SHAKE256. The key pairs are used in the TLS handshake and on the application layer for signature generation. Any ECC keys and signature algorithms must be replaced with PQC alternatives.

The current NIST PQC API for signature algorithms assumes that complete messages and not message digests are signed. For simpler integration with XML signatures, we nevertheless keep the current approach of signing message digests also when using PQC algorithms, which introduces minor overhead but is directly interoperable with the existing protocols. Notably, while the XML signature specification includes a list of supported algorithms, it explicitly supports extensibility via application defined (in our case PQC) algorithms [10].

Key Establishment and Symmetric Encryption. The contract credential installation process as standardized in ISO 15118 is based on an ephemeral-static ECDH key agreement for establishing a symmetric key (cf. footnote 4) with the above-mentioned ECC curves for the respective editions. The hash function for the Key Derivation Function (KDF) is SHA-256 in Edition 1 and SHA-512 in Edition 2. The symmetric key is then used to encrypt the private contract key with AES-CBC-128 in Edition 1 and AES-GCM-256 in Edition 2. Using the classical ISO 15118 installation process requires a suitable replacement of Diffie-Hellmann, e.g., a PQC Key Encapsulation Mechanism (KEM), and PQC algorithms for ECC. Also, the key length for AES should be 256 bits and the length of the hash functions should be at least 512 bits. However, since we additionally integrate the approach from [25,27] in QuantumCharge and generate keys securely in the EV's HSM (cf. Sec. 5.2), we do not require PQC key encapsulation as a replacement for Diffie-Hellmann and symmetric encryption in QuantumCharge (except for the underlying TLS channel).

5.2 General Approach

QuantumCharge's general approach is to update ISO 15118 (and accordingly other PnC standards) to support PQC with minimal protocol changes while providing a high level of security. This is intended to enable easy adoption by the industry. QuantumCharge has five central parts as described in the following.

First, QuantumCharge integrates PQC algorithms based on our analysis described in Sec. 5.1. For signatures in TLS and at the application layer, we propose to use the PQC algorithms selected by NIST for standardization [47], i.e., the two lattice algorithms Dilithium [7] and FALCON [23] and the hash-based algorithm SPHINCS+ [6]. The idea is to provide a proof of concept with NIST-approved algorithms of different families to be able to switch between algorithms in case of compromise. On the application layer, we only need signatures since the encryption of private contact keys for transfer is replaced by a secure key generation in the EV's HSM. For TLS, we propose to replace Diffie-Hellmann with the KEM Kyber [14] (which is currently the only KEM selected by NIST), SHA-512 as hash function, and AES-GCM-256 for encryption. Dilithium, FALCON, and SPHINCS+ as well as Kyber are currently the only schemes that are being standardized by NIST; more schemes will likely follow in the future and then will be relevant in this context as well.

Second, QuantumCharge integrates a migration concept to address the issue of existing legacy PnC entities that cannot be updated to support PQC. Hereby we assume that only EVs and CPs are affected, but all backend systems can be updated. In addition to the PQC algorithms, the classical ECC algorithms, as standardized in both editions of ISO 15118 remain supported. At the beginning of an ISO 15118 charging session, EV and CP agree on the protocol version to be used, i.e., conventional ECC-based ISO 15118 or QuantumCharge. This protocol negotiation allows legacy ECC algorithms to remain supported. Note that negotiation may fail if EV or CP are configured to only accept PQC-secured connections.

Third, QuantumCharge provides crypto-agility by supporting multiple PQC algorithms and extending the ISO 15118 process with an algorithm negotiation (which is out of scope in the standard, cf. [V2G20-2320] in [35]). Algorithm negotiation is only performed if QuantumCharge was selected for this session and is extensible with any arbitrary (PQC) algorithms, e.g., in case the existing algorithms are broken or new more secure/efficient ones are developed. Again, negotiation may fail if EV and CP cannot agree on common algorithms.

Fourth, QuantumCharge generates all keys of the EV securely in an HSM in the EV. Keys for the EV are no longer generated in backend systems. Following the TPM-approach from [25,27] we assume that a generic HSM with PQC support is used. The HSM is used for secure key generation, secure storage of (private) keys, and for secure execution of cryptographic operations. Further, the HSM provides additional features for restricting the usage of keys to a trustworthy system state (e.g., based on secure/authenticated boot).

Fifth, QuantumCharge includes changes to the credential installation and PnC authorization processes of ISO 15118 to allow for formal security proofs. Specifically, we enable the verification of strong formal authentication properties to meet the respective security requirements from Sec. 4.

5.3 Extending ISO 15118 with QuantumCharge

To enable the use of QuantumCharge, it must be supported by all involved entities (cf. Fig. 1). However, it is still possible for QuantumCharge-enabled EVs to use legacy CPs with classic ECC according to the original ISO 15118 standards. For security reasons, though, EVs can also be configured not to charge at such CPs. In the following, we describe QuantumCharge in more detail by describing the components and the integration into the ISO 15118 Edition 1 protocol flow, as Edition 1 is still the prominent edition of the protocol today. However, as the

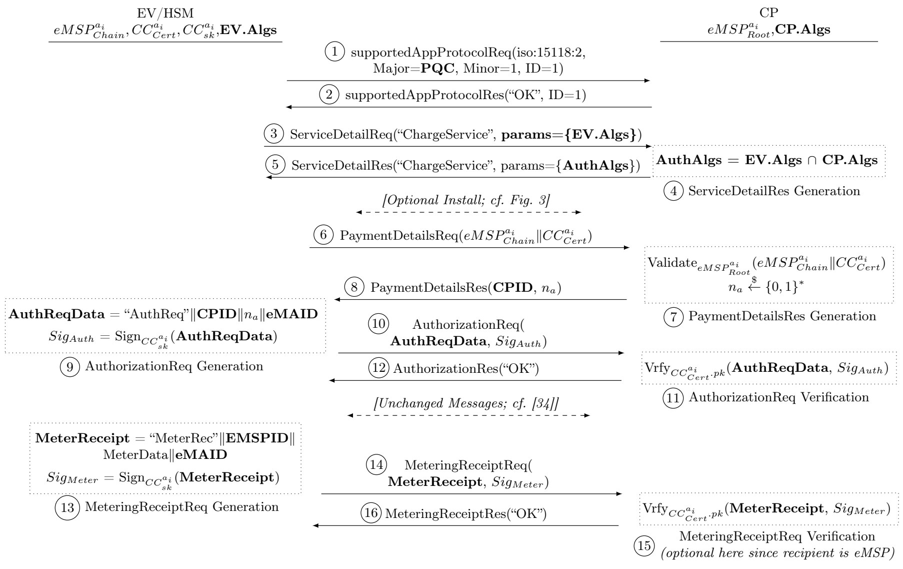
EV/HSM
$eMSP^{a_i}_{Chain}, CC^{a_i}_{Cert}, CC^{a_i}_{sk}$, **EV.Algs**
CP
$eMSP^{a_i}_{Root}$, **CP.Algs**
1 supportedAppProtocolReq(iso:15118:2, Major=**PQC**, Minor=1, ID=1)
2 supportedAppProtocolRes("OK", ID=1)
3 ServiceDetailReq("ChargeService", **params={EV.Algs}**)
AuthAlgs = EV.Algs ∩ CP.Algs
4 ServiceDetailRes Generation
5 ServiceDetailRes("ChargeService", params={**AuthAlgs**})
[Optional Install; cf. Fig. 3]
6 PaymentDetailsReq($eMSP^{a_i}_{Chain} \| CC^{a_i}_{Cert}$)
$\text{Validate}_{eMSP^{a_i}_{Root}}(eMSP^{a_i}_{Chain} \| CC^{a_i}_{Cert})$
$n_a \xleftarrow{\$} \{0,1\}^*$
7 PaymentDetailsRes Generation
8 PaymentDetailsRes(**CPID**, n_a)
AuthReqData = "AuthReq"‖**CPID**‖n_a‖**eMAID**
$Sig_{Auth} = \text{Sign}_{CC^{a_i}_{sk}}(\textbf{AuthReqData})$
9 AuthorizationReq Generation
10 AuthorizationReq(**AuthReqData**, Sig_{Auth})
$\text{Vrfy}_{CC^{a_i}_{Cert}.pk}(\textbf{AuthReqData}, Sig_{Auth})$
11 AuthorizationReq Verification
12 AuthorizationRes("OK")
[Unchanged Messages; cf. [34]]
MeterReceipt = "MeterRec"‖**EMSPID**‖ MeterData‖**eMAID**
$Sig_{Meter} = \text{Sign}_{CC^{a_i}_{sk}}(\textbf{MeterReceipt})$
13 MeteringReceiptReq Generation
14 MeteringReceiptReq(**MeterReceipt**, Sig_{Meter})
$\text{Vrfy}_{CC^{a_i}_{Cert}.pk}(\textbf{MeterReceipt}, Sig_{Meter})$
15 MeteringReceiptReq Verification (optional here since recipient is eMSP)
16 MeteringReceiptRes("OK")

Fig. 2. Usage procedures of contract credentials (changes in **bold**).

adoption of Edition 2 is expected to increase, we already explicitly consider the integration of QuantumCharge into ISO 15118-20 at the end of this section. The descriptions assume an error-free process. In the case of an error, the respective process may be aborted (e.g., as described above, if EV and CP fail to agree on an algorithm).

EV manufacturing. In the EV production, an HSM supporting ECC and PQC algorithms is installed. We assume the HSM ensures that keys are generated in the HSM (and not imported) and that private key export is not allowed.

During customization, multiple key pairs $\{PC^{a_i}_{pk}, PC^{a_i}_{sk}\}$ are generated by the HSM and the OEM creates the corresponding *OEM provisioning certificates* $PC^{a_i}_{Cert}$, where a_i denotes the respective cryptographic algorithm of the set of supported algorithms $\{a_1, \ldots, a_n\}$. In addition to the classic ECC algorithms EC-/EdDSA (for use with legacy systems), we support the PQC algorithms as mentioned in Sec. 5.2 with their different parameter sets. Key usage can be restricted to specific policies, e.g., access is only possible after a secure boot. We omit details on this aspect and refer to the TPM-based approach in [25,27].

Contract Credential Usage. Fig. 2 shows the PnC authorization process using the contract credential. First, the TLS channel between EV and CP is established (not shown)[8]. When a communication session is initiated by an EV, a

[8] Since PQC for TLS is addressed in other work (see Sec. 1), we omit the details on TLS in the following and discuss only the application layer of ISO 15118. In our current design, the choice of algorithms is done independently for TLS and application layer but could also easily be coordinated.

handshake is performed to negotiate the used application protocol and version. More specifically, the standard defines a major and a minor version number. The major number is used for the protocol version of either Edition 1 or 2. The minor number can indicate minor changes of the protocol (e.g., additional data elements). The EV sends a prioritized list of supported application layer protocols/versions to the CP, which responds with its selection.

To enable QuantumCharge and support backwards compatibility, we extend this initial application protocol/version negotiation. Notably, the minor version number cannot be used to indicate the use of PQC since with ISO 15118-20 a divergence in minor numbers is not indicated to the EV by the CP (if the major number matches and the minor number is lower, the CP simply responds with an *OK*; cf. [35] [V2G20-170]). To detect that the backwards compatibility mode is needed, however, the EV would require this indication. Hence, we show QuantumCharge support via the definition of new major numbers such that there are now four rather than two valid flags for the major number (for Edition 1 or 2, each with or without PQC). The major and minor numbers are used in the prioritized list sent by the EV (Step 1 in Fig. 2). The CP selects the protocol version and replies with an *OK* (Step 2).

If both sides support QuantumCharge, an additional handshake is performed to negotiate the signature algorithm using modified service detail messages (for the "ChargeService"; Steps 3-5). The EV sends a prioritized list of supported algorithms to the CP. The CP responds with its selection. At this point, the EV may start a contract credential installation as detailed later in this section.

If the EV possesses contract credentials using the chosen algorithm (from Step 5), it can start the PnC authorization process by sending its corresponding contract certificate chain to the CP (Step 6). The CP verifies the chain based on a locally installed eMSP root and responds with its unique ID and a fresh nonce n_a (Step 7-8). Afterwards, the EV uses its HSM to sign its authorization request data (including eMAID and the CP's ID to identify the sender/recipient of this signature, which is essential for the verification of formal authentication properties) and sends the signed data to the CP (Steps 9-10). The CP verifies the signature using the public key from the previously (in Step 6) received $CC^{a_i}_{Cert}$ and if the signature is valid, the EV is cleared to charge (Steps 11-12).

Similar to authorization request signing, an EV can use its PQC contract credentials to sign meter receipts (Step 13). The signed data again includes the eMAID and an ID of the recipient (i.e., the eMSP for, e.g., billing purposes) and is sent to the CP, which may verify the signature using the EV's public contract key (Steps 14-16). The meter receipt is later forwarded to the eMSP, along with other billing-relevant data, and the eMSP verifies the signature (not shown in Fig. 2).

Contract Credential Installation/Update. After the application protocol and signature algorithm negotiations between EV and CP are completed, the EV may choose to install new contract credentials (e.g., if it does not possess valid credential with the chosen algorithm). For this, the HSM generates the key pairs

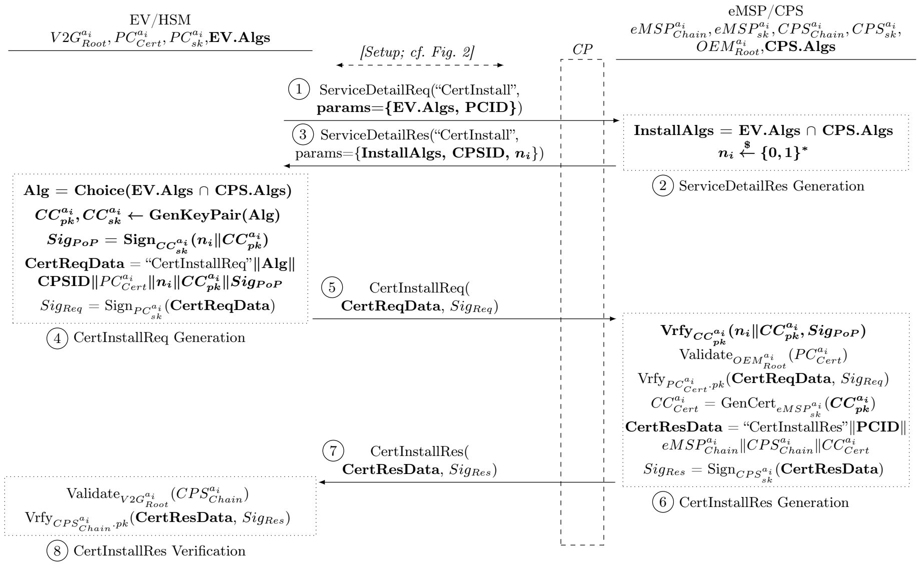
EV/HSM
$V2G^{a_i}_{Root}, PC^{a_i}_{Cert}, PC^{a_i}_{sk}$, **EV.Algs**
[Setup; cf. Fig. 2]
CP
eMSP/CPS
$eMSP^{a_i}_{Chain}, eMSP^{a_i}_{sk}, CPS^{a_i}_{Chain}, CPS^{a_i}_{sk}$,
$OEM^{a_i}_{Root}$, **CPS.Algs**
1 ServiceDetailReq("CertInstall", **params={EV.Algs, PCID}**)
InstallAlgs = EV.Algs ∩ CPS.Algs
$\boldsymbol{n_i \xleftarrow{\$} \{0,1\}^*}$
2 ServiceDetailRes Generation
3 ServiceDetailRes("CertInstall", params={**InstallAlgs, CPSID,** $\boldsymbol{n_i}$})
Alg = Choice(EV.Algs ∩ CPS.Algs)
$\boldsymbol{CC^{a_i}_{pk}, CC^{a_i}_{sk} \leftarrow \mathrm{GenKeyPair(Alg)}}$
$\boldsymbol{Sig_{PoP} = \mathrm{Sign}_{CC^{a_i}_{sk}}(n_i \| CC^{a_i}_{pk})}$
CertReqData = "CertInstallReq"‖**Alg**‖
CPSID‖$PC^{a_i}_{Cert}$‖$\boldsymbol{n_i}$‖$\boldsymbol{CC^{a_i}_{pk}}$‖$\boldsymbol{Sig_{PoP}}$
Sig_{Req} = Sign$_{PC^{a_i}_{sk}}$(**CertReqData**)
4 CertInstallReq Generation
5 CertInstallReq(**CertReqData**, Sig_{Req})
$\boldsymbol{\mathrm{Vrfy}_{CC^{a_i}_{pk}}(n_i \| CC^{a_i}_{pk}, Sig_{PoP})}$
Validate$_{OEM^{a_i}_{Root}}$($PC^{a_i}_{Cert}$)
Vrfy$_{PC^{a_i}_{Cert}.pk}$(**CertReqData**, Sig_{Req})
$CC^{a_i}_{Cert}$ = GenCert$_{eMSP^{a_i}_{sk}}$($\boldsymbol{CC^{a_i}_{pk}}$)
CertResData = "CertInstallRes"‖**PCID**‖
$eMSP^{a_i}_{Chain}$‖$CPS^{a_i}_{Chain}$‖$CC^{a_i}_{Cert}$
Sig_{Res} = Sign$_{CPS^{a_i}_{sk}}$(**CertResData**)
6 CertInstallRes Generation
7 CertInstallRes(**CertResData**, Sig_{Res})
Validate$_{V2G^{a_i}_{Root}}$($CPS^{a_i}_{Chain}$)
Vrfy$_{CPS^{a_i}_{Chain}.pk}$(**CertResData**, Sig_{Res})
8 CertInstallRes Verification

Fig. 3. Installation procedure of a contract credential (changes in **bold**).

$\{CC^{a_i}_{pk}, CC^{a_i}_{sk}\}$ and the EV requests the eMSP to issue the corresponding *contract certificates* $CC^{a_i}_{Cert}$ for the public keys $CC^{a_i}_{pk}$.

Fig. 3 shows the process for installing contract credentials using the modified $CertInstallReq$ message. The process again starts with an algorithm negotiation using modified service detail messages (for the "CertInstall"; Steps 1-3). The EV sends (via the CP) a prioritized list of supported algorithms to the CPS, which acts as a middleman between EVs and eMSPs in the installation process (Fig. 3 combines CPS and eMSP for simplicity). The CPS responds with a selection of accepted algorithms and a fresh nonce n_i. The EV uses its HSM to generate the contract credential key pair $\{CC^{a_i}_{pk}, CC^{a_i}_{sk}\}$, create a Proof-of-Possession (PoP) signature over n_i and $CC^{a_i}_{pk}$ with $CC^{a_i}_{sk}$, and sign the certificate request data using $PC^{a_i}_{sk}$ (Step 4). The credential update process is similar except that an old $CC^{a_i}_{sk}$ is used for signing instead of $PC^{a_i}_{sk}$ (not shown).

The signed certificate request data, which includes the selected algorithm, an ID of the recipient, the $PC^{a_i}_{Cert}$, n_i, $CC^{a_i}_{pk}$, and the PoP signature, is sent (via the CP) to the CPS (Step 5). The CPS verifies the PoP signature based on $CC^{a_i}_{pk}$, validates the provisioning certificate $PC^{a_i}_{Cert}$ based on a locally installed OEM root, verifies the certificate request data signature based on $PC^{a_i}_{Cert}$, and forwards the request to the eMSP (Step 6). Afterwards, the eMSP responds with a contract certificate $CC^{a_i}_{Cert}$ for $CC^{a_i}_{pk}$ based on which the CPS can build and sign the certificate response data (Step 6). The signed response data is sent to the EV, which verifies the CPS signature based on a locally installed root and stores the contract credential for later use (Steps 7-8).

Integration into ISO 15118-20. In case Edition 2 of ISO 15118 is supported, minor changes to the different processes are required. Firstly, the EV manufacturing process now also requires that the EV's key pairs and corresponding vehicle certificates are generated for usage in the TLS handshake. Key pairs are again generated in the EV's HSM and the OEM reads out the public keys and generates the certificates.

Secondly, compared to Edition 1, ISO 15118-20 slightly changes the contract credential usage message flow. The most important change is that *ServiceDetail* messages are sent after authorization and thus cannot be used for algorithm negotiation. Instead, we propose the use of the new *AuthorizationSetup* messages for this purpose, whereby the EV's request (usually empty) contains *EV.Algs* and the CP's response (usually only n_a and some information about the offered services) additionally contains *AuthAlgs* and *CPID*. Furthermore, ISO 15118-20 defines no *PaymentDetails* messages as the CP's nonce n_a is included in the *AuthorizationSetupRes* message and the EV's contract certificate chain is included in its *AuthorizationReq*. Respectively, QuantumCharge would also include the EV's contract certificate chain in its *AuthorizationReq*.

Finally, the contract credential installation process requires minor adjustments. In ISO 15118-20, *ServiceDetail* messages are sent after authorization and, thus, after credential installation. Additionally, the installation of multiple credentials from different eMSPs is supported, whereby multiple *CertInstall* message pairs are exchanged. We thus propose the use of *CertInstall* messages for algorithm negotiation, whereby the first *CertInstall* message pair contains the data of the *ServiceDetail* message pair in Fig. 3. Following *CertInstall* message pairs contain the proposed installation data with no further changes needed.

6 Formal Security Verification

We formally verify RS_6 to RS_9, as we focus on the changes made to the authorization protocol. RS_1, RS_2, and RS_3 are addressed by the usage of an HSM with PQC support. Private keys are stored and used in a secure area of the HSM (addressing RS_1 and RS_2). In addition, we assume that the HSM enforces key usage authorization, e.g., using a policy that enables key access only after a secure boot process (addressing RS_3). We assume that keys stored within the backend, especially Certificate Authority (CA) keys, will be processed at least as securely, for example, using a server HSM and additional physical access control to the processing hardware. Moreover, RS_4 is enabled by our changes to the credential installation process, as private credential keys are no longer generated in the backend but in the EV's HSM instead. Possible downgrade attacks should be addressed by timely disabling insecure algorithms by policies, especially the backwards compatibility with non-PQC algorithms should be disabled when quantum computer-based attacks become feasible for malicious actors. This will require close negotiation between the OEMs, eMSPs, and Charge Point Operators to determine the best security/operability trade-off for the specific ecosystem. We abstract from the used algorithms and assume they satisfy RS_5.

We formally verify our protocol in the symbolic model, also called the Dolev-Yao model [21]. In this model, cryptographic primitives are represented by symbolic functions and assumed to be perfectly secure. Instead, the security of the composition of these primitives is analyzed. Usually, the attacker has complete control over the network (cf. the *Compromised Network* adversary in Sec. 3). However, restrictions of this control, for example, by assuming secure or authenticated channels, are possible. In addition, we allow the presence of multiple dishonest parties, i.e., the attacker can corrupt multiple parties that are not directly involved in the transaction under proof. For example, if Alice sends a message to Bob, this message has to remain secure even if there is a compromised Charlie that also sends messages to Alice and Bob.

We use the Tamarin prover [45] to verify the security of our model, a state-of-the-art tool for automated symbolic protocol verification. In Tamarin, a model is specified as a set of rules that define a protocol's communication and data-processing steps. Security properties that are required to hold over all possible execution traces of the model are specified in first-order logic and called lemmas. Tamarin starts with a state where the property has been violated and performs a backward search over the possible rule executions to determine whether there is a valid path that leads to this state. If there is, the property does not hold, and Tamarin has found a counterexample. If there is no possible execution path that can lead to a violation of the property, this proves that the property holds.

Our complete Tamarin model files for QuantumCharge are provided in an online repository.[9] The files contain lemmas to verify the desired security properties and additional lemmas that verify the correctness of the Tamarin specification of our model. The repository also includes instructions on how to run the model and reproduce and verify the results of our formal analysis and details on the verification times.

We model the authentication requirements from RS_7 to RS_9 with the notion of *injective agreement*. Injective agreement is a well-established and strong authentication property originally defined by Lowe [40] and commonly used in current research (e.g. [11,28,66,39]). It encompasses verification of the identity of the communication partners, the integrity of the exchanged messages, and protection against replay attacks.

Definition 1 (Injective agreement). *A protocol guarantees* injective agreement *to an honest initiator A with an honest responder B on a set of data items* ***ds*** *if, whenever A, acting as initiator, completes a run with the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as a responder in this run. Moreover, each run of A corresponds to a unique run of B and both agents agree on the values of the variables in* ***ds*** *[40].*

[9] `https://code.fbi.h-da.de/seacop/quantumcharge-source`

To encompass crypto-agility (RS_6), we model cryptographic operations to be bound to an algorithm. They can become insecure at any time during the protocol run, which is modeled by a rule that exposes the secret key of a party, allowing the attacker to forge signatures freely. Moreover, parties can revoke support for algorithms they consider insecure, aborting all pending operations using this algorithm. Ideally, we would like to show that *injective agreement* holds for all transactions in our model, given that each insecure algorithm has been revoked before becoming insecure by at least one of the involved parties before being broken by the attacker. However, this property is unrealistically strong since an attack would always be possible against an entity using insecure algorithms. For example, the EV could send a message to the CPS, then the used algorithm becomes insecure and is revoked by the EV but not by the CPS. The message would then be accepted by the CPS regardless. We argue that the actual risk of such an attack is limited, as its possible timeframe is small. However, we can only show a slightly weaker property, requiring that *injective agreement* holds for parties that revoked the algorithm before it became insecure, but not necessarily for their communication partner.

Definition 2 (Injective agreement under insecure algorithms). *Honest A with honest B* injectively agree (cf. Def. 1) *on* ***ds*** *and all cryptographic algorithms used in this protocol run, either directly by A and B or within the setup of credentials they use during this protocol run, either remain secure during the entire protocol run or are revoked by A prior to becoming insecure.*

Using Def. 2, we formally verify that QuantumCharge provides RS_7, RS_8, and RS_9 with the Tamarin prover, under consideration of RS_6 by modeling crypto-agility as previously discussed. For this, all communication of CPs is assumed to be under the control of the adversary (cf. *Compromised CP* in Sec. 3). Entities that are directly involved in a specific protocol run (EV, CPS, or eMSP) are assumed to be *honest*, i.e., we assume that their cryptographic credentials have not been revealed to the adversary. This assumption is reasonable due to the use of HSMs (cf. RS_1, RS_2, and RS_3). However, in order to keep the needed assumptions as weak as possible, other entities of the same types that are not directly involved in the protocol run can be compromised (cf. *Compromised EV* and *Leaked Backend Data* in Sec. 3). Thus, our model can verify that, even if the key storage/usage of some entities is subject to an attack (e.g., due to an implementation error in some HSMs), the security of all other entities remains intact.

The execution of Tamarin with our model[9] shows that QuantumCharge satisfies the security properties, as no counterexamples are found. As the proof generation is fully automated, the following description summarizes the verified properties instead of going into proof details.

Secure credential installation. For the security of the credential installation process (RS_7), we consider a compromised CP and network. Note that, as we always allow for the presence of additional dishonest parties, the presence of compromised EVs trying to impersonate a different EV during the process is also considered. Using our Tamarin model, we verify that *injective agreement under insecure algorithms* holds for credential installation transactions in our protocol between EV and CPS in both directions. That is, the CPS (as initiator) *injectively agrees* with the EV (as responder) on the certificate installation request (Step 5 in Fig. 3 and the EV (as initiator) *injectively agrees* with the CPS (as responder) on the certificate installation response message (Step 7 in Fig. 3). As an example, we look at the first of these properties in detail in Appendix A.1.

Secure PnC authorization process. We verify that the CP and the EV *injectively agree under insecure algorithms* on authorization requests from the EV. That is, the Tamarin analysis shows that QuantumCharge provides strong security for charge authorizations (RS_8). Note that the algorithms used during the setup of the used credentials are assumed to have remained secure during their setup.

Charge Data Authenticity. As charge measurements originate from the CP's meter, it is difficult to prevent the CP from providing false meter data to the eMSP for billing. In ISO 15118, this is addressed by optional metering receipts. The EV may compare the CP's meter data with its own measurements before signing receipts. Our analysis shows that for metering receipts verified by the eMSP, *injectiv agreement under insecure algorithms* on the included data (RS_9) is satisfied between the eMSP and the EV. Thus, the data cannot be manipulated by the CP or a network attacker.

7 Implementation and Practical Feasibility Evaluation

We implemented QuantumCharge as a proof-of-concept. The setup is shown in Fig. 4 and the code is provided online.[9] The EV and CP are implemented on Raspberry Pi 3 Model B+ boards and connected via PLC stamp micro 2 EVBs to emulate Power Line Communication as per ISO 15118. The Pi boards have an Arm-based Quad Core CPU at 1.4 GHz and 1 GB RAM. While there is variance between manufacturers, this setup can nonetheless be argued to be representative since: (i) w.r.t. the CP's side, commercial controllers with comparable performance exist, such as Vector's vSECC [64] (using an Arm-based Quad Core CPU at 1 GHz and 2 GB RAM) and (ii) w.r.t. the EV's side, an option that is discussed in the industry is the offloading of cryptographic functions for PnC to one of the more capable vehicle ECUs such as the infotainment/head unit, which is usually in the Arm Cortex-A performance class [4] (e.g., [62] with a dual Arm Cortex–A15 at 1.2 GHz).

Our ISO 15118 implementation uses RISE V2G [63] in Java, with the changes proposed in Sec. 5. For the PQC algorithms, we use the OQS implementations in C, called from Java via JNI using the provided bindings. Certificates and keys for the PKI are generated using OQS' PQC OpenSSL prototype. Functions of the e-mobility backend (CPS/eMSP) are implemented on the CP-Pi for simplicity.

For each PQC algorithm, all working parameter options supported by OQS are included in our QuantumCharge implementation and evaluation. In the following, we only list the results of one parameter set per algorithm; the full results

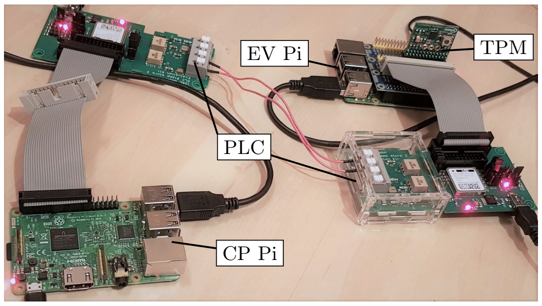

Fig. 4. Proof-of-Concept Setup

Table 1. Communication overhead (in byte).

Message	Algorithm					
	ecdsa256	ecdsa521	eddsa448	dilithium2	falcon512	sphincssha256 128frobust
PaymentDetailsReq Total	1,472	1,880	1,489	12,262	5,679	52,371
$eMSP_{Chain} \| CC_{Cert}$	1,435	1,843	1,452	12,225	5,642	52,331
AuthorizationReq Total	383	452	426	2,735	974	17,417
Sig_{Auth}	71	139	114	2,420	660	17,088
CertInstallReq Total	1,150	1,536	1,229	11,868	5,006	57,781
CC_{pk}	91	158	69	1,312	897	32
Sig_{PoP}	72	139	114	2,420	660	17,088
PC_{Cert}	478	614	489	4,080	1,885	17,449
Sig_{Req}	71	138	114	2,420	659	17,088
CertInstallRes Total	3,313	4,196	3,406	27,261	12,327	122,162
$eMSP_{Chain} \| CC_{Cert}$	1,435	1,843	1,452	12,225	5,642	52,331
CPS_{Chain}	1,422	1,828	1,455	12,228	5,638	52,335
Sig_{Req}	71	139	114	2,420	660	17,088

can be found online.[9] Specifically, we include PQC parameter sets for NIST security level 1, which is comparable to the current security of ISO 15118. The only exception is Dilithium, where we use security level 2, since Dilithium does not provide level 1 parameters.

We use the proof-of-concept implementation to measure the overhead of QuantumCharge in comparison to the default ISO 15118 algorithms. W.r.t. communication overhead, the most significant changes come from the inclusion of PQC public keys and signatures in various messages. Tab. 1 provides an overview of the resulting message sizes and the most significant element sizes.[10]

It shows only minor differences between the non-PQC algorithms. The variance between the PQC algorithms, however, is relatively high, with FALCON

[10] Message size totals are slightly larger than the sum of element sizes since totals represent the size of EXI-encoded XML messages (as sent between EV and CP) and since element sizes that are independent of the algorithm are omitted for simplicity.

Table 2. Storage overhead (in byte).

Message	Algorithm					
	ecdsa256	ecdsa521	eddsa448	dilithium2	falcon512	sphincssha256 128frobust
EV Cryptographic Data Total	3,558	4,709	3,538	36,269	17,569	122,342
V2G Root Certificate	472	608	483	4,074	1,882	17,443
Provisioning Cert. Chain	1,429	1,835	1,461	12,234	5,649	52,341
Provisioning Key Pair	121	223	73	3,870	2,202	115
Contract Certificate Chain	1,415	1,820	1,448	12,221	5,634	52,328
Contract Key Pair	121	223	73	3,870	2,202	115
CP Cryptographic Data Total	2,480	3,259	2,486	24,238	11,589	87,328
V2G Root Certificate	472	608	483	4,074	1,882	17,443
eMSP Root Certificate	468	602	479	4,070	1,876	17,439
CP TLS Certificate Chain	1,419	1,826	1,451	12,224	5,629	52,331
CP TLS Key Pair	121	223	73	3,870	2,202	115

offering the lowest overhead (closest to the non-PQC algorithms) due to its relatively small public keys and signatures. Since ISO 15118 messages use a 4-byte length field, their size is limited to 4,294,967,295 byte, which is not violated by any of the evaluated algorithms.

The overhead for *MeteringReceiptReq* messages is comparable to that of the *AuthorizationReq* since both cases involve the same cryptographic actions (a signature with the private contract key) and is thus not explicitly listed. Additional minor overhead (not detailed in Tab. 1 but part of the totals) is caused by: *(i)* The inclusion of algorithm IDs in different messages (e.g., a 2-byte ID per algorithm similar to TLS). *(ii)* The addition of the CPID in *PaymentDetailsRes* and *AuthorizationReq* messages (39-255 characters [35]). *(iii)* The addition of the CPSID in specific *ServiceDetailRes* and *CertInstallReq* message (e.g., 2 char country code plus 3 char operator ID similar to eMSP IDs [35]).

The storage overhead of QuantumCharge for EV and CP is shown in Tab. 2. The reported sizes for all data elements are based on their DER encoded format. All certificate chains use two Sub-CAs, i.e., the chains are the maximum allowed length of ISO 15118. Regarding storage overhead, our results again show only minor differences between the non-PQC algorithms and a high variance between the PQC ones with relations comparable to that of the communication overhead. Notably, a potential compatibility issue of QuantumCharge is created since ISO 15118, independently of its XML message definitions, limits the maximum size of DER-encoded certificates to 800 bytes. In ISO 15118-20, this limit is increased to 1600 byte. Since none of the evaluated PQC algorithms can meet this limit, any kind of PQC-ready ISO 15118 would need a further increase and thus affect the storage requirements of involved systems.

W.r.t. computational overhead, the most significant changes of QuantumCharge are PQC key pair generation, signing, and signature verification. Timing measurements are repeated 100 times using Java's *System.nanoTime()* and Tab. 3 shows the resulting averages. We see that most PQC algorithms are

Table 3. Computational overhead (in ms rounded to four significant figures).

Message	Algorithm					
	ecdsa256	ecdsa521	eddsa448	dilithium2	falcon512	sphincssha256 128frobust
PaymentDetailsReq Handling (CP)	102.4	325.2	82.67	39.28	20.1	279.5
Validate Certificate Path	99.59	322.2	79.63	35.80	17.06	272.9
AuthorizationReq Generation (EV)	42.39	85.05	51.45	30.44	60.74	1,157
Generate XML Signature	31.23	73.50	39.61	21.39	52.11	1,147
AuthorizationReq Handling (CP)	52.97	129.2	45.80	27.75	22.35	100.7
Verify XML Signature	46.27	122.4	39.24	21.39	16.29	93.59
CertInstallReq Generation (EV)	2,508	2,767	1,113	507.6	693.3	2,192
Generate Key Pair	2,372	2,552	807.1	42.04	230.2	71.18
Generate PoP Signature	29.73	72.66	40.90	12.33	38.46	824.3
Generate XML Signature	41.53	78.44	50.32	29.46	51.41	786.0
CertInstallReq Handling (CP)	190.2	407.6	188.9	235.2	225.0	3,582
Verify PoP Signature	54.81	87.75	27.47	7.396	3.078	48.62
Verify XML Signature	29.11	74.08	30.37	18.07	15.84	64.76
Validate Provisioning Certificate	27.11	64.95	26.66	14.19	8.631	64.86
CertInstallRes Generation	69.94	172.9	94.59	175.4	183.3	3,373
Generate Contract Certificate	16.95	52.92	17.50	19.74	55.89	1,299
Generate XML Signature	18.24	70.70	18.07	16.46	51.59	1,340
CertInstallRes Handling (EV)	208.6	456.5	245.1	143.0	114.2	536.8
Verify XML Signature	60.06	119.9	71.90	41.56	35.54	117.0
Validate Certificate Path	102.7	288.6	123.1	58.57	38.33	316.8

suited for the use case. Notably, Dilithium and FALCON are sometimes even faster than the standard algorithms. However, signature generation with the SPHINCS+ algorithm starting from security level 3 and using the small s-parameter sets (not shown) was too slow to meet ISO 15118's limits (fast f-parameter sets showed no issues). Specifically, *CertInstallReq* generation was with 49 seconds over the relevant ISO 15118 limit of 40 seconds.

The performance evaluation shows that QuantumCharge mostly maintains compatibility with the current ISO 15118 limits and the only general (for all algorithms) issue arises from the larger certificate sizes. Since the respective limit, however, was already increased with ISO 15118-20, we argue that a further increase for PQC support is reasonable. The evaluation shows that for SPHINCS+, it is preferable to use the f-parameter sets instead of the s-parameter sets. Since a variety of PQC algorithms (for all security levels) are compatible with existing time limits, we consider RF_1 met. Further, as QuantumCharge is compatible with ISO 15118's current roles and data flows, we argue that RF_2 is met. Moreover, as our extension supports backwards compatibility, RF_3 is met.

8 Conclusion

In this paper, we propose QuantumCharge, a PQC extension for ISO 15118. We analyze the PnC standards and protocols and show where PQC algorithms are required. QuantumCharge includes concepts for migration, crypto-agility, verifiable security, and the use of PQC-enabled HSMs. As baseline, QuantumCharge implements Dilithium, FALCON, and SPHINCS+ and can be extended easily with other PQC algorithms. With our prototype, we analyze the introduced communication and computational overhead. Our results show that all PQC algorithms require increasing the defined maximum certificate size in both editions of ISO 15118 and that all algorithms can meet the defined timing requirements. Hence, all PQC signature algorithms Dilithium, FALCON, and SPHINCS+ (using the fast f-parameter sets) selected by NIST for standardization can be used in QuantumCharge. Our formal analysis using Tamarin shows the security of our protocol changes. QuantumCharge provides strong security guarantees, user-friendly performance, and high compatibility to ISO 15118 as well as legacy systems, thus paving the way for PQC-secure charging.

Acknowledgements This research work has been partly funded by the German Federal Ministry of Education and Research and the Hessian State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cyber-Security ATHENE.

A Appendix

A.1 Tamarin Lemma for RS_7

In Listing A.1, we give the Tamarin lemma for *injective agreement under secure algorithms* for the CPS. The `Commit_CPS_Install` event in Line 3 denotes that the `CPS` accepted a credential installation request by the client (identified by its `PCID`). The CPS sends accepted requests to the eMSP who generates the client's certificate, including the client's public key `cc_pub`, and sent back to the CPS. The CPS signs the response and sends it to the client (cf. Step 5 to 7 in Fig. 3). We require that for all such commit events, there is a corresponding event `Running_EV_Install`, denoting that the EV identified by `PCID` previously sent a certificate installation request (cf. Step 5 in Fig. 3) for the public contract credential key `cc_pub` (Line 4 to 6). In addition, there must not be an additional commit event for the same public key `cc_pub` by any certificate provisioning service (Lines 7-9). This ensures that the `Running_EV_Install` event corresponds to a unique `Commit_CPS_Install` event, i.e., that no replay attacks are possible. We only require this property to hold if all algorithms used by the involved parties either remain secure during the entire transaction or have been revoked by the CPS before completing the transaction (Lines 10-14). This includes the algorithm used for the provisioning certificate of the EV, as well as the signature testifying its validity, the algorithm used for the contract credentials (`cc_pub`), and the algorithm used by the CPS's certificate chain. Note that the CPS will abort the transaction if it uses revoked algorithms. We model the insecurity of an algorithm the same way as the corruption of an entity, that is, the private key of a credential set for this algorithm and entity is given to the attacker, which is denoted by a `KeyReveal` event.

```
lemma auth_install_inj_agreement_insec_algs_CPS:
" All CPS PCID cc_pub #i.
    Commit_CPS_Install(CPS,PCID,cc_pub) @i
    ==> ( (Ex #j.
        Running_EV_Install(PCID,CPS,cc_pub) @j
        & (#j<#i)
        & not( Ex CPS2 PCID2 #i2.
            Commit_CPS_Install(CPS2,PCID2,cc_pub) @i2
            & not(#i2=#i) ) )
        | (Ex entity alg #kr.
            KeyReveal(entity ,alg) @kr
            & Honest(entity ,alg) @i
            & not(Ex #kr2. E_RevokedAlg(CPS,alg) @kr2
                & kr2<i) ) ) "
```

Listing A.1. Injective Agreement under Insecure Algorithm Lemma in Tamarin.

References

1. Acharya, S., Dvorkin, Y., Karri, R.: Public plug-in electric vehicles+ grid data: Is a new cyberattack vector viable? IEEE Transactions on Smart Grid **11**(6), 5099–5113 (2020)
2. Alnahawi, N., Schmitt, N., Wiesmaier, A., Heinemann, A., Graßmeyer, T.: On the State of Crypto Agility. In: Tagungsband zum 18. Deutschen IT-Sicherheitskongress. vol. 18, pp. 103–126. German Federal Office for Information Security (BSI) (2022)
3. Alnahawi, N., Wiesmaier, A., Grasmeyer, T., Geißler, J., Zeier, A., Bauspieß, P., Heinemann, A.: On the State of Post-Quantum Cryptography Migration. In: INFORMATIK 2021. pp. 907–941. Gesellschaft für Informatik, Bonn (2021)
4. Arm: A Starter's Guide to Arm Processing Power in Automotive (2018), https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/a-starters-guide-to-arm-processing-power-in-automotive
5. Atkins, D.: Requirements for Post-Quantum Cryptography on Embedded Devices in the IoT (2021)
6. Aumasson, J.P., Bernstein, D.J., Beullens, W., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Hülsing, A., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Westerbaan, B.: SPHINCS+ Submission to the NIST post-quantum project, v.3 (2020)
7. Bai, S., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation (2021)
8. Bao, K., Valev, H., Wagner, M., Schmeck, H.: A threat analysis of the vehicle-to-grid charging protocol iso 15118. Computer Science-Research and Development **33**(1-2), 3–12 (2018)
9. Barker, W., Polk, W., Souppaya, M.: Getting ready for post-quantum cryptography: Explore challenges associated with adoption and use of post-quantum cryptographic algorithms. Tech. rep., NIST Publications (2020). https://doi.org/10.6028/NIST.CSWP.05262020-draft

10. Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E.: XML Signature Syntax and Processing Version 1.1. W3C recommendation, World Wide Web Consortium (W3C) (2013)
11. Basin, D., Sasse, R., Toro-Pozo, J.: Card brand mixup attack: Bypassing the {PIN} in {non-Visa} cards by using them for visa transactions. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 179–194 (2021)
12. Basu, K., Soni, D., Nabeel, M., Karri, R.: NIST Post-Quantum Cryptography – A Hardware Evaluation Study. Cryptology ePrint Archive, Report 2019/047 (2019)
13. Bindel, N., McCarthy, S., Rahbari, H., Twardokus, G.: Suitability of 3rd Round Signature Candidates for Vehicle-to-Vehicle Communication – Extended Abstract. 3rd PQC standardization conference, NIST (2021)
14. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehle, D.: CRYSTALS – Kyber: A CCA-Secure Module-Lattice-Based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367 (2018)
15. Bova, Francesco, Goldfarb, Avi, Melko, Roger G.: Commercial applications of quantum computing. EPJ Quantum Technol. **8**(1), 2 (2021)
16. Bürstinghaus-Steinbach, K., Krauß, C., Niederhagen, R., Schneider, M.: Post-Quantum TLS on Embedded Systems: Integrating and Evaluating Kyber and SPHINCS+ with Mbed TLS. In: ACM Asia Conference on Computer and Communications Security. p. 841–852. ASIA CCS '20, ACM (2020)
17. Campos, F., Meyer, M., Sanwald, S., Stöttinger, M., Wang, Y.: Post-quantum cryptography for ECU security use cases. In: 17^{th} escar Europe: embedded security in cars (conference proceedings). Ruhr-Universität Bochum (2019)
18. Chang, Y.A., Chen, M.S., Wu, J.S., Yang, B.Y.: Postquantum SSL/TLS for Embedded Systems. In: IEEE Conference on Service-Oriented Computing and Applications. p. 266–270. IEEE (2014)
19. Crockett, E., Paquin, C., Stebila, D.: Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. Cryptology ePrint Archive, Report 2019/858 (2019)
20. Dang, V.B., Farahmand, F., Andrzejczak, M., Mohajerani, K., Nguyen, D.T., Gaj, K.: Implementation and Benchmarking of Round 2 Candidates in the NIST Post-Quantum Cryptography Standardization Process Using Hardware and Software/Hardware Co-design Approaches. Cryptology ePrint Archive, Report 2020/795 (2020)
21. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on information theory **29**(2), 198–208 (1983)
22. European Telecommunications Standards Institute (ETSI): Migration strategies and recommendations to quantum safe schemes. TR 103 619 V1.1.1 (2020)
23. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU Specification v1.2 (2020)
24. Fritzmann, T., Vith, J., Sepúlveda, J.: Strengthening Post-Quantum Security for Automotive Systems. In: Euromicro Conference on Digital System Design (DSD). pp. 570–576 (2020)
25. Fuchs, A., Kern, D., Krauß, C., Zhdanova, M.: HIP: HSM-Based Identities for Plug-and-Charge. In: Conference on Availability, Reliability and Security. ARES '20, ACM (2020)
26. Fuchs, A., Kern, D., Krauß, C., Zhdanova, M.: TrustEV: Trustworthy Electric Vehicle Charging and Billing. In: ACM/SIGAPP Symposium on Applied Computing SAC 2020. ACM (2020)

27. Fuchs, A., Kern, D., Krauß, C., Zhdanova, M., Heddergott, R.: HIP-20: Integration of Vehicle-HSM-Generated Credentials into Plug-and-Charge Infrastructure. In: Computer Science in Cars Symposium. CSCS '20, ACM (2020)
28. Gazdag, S.L., Grundner-Culemann, S., Guggemos, T., Heider, T., Loebenberger, D.: A formal analysis of ikev2's post-quantum extension. In: Annual Computer Security Applications Conference. p. 91–105. ACSAC '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3485832.3485885, https://doi.org/10.1145/3485832.3485885
29. Gupta, D.S., Ray, S., Singh, T., Kumari, M.: Post-quantum lightweight identity-based two-party authenticated key exchange protocol for Internet of Vehicles with probable security. Computer Communications **181**, 69–79 (2022)
30. Hülsing, A., Rijneveld, J., Schwabe, P.: ARMed SPHINCS - Computing a 41 KB signature in 16 KB of RAM. In: Cheng, C., Chung, K., Persiano, G., Yang, B. (eds.) Public-Key Cryptography - PKC 2016. LNCS, vol. 9614, pp. 446–470. Springer (2016)
31. Hülsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme. RFC 8391 (2018)
32. Hülsing, A., Ning, K.C., Schwabe, P., Weber, F., Zimmermann, P.R.: Post-quantum WireGuard. In: IEEE Symposium on Security and Privacy. pp. 304–321 (2021)
33. ISO/IEC: Road vehicles – vehicle to grid communication interface – part 1: General information and use-case definition. ISO 15118-1:2013, ISO (2013)
34. ISO/IEC: Road vehicles – vehicle-to-grid communication interface – part 2: Network and application protocol requirements. ISO 15118-2:2014, ISO (2014)
35. ISO/IEC: Road vehicles – vehicle to grid communication interface – part 20: 2nd generation network and application protocol requirements. ISO 15118-20:2022, ISO (2022)
36. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: Post-quantum crypto library for the ARM Cortex-M4, https://github.com/mupq/pqm4
37. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. Cryptology ePrint Archive, Report 2019/844 (2019)
38. Kern, D., Krauß, C.: Analysis of e-mobility-based threats to power grid resilience. In: Proceedings of the 5th ACM Computer Science in Cars Symposium. CSCS '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3488904.3493385, https://doi.org/10.1145/3488904.3493385
39. Kern, D., Lauser, T., Krauß, C.: Integrating privacy into the electric vehicle charging architecture. Proceedings on Privacy Enhancing Technologies **3**, 140–158 (2022)
40. Lowe, G.: A hierarchy of authentication specifications. In: Computer Security Foundations Workshop. pp. 31–43. IEEE (1997)
41. Macaulay, T., Henderson, R.: Cryptographic Agility in Practice: Emerging Use-Cases. Infosec Global (2019)
42. Malina, L., Ricci, S., Dzurenda, P., Smekal, D., Hajny, J., Gerlich, T.: Towards Practical Deployment of Post-quantum Cryptography on Constrained Platforms and Hardware-Accelerated Platforms. In: Simion, E., Géraud-Stewart, R. (eds.) Innovative Security Solutions for Information Technology and Communications, LNCS, vol. 12001, pp. 109–124. Springer (2020)
43. Mathilde, R., Aymeric, G., Yolan, R.: PQ-WireGuard: we did it again. https://csrc.nist.gov/CSRC/media/Presentations/pq-wireguard-we-did-it-again/images-media/session-5-raynal-pq-wireguard.pdf (2021)
44. McGrew, D., Curcio, M., Fluhrer, S.: Leighton-Micali Hash-Based Signatures. RFC 8554 (2019)

45. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Computer Aided Verification. LNCS, vol. 8044, pp. 696–701. Springer (2013)
46. Mosca, M., Piani, M.: 2021 Quantum Threat Timeline Report. Tech. rep., Global Risk Institute (2022)
47. NIST: Post-Quantum Cryptography PQC — Selected Algorithms 2022, `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`
48. NIST: Project: Post-Quantum Cryptography (2017), `https://csrc.nist.gov/Projects/post-quantum-cryptography`
49. OCA: Open Charge Point Protocol 2.0.1 - Part 2 - Specification. Open standard, Open Charge Alliance, Arnhem, Netherlands (3 2020), `https://www.openchargealliance.org/protocols/ocpp-201/`
50. Oder, T., Speith, J., Höltgen, K., Güneysu, T.: Towards Practical Microcontroller Implementation of the Signature Scheme Falcon. In: Ding, J., Steinwandt, R. (eds.) Post-Quantum Cryptography, LNCS, vol. 11505, pp. 65–80. Springer (2019)
51. Ott, D., Peikert, C., other workshop participants: Identifying Research Challenges in Post Quantum Cryptography Migration and Cryptographic Agility. arXiv preprint arXiv:1909.07353 (2019)
52. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking Post-Quantum Cryptography in TLS. Cryptology ePrint Archive, Report 2019/1447 (2019)
53. Paul, S., Scheible, P.: Towards Post-Quantum Security for Cyber-Physical Systems: Integrating PQC into Industrial M2M Communication, LNCS, vol. 12309, p. 295–316. Springer (2020)
54. Ravi, P., Sundar, V.K., Chattopadhyay, A., Bhasin, S., Easwaran, A.: Authentication Protocol for Secure Automotive Systems: Benchmarking Post-Quantum Cryptography. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5 (2020)
55. Renesas Electronics Corporation: R-Car Automotive System-on-Chips (SoCs) (2022), `https://www.renesas.com/us/en/products/automotive-products/automotive-system-chips-socs#parametric_options`
56. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comput. **26**(5), 1484–1509 (oct 1997)
57. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Post-Quantum Authentication in TLS 1.3: A Performance Study. In: Network and Distributed System Security Symposium. Internet Society (2020)
58. Smith, M., Castellano, J.: Costs Associated With Non-Residential Electric Vehicle Supply Equipment: Factors to consider in the implementation of electric vehicle charging stations. U.S. Department of Energy Vehicle Technologies Office (2015)
59. Smyslov, V.: Intermediate exchange in the IKEv2 protocol. IETF draft (2021)
60. Stebila, D., Fluhrer, S., Gueron, S.: Hybrid key exchange in TLS 1.3. IETF draft (2020)
61. Stebila, D., Mosca, M.: Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. In: Avanzi, R., Heys, H. (eds.) Selected Areas in Cryptography – SAC 2016. LNCS, vol. 10532, pp. 14–37. Springer (2016), `https://openquantumsafe.org/`
62. Texas Instruments: DRA745 – Infotainment Applications Processor (2019), `https://www.ti.com/product/DRA745`

63. V2G Clarity: Reference Implementation Supporting the Evolution of the Vehicle-2-Grid communication interface (RISE V2G) (2020), https://github.com/V2GClarity/RISE-V2G
64. Vector: vSECC – Communication Controller for High Power Charging Stations V2.3 (2022), https://cdn.vector.com/cms/content/products/vSECC/Docs/vSECC_FactSheet_EN.pdf
65. Wang, W., Stöttinger, M.: Post-Quantum Secure Architectures for Automotive Hardware Secure Modules. Cryptology ePrint Archive, Report 2020/026 (2020)
66. Wesemeyer, S., Newton, C.J., Treharne, H., Chen, L., Sasse, R., Whitefield, J.: Formal analysis and implementation of a tpm 2.0-based direct anonymous attestation scheme. In: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security. p. 784–798. ASIA CCS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3320269.3372197, https://doi.org/10.1145/3320269.3372197
67. Zhdanova, M., Urbansky, J., Hagemeier, A., Zelle, D., Herrmann, I., Höffner, D.: Local power grids at risk – an experimental and simulation-based analysis of attacks on vehicle-to-grid communication. In: Proceedings of the 38th Annual Computer Security Applications Conference. p. 42–55. ACSAC '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3564625.3568136, https://doi.org/10.1145/3564625.3568136