Non-Repudiation and End-to-End Security for Electric-Vehicle Charging

Pol Van Aubel, Erik Poll, and Joost Rijneveld
Radboud University, the Netherlands
{pol.vanaubel,e.poll,j.rijneveld}@cs.ru.nl

***Abstract*—In this paper we propose a cryptographic solution that provides non-repudiation and end-to-end security for the electric-vehicle-charging ecosystem as it exists in the Netherlands. It is designed to provide long-term non-repudiation, while allowing for data deletion in order to comply with the GDPR. To achieve this, we use signatures on hashes of individual data fields instead of on the combination of fields directly, and we use Merkle authentication trees to reduce the overhead involved.**

m				
m_{shared}		m_{CPO}	m_{eMSP}	
EV id	Time	CP Location	Contract id	Rate

Fig. 1. We can view a message from an EV as containing fields used solely by the CPO, fields used solely by the eMSP, and fields used by both. Given here is an example message m with 5 data fields, 2 of which are shared.

I. Introduction

Electric-vehicle charging in the Netherlands requires communication between at least four entities:

- Electric Vehicles (EVs);
- Charge Points (CPs) charging the vehicles;
- Charge Point Operators (CPOs) running the CPs; and
- e-Mobility Service Providers (eMSPs) contracted by the drivers of EVs to provide the energy.

To facilitate this communication, several protocols are used or will be used in the near future. We will look at the combination of three of these:

- ISO15118 [1] between EV and CP;
- OCPP [2] between CP and CPO; and
- OCPI [3] between CPO and eMSP.

These protocols are intended to exchange, among other things, charge data records that describe charge sessions that have taken place, including location and the measurements taken by the electricity meter. They are used by CPOs and eMSPs for billing customers and each other.

The protocols rely on TLS to provide authenticity and secrecy against external (MITM) attackers. We do not propose to replace this mechanism, these security guarantees are important. However, the security guarantees of TLS are also insufficient because:

1) TLS does not provide long-term authenticity or non-repudiation on the data it transported. Therefore, the eMSP has no way to prove that data was generated by the CPO or EV. Similarly, the CPO has no way to prove that data was generated by an eMSP or EV.
2) CPOs act as intermediaries (i.e. proxies) between EV and eMSP. Although they use TLS for communication with the EV, and for communication with the eMSP, they are between two TLS links. They forward data and see that data pass in plaintext. With regards to the example in Figure 1, the CPO should not be able to see the values of Contract id and Rate. On the other hand, it should not forward the CP Location to the eMSP.

To address the first concern, we need non-repudiation: a party needs to be able to prove later that another party generated some data. This means we need some form of asymmetric signature that can be stored and authenticated long-term, for data at rest, and it also provides end-to-end authenticity.

For the second concern, we need end-to-end encryption between CP or EV and the eMSP, using a key which is not known to the proxying CPO.

ISO15118 does provide non-repudiation for some of its messages through the use of XML signatures, and end-to-end-security for its certificate updating mechanism through the use of public-key encryption [4]. However, this mechanism is insufficient. The encryption is not usable for data fields in protocol messages, and the signature structure does not allow for partial data deletion, which we believe is needed to ensure compliance with the General Data Protection Regulation [5].

There exists a fundamental tension between non-repudiation and data minimization, including the right to be forgotten. Data minimization requires that only that personal data which is needed to fulfil a specific purpose is collected, and, once no longer needed to fulfil that purpose, is removed. This means that data must be deleted, anonymized, abstracted, or otherwise made “less personal” over time [5]. However, signatures over data become invalid as soon as the data is changed.

We analyse the tradeoffs of some possible solutions and propose a security architecture that deals with these issues. Our main contribution is the security architecture that provides non-repudiation and secrecy in the presence of proxies, while balancing concerns of eMSPs and CPOs as well as legal concerns with regards to the possibility of data deletion imposed by the GDPR [5]. Our signature scheme allows for data removal. To achieve this, we borrow concepts from Merkle authentication trees [6], and sign the hashes of individual fields

Funding: This work is supported by the EU Regional Development Fund (ERDF), as part of the project Charge & Go.

instead of signing the combination of fields directly.

Section II provides the general signature and secrecy scheme for the protocol ecosystem. Section III provides the steps of protecting a document in detail, suggests existing cryptographic standards to use to implement our solution, and reviews some changes required for the protocols. Finally, we draw some conclusions in Section IV.

II. End-to-End Security Architecture

A complete list of our requirements is:

1) Non-repudiation: data must be authenticated in such a way that it can be proven that a party generated it.
2) End-to-end secrecy: data being forwarded must be hidden from the intermediate parties.
3) The possibility for data minimization: to ensure user privacy, data must be removable once no longer needed. It should be possible to remove individual fields without affecting the validity of signatures for other data fields.
4) The overhead on size of messages should be limited, because OCPP is often run over GPRS links that may be billed per byte.
5) Offline operation: since charge stations can operate offline, the solution must work without an active connection between all parties.

To satisfy these requirements, we need to add signatures for non-repudiation, and authenticated encryption (AE) for end-to-end secrecy. In this section, we first look at how to combine signatures and encryption. Next, we look at the structure of our signatures for achieving non-repudiation, then finally at the AE for end-to-end secrecy.

A. Combining Signatures and Encryption

When combining AE and asymmetric signatures, an important choice is whether to sign the encrypted data (encrypt-then-sign) or to sign the raw data and encrypt afterwards (sign-then-encrypt if the signature itself is also encrypted, or sign-and-encrypt if it is not). We start with this choice because it eliminates several possible architectures. Encrypt-then-sign has several drawbacks, but the most significant one is that in order to verify the signature, the data has to be stored encrypted, along with its corresponding decryption key. We do not want to force long-term retention of ciphertexts for several reasons:

- Either the data is stored both encrypted and decrypted, in which case verification requires verifying the signature on the encrypted data *and* decrypting and checking that the data matches the stored decrypted version; or the data is only stored encrypted, in which case every usage requires decryption.
- Unless every data field is encrypted separately, this scheme will never allow for data minimization: it is not possible to delete part of an authenticated ciphertext; whereas it is possible to come up with a signature scheme that allows for deleting individual plaintexts.

So encrypt-then-sign is disregarded as an option.

The difference between sign-and-encrypt and sign-then-encrypt is whether or not the signature is encrypted together with the plaintext. Both these options provide non-repudiation on the plaintext, but because not encrypting the signature allows us to further reduce the overhead of the scheme, we prefer sign-and-encrypt.

B. Signature solutions

Since we will sign the plaintext data, we can consider possible signature schemes without needing to consider encryption. We will look at four such possible schemes, progressively satisfying more of our requirements, with scheme *4)* being the solution we prefer. Schemes *1)* and *2)* do not satisfy requirement 3 (removability). The reason we describe these regardless is that they are the straightforward ways of providing non-repudiation, and need to be shown to be insufficient here. We will use the example introduced in Figure 1 to clarify how the parts of a message are signed.

1) Sign the entire message m using a single signature: The advantage of this scheme is that it has the absolute minimum of overhead. However, there is a large disadvantage: there is no option of ever removing any data, as signature checks would require the entire message. This violates requirement 3. In fact, it would also mean that eMSPs now would have to receive m_{CPO}, which they currently do not, making this strictly worse than the current situation in terms of data privacy.

2) Two signatures per message: We sign m_{CPO} together with m_{shared}, and m_{eMSP} also together with m_{shared}. Now the CPO only has to forward m_{eMSP} and m_{shared}. This avoids making the situation of data privacy worse. However, deletion of individual fields within m_{CPO}, m_{eMSP}, and m_{shared} is still not possible, and the overhead is greater than with one signature.

3) Two signatures per message over hashes of fields: As above, but instead of signing the combinations m_{CPO} and m_{shared}, and m_{eMSP} and m_{shared} directly, we sign hashes of the individual data fields contained within them. This allows for individual data-field deletion from m_{shared}, m_{CPO}, and m_{eMSP} by simply replacing a data field with its own hash. This does require some attention to ensure that the hash cannot be used to recover the data, which we will describe in section II-D. This still has the overhead of needing to transmit two signatures, however, which can be avoided.

4) One signature per message over hashes of fields: To further lower the overhead of scheme *3)*, instead of creating two separate signatures, we can create a single signature over two hashes: the hash of the collection of hashes of data fields inside m_{CPO} and m_{shared}, and the hash of the collection of hashes of data fields inside m_{eMSP} and m_{shared}. Effectively, we are signing a tree of hashes, with as root node the combination of these two hashes, and at the leaf level the hashes of the data fields. This is similar to a Merkle authentication tree [6][1].

The signature can be validated by anyone who knows these two hashes. They can either arrive at them by hashing values of data fields they know, or be provided with hashes of data

[1]We use a different method of authenticating the root node, we only have two levels in the tree, and we allow a variable number of leaves per node.

fields they are not supposed to know, or even be provided with the hash of a collection of those hashes. So in our example the CPO would send the hash of the collection of hashes of data fields inside m_{CPO} and m_{shared} to the eMSP, so that the eMSP can verify the combined signature. As in scheme *3)*, the techniques from section II-D must be applied to ensure these hashes cannot be used to recover deleted or encrypted data.

The CPO now has to generate and send the hash for the fields that are signed for itself, along with the signature and data intended for the eMSP, but the overhead on the GPRS link between charge point and CPO is minimized.

We provide a full step-by-step description of generating a signature, and the method to deterministically rebuild the signed document, in section III-A.

C. Encryption

Now that we have settled on a signature scheme, we can look at how to provide end-to-end secrecy. Secrecy of data intended for the CPO but not for the eMSP – m_{CPO} in our example from Figure 1 – can be guaranteed by allowing the CPO to simply not send that part to the eMSP. However, the parts of messages that must only be visible to the eMSP – m_{eMSP} in our example – need encryption to guarantee secrecy. Since we assume that the CPs are under the control of the CPO, we can consider the CPO and CP together as a single proxy from the view of the EV or eMSP.

Since the encryption will be short-term, until delivery at the eMSP, we do not need to consider requirement 3. Therefore we can simply encrypt the part of the message that needs secrecy guarantees against the CP and CPO as a single object. This minimizes the required overhead.

D. Preventing hash reversal through brute forcing

The concept of hashing some data to anonymize it is not new. If personal data is hashed, and the original removed, we believe this satisfies the data minimization and removal requirements of the GDPR if it is impossible to reverse this process. Generally, however, with simple hashing we run into the problem that the pre-image space of the hash is too small. E.g., it is easy to generate the hashes of all valid vehicle license plates, or the hashes of all birth dates, or the hashes of all valid credit card numbers, and then simply find the target value among them. In order to prevent this, the hash must be salted with enough random bits. We will use a 128-bit salt.

As long as the original data field exists, the salt for that data field must be stored alongside it. When the data is anonymized, the salted hash is kept, but the data field and salt are removed. Obviously, this means we cannot use the same salt for separate fields in a message. However, we also do not want to use completely random salts for each individual field because all these salts would need to be transmitted along with their fields. Instead, we use two 128-bit seed values: one for the CPO, included in the plaintext part of a message, and one for the eMSP, included in the ciphertext part of a message. The reason for using two separate seeds is because if a single seed was used, the CPO could use that seed to try and recover the plaintext of encrypted fields, and the eMSP could use that seed to try and recover the part of the message that the CPO did not forward to the eMSP.

The way the salt for a data field is generated is by simply running a keyed hash function with the 128-bit seed as key and the data field as input. These salts are then stored, and the seed must be deleted. Finally, to find the hash value for a data field, its salt is used as the key for a keyed hash function, and the data field value as input. Upon data field deletion, the salt is removed as well, and the hash is kept.

III. Implementation

This section first describes all the steps required to protect a message, and then suggests specific cryptographic standards to base the implementation on.

A. Protecting a message

Protecting a message consists of 10 steps:

1) Build documents to be signed from individual data fields: We are protecting a message that consists of several fields and which may have more than one recipient, each of which only needs to know some of the fields. Therefore, each recipient needs to be able to verify the signature using only the fields it sees in plaintext. Although we do not need to duplicate these fields in the message, we do need to build a separate "document" for each recipient containing all those fields. From our example in Figure 1, these would be a document consisting of $\{\text{EV id}, \text{Time}, \text{CP Location}\}$ for the CPO, and a document consisting of $\{\text{EV id}, \text{Time}, \text{Contract id}, \text{Rate}\}$ for the eMSP.

2) Add recipient and signer identifiers: Any signed message that does not include the intended recipient is vulnerable to an attack known as surreptitious forwarding, where the recipient of a signed message may fool a third party into thinking the original signer had intended the message for them. To protect against this, and various related attacks, we always ensure that the identifiers of the intended recipients and the signers of the data are signed as well. Since the documents from the example already contain the EV id (the signing party), they now become $\{\text{CPO id}, \text{EV id}, \text{Time}, \text{CP Location}\}$ and $\{\text{eMSP id}, \text{EV id}, \text{Time}, \text{Contract id}, \text{Rate}\}$.

3) Generate and add a random seed per document: As described in section II-D, to prevent hash reversal, we need a 128-bit seed unique to each recipient. This seed must be generated by a cryptographically secure random number generator. The seed will be signed, and added to the message. Our documents now look like $\{\text{CPO id}, \text{EV id}, \text{Time}, \text{CP Location}, \text{Seed}_{\text{CPO}}\}$ and $\{\text{eMSP id}, \text{EV id}, \text{Time}, \text{Contract id}, \text{Rate}, \text{Seed}_{\text{eMSP}}\}$.

4) Encrypt specific fields: The data fields to be encrypted from the example are the fields contained in m_{eMSP}: Contract id and Rate. The fields in m_{shared} must not be encrypted because they should also be visible to the CPO. The random seed for a recipient computed in the previous step must *always* be one of the encrypted fields if any fields for that particular recipient are encrypted, so $\text{Seed}_{\text{eMSP}}$ has to be encrypted as well. This is to prevent a CPO or other proxy from being able to use the techniques described in section II-D to recover plaintexts. So the encryption here would be $c = E(m_{\text{eMSP}}, \text{Seed}_{\text{eMSP}})$.

5) Add ciphertexts to the documents: To ensure that the signature also links the ciphertexts to the overall message, the ciphertext from the previous step is added to the document for its recipient. So the document for our eMSP becomes $\{\text{eMSP id}, \text{EV id}, \text{Time}, \text{Contract id}, \text{Rate}, \text{Seed}_{\text{eMSP}}, c\}$.

6) Replace field values with hash values: As described in section II, we will sign the hashes of the data to allow for data removal. However, some identifier that signifies the type of the data field should remain. E.g. if the data fields are in key:value format, then only the value is replaced by the hash, so that the meaning a (deleted) field had remains clear.

The hash $H_k(d)$ of a field d is computed as

$$H_k(d) = keyedhash(salt_d, d)$$
$$\text{where} \qquad salt_d = keyedhash(seed, d)$$

The $seed$ was generated in step 3, and differs depending on which party the hash is for: Seed_{CPO} or $\text{Seed}_{\text{eMSP}}$.

The value of d is then replaced by $H_k(d)$. Note we do not have to send these values as part of a message, since they can be recomputed by the receiving party.

Our eMSP document becomes $D_{\text{eMSP}} = \{H_k(\text{eMSP id}), H_k(\text{EV id}), H_k(\text{Time}), H_k(\text{Contract id}), H_k(\text{Rate}), H_k(\text{Seed}_{\text{eMSP}}), H_k(c)\}$. The CPO-document is transformed analogously.

7) Sort on keys: Although the ordering for data fields may not matter in the protocols, hash values are computed on the bytes used to represent the document, for which ordering does matter. Therefore, to ensure that these documents can reliably be rebuilt without having to store the order of fields, we sort the documents lexicographically on keys.

8) Generate authentication tree root from the individual document hashes: Now we compute a hash value for each document. These hashes are computed using a plain, unsalted hash function. These hash values need to be added to the message m if the document contained encrypted fields.

These document hashes are then combined to form an authentication tree root node. They need to have the recipient-identifier as key. So the root node for our example becomes $root = \{\text{CPO id} : H(D_{\text{CPO}}), \text{eMSP id} : H(D_{\text{eMSP}})\}$.

As in step 7, sort the root node to ensure reliable rebuilding.

9) Sort and sign the authentication tree root node: Now, sign the root node: $s = SIGN(root)$.

10) Add the signature, ciphertext, random seeds, and required authentication tree hashes to the message: The signature s must obviously be added to the message. The fields that have to be encrypted can now be replaced by the ciphertext c computed in step 4. This also adds the random seeds for that recipient. The random seeds computed in step 3 that were not already part of ciphertexts, so Seed_{CPO}, are added. Finally, the hashes from the authentication tree root that are computed over data fields that are encrypted must be added so that the CPO can verify the signature; which in this case is $H(D_{\text{eMSP}})$.

Our final message to the CPO is given in Figure 2. The message the CPO sends to the eMSP is given in Figure 3.

B. Signature and encryption format

The implementation of this scheme should be based on the JSON Web Signatures [7] (JWS) and JSON Web Encryption [8] (JWE) standards, using JSON Web Algorithms [9] and JSON Web Key [10]. The main reason for this is that the two main protocols used in the ecosystem, OCPP and OCPI, already use JSON for their message exchange. It is therefore relatively simple to reuse their message definitions when signing and encrypting data.

Another option to consider would be to use XML signatures and encryption, as used by ISO15118 [4]. However, the CPOs and eMSPs involved in the development of OCPP and OCPI have consciously chosen JSON as their message format, with OCPP making the switch away from XML with version 1.6. We therefore believe it is more in line with the direction of the industry to standardize on JSON-based standards.

C. Cryptographic primitives

We limit ourselves to cryptographic primitives that must be supported for the mandatory TLS support in OCPP and OCPI. This means we will use:

- AES-128-GCM for AE, and
- ECDHE on NIST-P-256 curve for key encapsulation.
- SHA256 and HMAC-SHA256 for (keyed) hashing.
- ECDSA on NIST-P-256 curve for digital signatures.

The encryption in step 4 is performed according to JWE [8]. We use Elliptic Curve Diffie-Hellman Ephemeral Static for key agreement (ECDH-ES) (see [8, app. C] for an example), so that requirement 5 (offline operation) is met.

For the $keyedhash$ functions used in step 6, we suggest using HMAC. The values in the document should be BASE64URL-encoded 256-bit output. For the hashing in step 8, we suggest a normal hash on the BASE64URL representation of the JSON document, similar to how signatures are computed in JWS.

The signature in step 10 is performed according to JWS [7]. We suggest using ECDSA on curve P-256 with SHA256 (see [7, app. A.3] for an example).

To add ciphertexts and signatures to documents and messages, we suggest using the compact serializations defined by JWE and JWS. Regardless of serialization used, both the signature and ciphertext will contain a JOSE header. If default algorithms are used, then one could consider removing their corresponding fields from the header transmitted to the recipient. Having the recipient reinstate these fields themselves before decryption and signature verification would save a few bytes in overhead.

D. Changes to the current ecosystem

To verify a signature, a recipient needs to reconstruct its own document, and know the hashes of the other documents. The recipients obviously needs to store either the values of the data fields and their salts, or the corresponding hashes. The random seed must never be stored, as that would imply all subsequently deleted salts could be recovered.

m_{shared}	m_{CPO}	Seed_{CPO}	$E(m_{\text{eMSP}}, \text{Seed}_{\text{eMSP}})$	$H(D_{\text{eMSP}})$	$SIGN(root)$

Fig. 2. The example message m with encrypted m_{eMSP}, seeds, and signature added, as sent to the CPO. Individual fields from m not displayed for brevity.

m_{shared}	$E(m_{\text{eMSP}}, \text{Seed}_{\text{eMSP}})$	$H(D_{\text{CPO}})$	$SIGN(root)$

Fig. 3. The part of the message that the CPO forwards to the eMSP. Note that $H(D_{\text{CPO}})$ replaces $H(D_{\text{eMSP}})$ so that the eMSP can verify the signature.

All protocols need to be extended to allow for signatures, ciphertexts, and random seeds to be transported. There also needs to be some way to define, protocol-independently, which data is signed and encrypted to which recipients. This will also need to include a standardized way to identify recipients and signers, as well as standardized data field keys.

The inclusion of a signed Time in the protocols provides basic replay protection, but business logic is needed to e.g. prevent an EV from having two sessions at the same time.

IV. Conclusions

There are shortcomings of the EV-charging ecosystem, with two serious ones being lack of non-repudiation and lack of end-to-end secrecy. Due to this, CPOs and eMSPs cannot check or prove that a message really originates from a particular party, and do not have a way to verify the integrity of that information in the long term. On top of this, when information is forwarded by an intermediate party such as a CPO or a Clearing House, this information is readable by that party. This is bad for customer privacy and for sensitive corporate information such as the precise billing rate in use.

However, the ecosystem has several constraints when it comes to introducing security measures. We need to minimize the increase in message size, and the protocols are not all built on the JSON. Furthermore, we need the architecture to allow for data minimization and removal in order to be able to comply with the GDPR. It is not trivial to add non-repudiation and end-to-end secrecy to any existing ecosystem, especially under these constraints. We have shown the tradeoffs of possible solutions and described a possible solution using a tree-based signature scheme and encryption. Inevitably, there is a price to be paid: it is impossible to achieve security without any overhead. Our solution adds a signature, seeds, and hashes. An exact analysis of the overhead will be in a future technical report accompanying this paper.

To the best of our knowledge, we are the first to suggest using authentication trees to allow for data removal from collections without invalidating signatures over those collections, in order to achieve GDPR-compliance in a setting where one requires non-repudiation and end-to-end authenticity. This is a general solution that we believe is broadly applicable to a large number of scenarios. In particular, this situation reminds us of blockchain processing, where there is also tension between processing of personal data on the blockchain and the requirements of the GDPR [11]. One solution there is off-chain processing, where the hash of a document is the only thing that resides on the blockchain. Our solution would be equally applicable there; instead of using the hash of an entire document, the root hash of a tree could be used.

For key distribution, our solution requires a PKI, but for this we hope to reuse the ISO15118 PKI required for that protocol. However, the restrictions ISO15118 places on its PKI may not be compatible with our use – or, for that matter, with use in OCPP and OCPI. Whether this is the case, and what changes to the PKI would be necessary to facilitate this, are the subject of future research.

We observe that the protocols we have seen are now including TLS requirements to secure data in transit. Although a necessary first step, we stress that protocols should also consider securing data at the application layer, to allow for secure data forwarding, and long-term guarantees for data at rest. To achieve this using our solution, protocols also need to be extended to allow for signatures, ciphertexts, and random seeds to be transported. Furthermore, it requires a protocol-independent catalogue of data field keys, ways of identifying parties, and determination of of what data should be used by which party, defining what data should be signed or encrypted to which recipients.

The examples we used are based on a message generated by an EV or CP, proxied by the CPO, to an eMSP. However, the same solution can be used in other scenarios, such as when there are multiple intermediate parties. The aforementioned technical report will properly cover these other cases.

References

[1] “ISO 15118-1:2013 — road vehicles – vehicle to grid communication interface – part 1: General information and use-case definition,” ISO Standard. [Online]. Available: https://www.iso.org/standard/55365.html

[2] “Open Charge Point Protocol,” Open Charge Alliance, Protocol Specification. [Online]. Available: https://www.openchargealliance.org/protocols/ocpp-20/

[3] “Open Charge Point Interface,” Nationaal Kennisplatform Laadinfrastructuur Nederland, Protocol Specification. [Online]. Available: https://github.com/ocpi/ocpi

[4] “ISO 15118-2:2014 — road vehicles – vehicle-to-grid communication interface – part 2: Network and application protocol requirements,” ISO Standard. [Online]. Available: https://www.iso.org/standard/55366.html

[5] European Commission, “General data protection regulation,” 2016.

[6] R. C. Merkle, “Secrecy, authentication, and public key systems,” Ph. D. Thesis, Stanford University, Tech. Rep. 1979-1, June 1979.

[7] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Signature (JWS),” Internet Requests for Comments, RFC 7515, May 2015. [Online]. Available: https://ietf.org/rfc/rfc7515.txt

[8] M. Jones and J. Hildebrand, “JSON Web Encryption (JWE),” Internet Requests for Comments, RFC 7516, May 2015. [Online]. Available: https://ietf.org/rfc/rfc7516.txt

[9] M. Jones, “JSON Web Algorithms (JWA),” Internet Requests for Comments, RFC 7518, May 2015. [Online]. Available: https://ietf.org/rfc/rfc7518.txt

[10] ——, “JSON Web Key (JWK),” Internet Requests for Comments, RFC 7517, May 2015. [Online]. Available: https://ietf.org/rfc/rfc7517.txt

[11] S. van Heukelom, J. Naves, and M. van Graafeiland, “Juridische aspecten van blockchain,” Pels Rijcken & Droogleever Fortuijn, Whitepaper, nov 2017. [Online]. Available: https://www.pelsrijcken.nl/actueel/publicaties/whitepaper-juridische-aspecten-van-blockchain/