Direct Anonymous Attestation on the Road: Efficient and Privacy-Preserving Revocation in C-ITS

ABSTRACT

Vehicular networks rely on public key infrastructure (PKI) to generate long-term and short-term pseudonyms that protect vehicle's privacy. Instead of relying on a complex and centralized ecosystem of PKI entities, a more scalable solution is to rely on Direct Anonymous Attestation (DAA) and the use of Trusted Computing elements. In particular, revocation based on DAA is very attractive in terms of efficiency and privacy: it does not require the use of CRLs and revocation authority can exclude misbehaving participants from a V2X system without resolving (i.e. learning) their long-term identity. In this paper we design a novel revocation protocol based on DAA and show a detailed design and modeling of the implementation on a real TPM platform in order to demonstrate its significant performance improvements compared to existing solutions.

KEYWORDS

Security, Privacy, Direct Anonymous Attestation, Trusted Platform Module (TPM), Revocation, C-ITS, V2X

1 INTRODUCTION

Connected vehicles, as part of the emerging Cooperative Intelligent Transportation Systems (C-ITS) are positioned to transform the future of mobility. This change is enabled by the exchange of messages between vehicles (V2V) and between vehicles and transport infrastructure (V2I), comprising together the overall vehicular communication (V2X). V2X communication systems are expected to greatly improve road safety and traffic efficiency while better supporting autonomous driving. V2X can also save lives by providing road hazard warnings to the driver and reducing collisions [4].

However, despite their benefits, privacy is a key concern in this facet of C-ITS, since the involved vehicle transmissions can be used to infringe the users' location privacy [28]. Many V2X applications rely on broadcasting continuous and detailed location information, as for example through the Cooperative Awareness Messages (CAM), which are broadcasted unencrypted by vehicles at the frequency of 10 Hz. If this information is misused (all exchanged messages can be eavesdropped within radio range) can lead to the extraction of detailed location profiles of vehicles and path tracking [9]. Since there is usually a strong correlation between a vehicle and its owner [11], location traces of vehicles have the potential to reveal the movement and activities of their drivers.

Addressing this challenge, current approaches are based on PKI-based solutions [10] with privacy-friendly authentication services through the use of short-term anonymous credentials, i.e., *pseudonyms* [19]. The common denominator in such architectures is the existence of trusted (centralized) infrastructure entities for the support of services such as authenticated vehicle registration, pseudonym provision, revocation, etc. The location privacy is protected by requiring that each vehicle uses multiple pseudonyms, changing frequently from one pseudonym to another [10].

Use of changing pseudonyms can be considered the state-of-the-art in VANET privacy enhancing technologies like the one that was recently proposed in [26]. Prominent solutions include the Security Credential Management System (SCMS) [26], which is a product of vehicle OEM consortia and the US Department of Transport (USDOT), and the Cooperative-ITS Certificate Management System (CCMS) developed by the European Committee for Standardisation (CEN) and European Telecommunications Standards Institute (ETSI), with support from the European Commission [5].

These architectures have several inherent drawbacks stemming from the fact that they are based on a complex and centralised ecosystem of PKI entities, which we need to trust for issuing and distributing pseudonym certificates. First, a technical and organisational separation of capabilities between the PKI entities is required to cope with internal attackers, resulting in a very costly solution to implement in practice. The bottleneck of having to connect to the back-end infrastructure to acquire pseudonym certificates is resolved by downloading a larger pseudonym pool size, which then provides less protection against Sybil attacks. In the context of revocation policies for removing misbehaving nodes from the network, they are based on dissemination of CRLs, which is an inefficient solution in terms of computational and communication overhead.

To address the aforementioned challenges of centralised PKI solutions, several researchers have suggested moving towards a decentralised approach, where trust is shifted from the back-end infrastructure to the edge [8, 23]. In its recent white paper on privacy-by-design aspects of C-V2X, 5GAA is also pointing to the need of more scalable and decentralised solutions eliminating the need for trust built around "federated infrastructures" [1].

As it has been shown by recent work, one way to do this is by leveraging the use of Direct Anonymous Attestation (DAA) and the incorporation of trusted computing technologies [8, 13, 23]. DAA, originally introduced by Brickell, Camenisch and Chen [2], is a cryptographic protocol designed primarily to enhance user privacy within the remote attestation process of computing platforms, which has been adopted by the Trusted Computing Group (TCG) [21], in its latest specification.

Applying the DAA protocols for securing V2X communication results in the redundancy (and removal) of most of the PKI infrastructure entities, including the pseudonym certificate authority: vehicles can now create their own pseudonym certificates using an in-vehicle trusted computing component (TC), and DAA signatures are used to self-certify each such credential that is verifiable by all verifiers. Furthermore, a DAA-based model supports a more efficient revocation of misbehaving vehicles that doesn't require the use of CRLs, removing therefore all the computational and communication overhead that comes with it. Instead, when the Revocation

Authority issues a revocation request, this triggers the TC of the misbehaving vehicle to delete all of its pseudonym certificates and cryptographic key pair, thus, rendering the TC unable to generate new pseudonyms in the future. However, the details of this process has not been shown and demonstrated so far and its feasibility remains an open question.

In this paper, we are providing a novel revocation protocol for C-ITS based on DAA that leverages the benefits of trusted computing to offer an efficient and privacy-respecting solution. Our protocol provides both revocation of the vehicle from the system and revocation of specific pseudonyms that cannot be reused again (referred to as hard and soft revocation respectively). We provide an implementation of the protocol in a real TPM, in order to demonstrate that the proposed solution is applicable to the real-world and can meet the strict performance requirements, as documented in ETSI standards, and also verify that near-constant revocation time is achievable even when multiple pseudonyms are entailed.

The rest of the paper is organized as follows. Section 2 discusses related work and provides and analytical evaluation of what benefits our solution provides. Then, Section 3 presents a conceptual overview of our solution, while in Section 4 we present the details of the protocol's design and modeling. In Section 5 we evaluate our protocol experimentally and in Section **??** we discuss the trust assumptions. Section 6 concludes the paper.

2 MOTIVATION AND CONTRIBUTION

Revocation is a standard consideration for any C-ITS system. In case of a misbehaving vehicle, the wrongdoer can be evicted and be prevented from further participation. In the case of PKI-based solutions, the revocation can be done in standardised ways by adding the revoked certificates to a Certificate Revocation List (CRL), which is then published by the CA responsible for that trust domain. However, for vehicles using short-lived pseudonym certificates, things are more complicated. If a vehicle possesses multiple certificates that are unlinkable, every single certificate needs to be put on the CRL, which would increase the bandwidth requirement to unfeasible levels. Nowatkowski et al. [18] have shown that the CRL list may grow as much as 2.2 GB, depending on the policy for the number of pseudonyms on vehicles.

There are several approaches in the bibliography that try to address the problem with the size of CRLs. First, CCMS takes the approach not to revoke pseudonym certificates, but instead revoke only the long-term identity of the vehicle [5] to prevent it from acquiring new ones. SCMS includes a linkage value to pseudonym certificates derived from cryptographic seed material [26]. Publication of the seed is sufficient to revoke all certificates belonging to the revoked vehicle. However this requires the addition of two new entities in the architecture, called Linkage Authorities (LAs), with corresponding technical and organizational guarantees to operate separately. Alternative solutions leverage encrypted pseudonyms during the provisioning process [17, 20] so that a vehicle can only decrypt pseudonyms after receiving the encryption keys.

In addition to the efficiency problems of the above revocation mechanisms, they all require the resolution of vehicle's long-term identity from their pseudonyms, in order to work. Even though this can be performed typically only by a dedicated authority, it still poses a privacy threat and require that this resolution authority is fully trusted and won't misbehave or get compromised.

In order to address these shortcomings of PKI-based solutions, there is an increasing effort by researchers to apply anonymous credentials as a solution for privacy-respecting V2X communication. For example, Föster et al. presented PUCA [7] a pseudonym scheme that allows vehicles to use anonymous credentials for authentication with the PCA when obtaining new pseudonyms in the existing European ITS system, changing only the pseudonym issuance phase. PUCA foresees no way of credential revocation. The REWIRE V2X revocation protocol [6] uses trusted computing to enable revocation without pseudonym resolution. An enhanced variant was presented in O-TOKEN [24] where an additional key pair is embedded into pseudonym certificates. However, in both schemes, there are inherent trust assumptions been made on the *correctness* of each vehicle Electronic Unit (ECU) (either configuration or behavioural execution correctness) that limits their feasibility and applicability in real-world environments. More specifically, they have not considered an enhanced threat model where malicious and/or compromised ECUs can monitor and modify all interactions between the host and the attached Trusted Component (TC); i.e., using the TC as an "*oracle*" that can interact with for executing sensitive crypto operations in order to bypass the revocation.

Whitefield et al. [23] first applied DAA to the V2X case and showed how to enable vehicles manage their own pseudonym certificates. Hicks et al. [13] also proposed a scheme that leverages the decentralized properties of DAA towards enabling a secure and privacy-preserving revocation coupled with strong vehicle authentication. However, the proposed architecture relies on a number of infrastructure entities that are not aligned with the current ETSI standards; for instance, it is not clear whether the inclusion of an Authorization Authority (AA) is better for privacy-preserving revocation against what has been proposed in PKI-based systems. Finally, Kumar et. al. [16] proposed the use of DAA for verifier-local revocation in online subscription systems, however, their scheme is rather inefficient as it is linear to the size of the revocation list and not constant.

2.1 Overview of Revocation using DAA

Figure 1 introduces the typical DAA [2] pseudonym life-cycle architecture [23]. As we can see, only two trusted third parties are needed; (i) the Issuer who is responsible for authenticating vehicles through the JOIN protocol (Step 0) and (ii) the Revocation Authority (RA) that shuns out misbehaving vehicles from the ITS within the revocation domain that is managing. In our context, vehicles are the combination of a *host*, that is a vehicular on-board computer, and a *trusted computing component* (TC) that executes in the "secure world"; together they form the platform which we refer to from this point on-wards as the vehicle.

Using DAA, the trusted computing component (TC) in the vehicle is responsible for creating the pseudonym certificates without involving any infrastructure component from the back-end (Step 1 - DAA CREATE). However, the vehicle cannot use pseudonyms unless it has been registered with the RA (Step 2 - DAA JOIN). The registration consists of providing the RA with unique values that can be used later to revoke the key (DAA SETUP). We call these values *revocation hashes*. Upon such a registration, the vehicle will

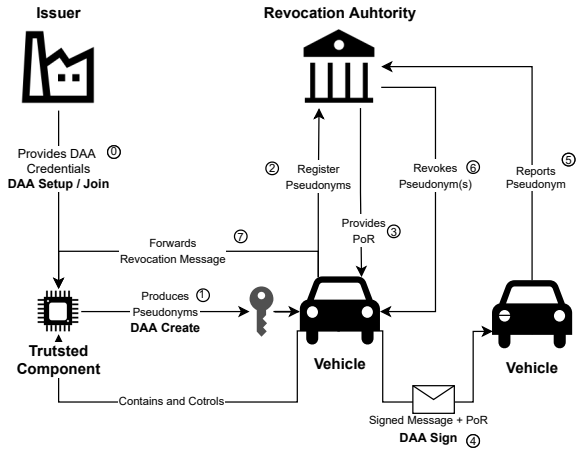


Figure 1: Conceptual overview

receive Proof of Registration (PoR) (Step 3), which is sent along with a signed message (Step 4). Without PoR, any vehicle should disregard the message, as the RA would not be able to revoke such a key. In case a vehicle suspects malicious behaviour of a vehicle, it reports the corresponding pseudonym to the RA (Step 5). Let's assume a number of reports containing a misbehaving vehicle's pseudonym have been already issued to the RA, and the decision to revoke the vehicle has been made based on strong evidence.

The RA does not perform any pseudonym resolution to discover the identity of the misbehaving vehicle. Instead, it initiates the revocation protocol by creating a signed revocation message using its secret key and broadcasts this to all vehicles containing the public pseudonym key that needs to be revoked (Step 6). All vehicles receive the revocation message and their hosts are required to forward them to their corresponding TCs (Step 7). It is the TC of the revoked vehicle which is responsible for deleting the pseudonym certificates and no longer use them.

To be more precise, we differentiate between two kind of revocations: soft- and hard-revocation. Soft revocation means the RA revokes a specific pseudonym, that was used for signing the message based on which the misbehavior policy violation was detected, while hard revocation means revoking all pseudonyms associated to a specific vehicle, thus, not allowing it to further participate in the overall system as an authenticate participant. These two revocation variants essentially reflect the current need for *message-based* and *identity-based* revocation [13]: The first scenario might be triggered when revocation needs to occur due to a technical defect of a vehicle due to a malfunctioning sensor (thus, we want to temporarily revoke his ability to participate in the system by not allowing it to re-use this specific pseudonym). The second case mainly deals with malicious attackers who need to be barred from communication as soon as possible.

Besides enhanced revocation models, with minimal trust assumptions on a vehicle host (Appendix A.3), such solutions need to also be rather efficient and to not pose additional performance overhead, compared to the existing schemes. For instance, in the case of PKIs, pseudonym resolution is required which assumes the interaction between a number of trusted back-end infrastructure entities: These protocols usually require the RA to inquire the Pseudonym Certification Authority for the vehicle identifier that requested a specific pseudonym, P_s. Then, the RA needs to ask the Identity Provider to retrieve all the identifiers issued for this vehicle so that the PCA can then identify all pseudonyms been provided to this specific vehicle. Extensive research in academia [10] has showcased the scalabaility and performance issues with such architectures since even for a small number of pseudonyms (e.g., around 100 pseudonyms) this process requires an additional 2-3 seconds (excluding the network latency and bandwidth requirements).

In the case of other DAA-based solutions, revocation is an inefficient process which is linear in the size of the revocation list [16]. Specifically, the signature-based revocation that allows for the credentials of misbehaving vehicles to be revoked introduces a significant computational overhead. For example, using the ECC-DAA [3] scheme on a 192 bit curve and a blacklist with 200 revoked signatures, attestation takes 24.4 seconds on an ARM11 host [27]. The corresponding proof verification takes 1.4 seconds to verify on a standard desktop PC [13].

Table 1: Notation used

Symbol	Description
$EK^{\dagger}$	Endorsement Key
$AK^{\dagger}$	Authorization Key
$AK_{old}{}^{\dagger}$	Authorization Key from previous iteration
$RAK^{\dagger}$	Revocation Authority Key
$EpK\dagger$	Ephemeral Key
KH_x	Key Handle identifying a loaded key (x) in the Tc.
$E_{\texttt{TMP}}, A_{\texttt{TMP}}$	Key Creation Template EpK and AK respectfully
P_d	A digest representing a governing policy
r_{index}	A non-volatile index in Tc containing 64 bits.
ACI	A non-volatile index in Tc containing 8 bits.
r_{bit}	Revocation bit(s).
$A_{\texttt{LIMIT}}$	Max number of correct authorizations for ACI
$A_{\texttt{COUNT}}$	Current number of correct authorizations for ACI
ACI_{old}	An instance if ACI from previous iteration
σ_x	Cryptographic signature
H	A hash output (digest)
λ	A set of hard revocation and soft revocation policy (leaf digests, l)
β	A compound policy (of branch digests, b) to satisfy to allow revocation.
$\mathcal{P}$	A set of β plus initial write policy.
$P_{\texttt{Auth}}$	Final policy digest (P_d) of all data in $\mathcal{P}$ signed by AK
t	A ticket generated by Tc proving verification of a signature.

† An asymmetric keypair, containing both public and a private key, denoted xK_{pub} and xK_{priv}.

3 CONCEPTUAL PROTOCOL OVERVIEW

In this section we make a high level overview of our protocol, showing how we can implement policy regulations for governing the pseudonyms using the functionality of the Trusted Platform Module (TPM) been used as the trusted component (due to space limitation the building blockas of a TPM are described in Appendix A.1). More specifically, we present in the following section how we utilize an internal, tamper-proof register of the TPM, where each bit represents the state of a pseudoRnym. Creating such a register, or index, in a way that remains tamper-proof for even the host requires deep analysis of the TPM and its internal functions.

In the remainder of this paper, the symbols and abbreviations depicted in Table 1 are adopted.

3.1 Soft- and Hard Revocation

We call the tamper-proof index of the TPM *Revocation Index*. It has 64 bits and as we mentioned above, each bit represents the state of a pseudonym, i.e. a set bit means revoked otherwise the pseudonym is not revoked. So now, revocation of a single pseudonym, namely soft revocation, is simple. As we trust the TC managing the keys, it will be asked to set the key's respective revocation bit to a revoked state. We will use one of the 64 bits to be able to revoke all pseudonyms in one go, i.e., performing hard revocation. That means, the DAA key is linked to the first bit of the Revocation Index, while the pseudonyms are linked to one of the other 63 bits, as well as the first (hard revocation) bit.

We must accommodate the possibility that different authorities govern different areas of the vehicular network, i.e., an RA in one domain should not be allowed to revoke pseudonyms linked to another domain. Therefore, we must protect each bit in the revocation index, allowing only predetermined RA's to execute a revocation process. We propose doing this by building policies for each of the pseudonyms, representing the command being executed (set bit) with particular parameters (which bit). A particular RA must sign these to authorize a revocation. We call them *revocation hashes* and each pseudonym has two: one for soft revocation and one for hard revocation. These revocation hashes are registered with the RA, who must sign one before using it in revocation. It should now be clear that the hard revocation hash must include both the hard-revocation bit and the pseudonyms' unique soft-revocation bit. If this is not the case, the hard-revocation hash for all linked pseudonyms would be equal, as they are to be signed by the same RA and setting the same bit. By including the soft-revocation bit, we "blind" the revocation hash.

Pseudonym limitations. Since the size of the Revocation Index is 64 bits and given we need one for the hard revocation, we can revoke only 63 pseudonyms with this index, which is not enough to cover the requirements of vehicular applications. A naive approach to support more pseudonyms would be to create more revocation indexes and having a new DAA key for each one, following the same principle. However, this poses two distinct problems:e first, the DAA Setup and Join phases are very time-consuming and require communication with the Issuer. Secondly, only 63 pseudonyms can be linked together in the TPM, making hard revocation of a larger set impossible. Therefore, we create multiple revocation indexes using all of their bits for soft revocation and we maintain only one hard revocation bit to be the one defined in the initial revocation index. So all pseudonyms share the same hard revocation bit, meaning hard revocation hashes would be equal for all pseudonyms having soft revocation bits in other indexes. An example of this is shown in Figure 2, where all pseudonyms linked to the DAA key share a common hard revocation bit (first bit of r_1), while their corresponding soft revocation bits span several revocation indexes.

With the current implementation of the TPM, it is not possible to set multiple bits in different indexes, which means they would have to be executed as two commands, removing the "blinding" of the hard revocation hash. To combat this challenge, we propose that the hard-revocation hash represents a command that sets the hard revocation bit and any other unique combination of bits in the index, allowing for 2^{63} pseudonyms with a unique hard-revocation

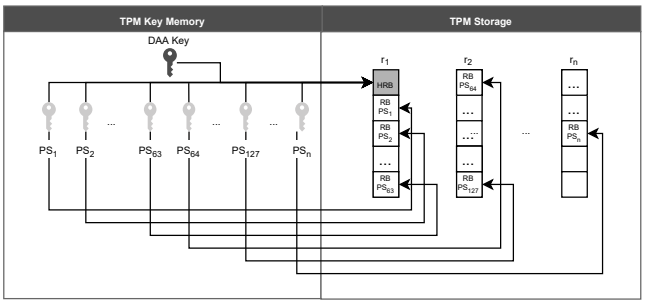


Figure 2: Pseudonyms in TPM, linked to different indexes that shares the same hard revocation bit

index. This requires that the revocation index's bits are revocable only by a single RA; otherwise, an anonymizing mask could cause the unintentional revocation of keys linked to another revocation domain. To support multiple RA domains in a single index, the number of linked pseudonyms is limited to 2^n where n represents the available bit space for each pseudonym set.

3.2 Building the protocol

It should now be evident that we protect the revocation index with a set of policies. This, however, raises an interesting issue. When we write a policy that determines what parameters must be used, a so-called *command parameter hash* must include the name[1] of the entity it applies to - in this case, the index. As the policy is intended *for* the index, it is included in the public area of said index. The index's name depends on the policy, and the policy depends on the index: we have a hash loop without an end, depicted in Figure **??**.

To avoid the hash loop, we use a different approach[2] where a unique key authorizes the policy, and we call this key the Authorization Key (AK) (see Figure 3). The policy is now that *any* policy signed by the AK is a valid policy. Now, the revocation index's actual policy digest is the name of the authorizing key, and to satisfy that policy, one must have the policy signed (authorized) by the AK. The index and revocation policies, therefore, are no longer coupled together. However, this approach raises another difficulty: we must guarantee that the AK can only sign a single policy. If not, the host can sign any policy to comply with and control the revocation index.

To address this challenge, we protect the authorization key by yet another policy. This policy must dictate how many times the host can use the key. We do this by creating another index called the Authorization Counter Index (ACI) and giving it a PIN or password as an authorization value. The ACI contains two parts: Authorization Counter and Authorization Limit. We then use this index to create the authorization key policy, using `PolicySecret`. This policy states that we must prove that we know the ACI's secret by authenticating with it. Every time the password is used for this index, the internal counter increments. When the counter reaches the limit, it fails. With this index, we can define the Authorization Key policy as the correct authentication with the ACI, meaning we must

[1]Name is a hash of the public parameters of an entity, including its policy.
[2]This was identified as a solution in collaboration with the TPM working group of TCG

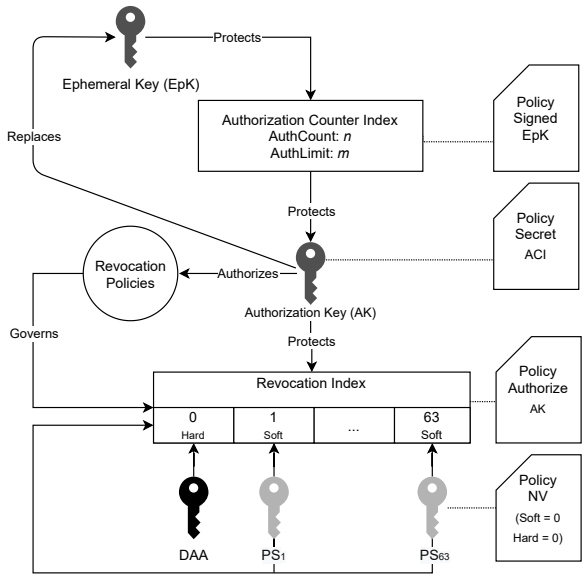


Figure 3: Protocol functionality and lifetime, as seen from the Trusted Component

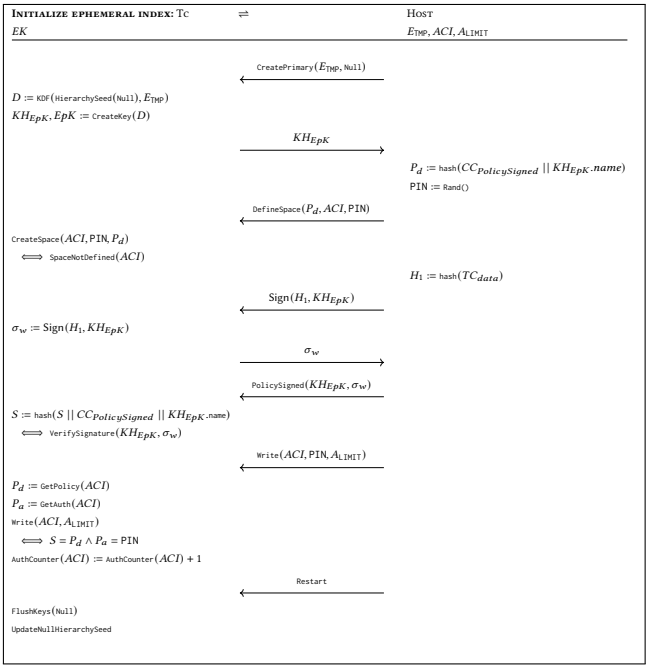


Figure 4: Initialize Primordial Authorization Counter Index

prove we know the password to that index, thereby incrementing the counter and limiting the use of the key to the authorization limit. Now the host can still re-create the ACI and thereby resetting the counter, inferring additional authorizations. To combat this challenge, we protect writing to the ACI using a policy that requires a signature from an ephemeral key - that is, a key that only exists during the initial setup, thereby making the ACI immutable. A straightforward way is to create the ephemeral key in TC's NULL hierarchy, use it, and reboot. Because the keys are generated from a hierarchy seed, and the NULL hierarchy seed is reset on reboot, the host cannot recreate the key.

Looking over this set of required tasks, it becomes an issue that the reboot is necessary for each revocation index created. It is neither efficient nor safe to reboot the Electronic Control Unit (ECU) after 64 pseudonyms have been used. One solution is to give the Authorization Key one additional authorization and use this as the ephemeral key for the next ACI. This has the same effect as a reboot and will render the ACI immutable after the initial write.

Now we have the foundation where we have successfully initialized both the ACI and the revocation index. Before the host can start using the keys, it must initiate the revocation index by writing to it; otherwise, it is unusable. In order to make sure that this initial write-operation can only happen once, we will give the authorization key an additional authorization and, during activation of the index, sign a digest allowing the initial write. Since the authorization key will run out of authorizations, the host will not recreate the index. The host has no incentive to misuse this additional authorization, as no other than the discussed operation will allow the keys to work.

4 ARCHITECTURAL DETAILS & PROTOCOLS

In this section, we look into the protocol's design and modeling and outline the commands and functionality needed for the protocol to hold. We showcase a TPM as a Trusted Component and use the features and commands from the standard as we implement the phases described.

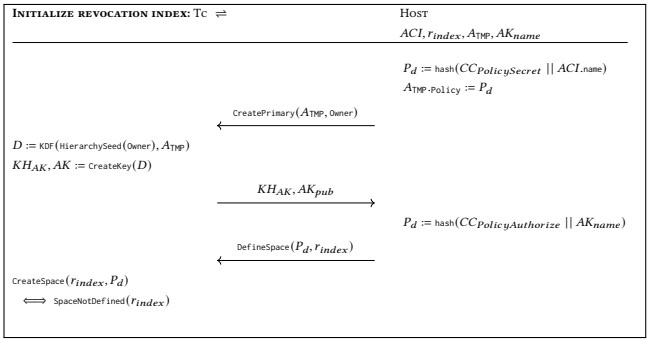
4.1 Authorization Counter Index

The initial phase of creating the Primordial Authorization Counter Index can be seen in Figure 4. As it can be seen, the host must provide an index identifier, template, and authorization limit. As a potentially untrusted host passes much information, this underlines why a trusted entity should verify the index. The first thing to be done is to create the Ephemeral Key in the NULL hierarchy, as discussed previously. This can be certified by a certification process that documents the key and TC's nature and the current boot count.

To create the index, the host creates a policy based on the Ephemeral Key and instructs the TC to define the space with a random secret. The secret is essentially a password, but here we only count how many times it has been used. Hence there is no need to keep it, in fact, secret.

The policy defined earlier must be complied with to write the authorization limit to the index, so the host uses the Ephemeral Key to sign a nonce. After complying with the policy, writing the authorization count, and inherently incrementing the authorization count, the key should be rendered inoperable by executing a reset. After the reset has been completed, the index can be certified by the endorsement key. Such a certificate can be used to prove that the index has a specific authorization limit, and the current boot count proves the reset has been executed. After the reboot, the ACI is

Figure 5: Initializing Revocation Index



guaranteed immutable and is prepared to act as a guard for limiting the number of times the upcoming authorization key can be used.

4.2 Revocation Index

Before initializing the revocation index, we need to create the authorization key and link it to the ACI. As seen in the first line in Figure 5, the host builds a `PolicySecret` policy based on the ACI name, meaning to use the key, the host needs to provide the secret for the ACI, incrementing the authorization counter. This policy is embedded in the template for the key, and in the future, only this template with this specific policy will allow recreating the correct authorization key.

As with other entities, the created key can be certified and signed by the Endorsement Key for later verification. To create the index, the host calculates a new policy digest that links the index's use to the authorization key using `PolicyAuthorize`, meaning any policy signed by the authorization key is valid. As with the key, the index can be certified to allow for verification. The index is now built within the trusted component, and its policy depends on what the authorization key signs in the next phase.

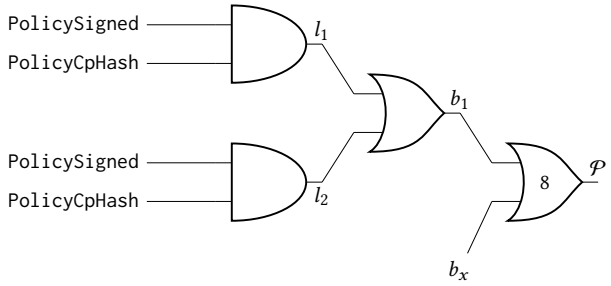
4.3 Generate Final Policy Digest

The policy digest to be authorized by the authorization key is calculated as described in Algorithm 1. Recall that the policy is a compound policy built by logical `AND` and `OR` statements. We can visualize the logical composition as in Figure 6, where we see two policies must both be true to produce l_1 or l_2. The first would be setting the soft revocation bit and the latter for setting the hard revocation bits. Either of these will satisfy the `OR` operation, producing b_1, which in turn is an input to a final `OR` operation. To say that in other words: if the RA provided an authentic signature and fed either a hard- or soft-revocation hash (that is unique to a key), this is a valid branch for a single pseudonym, and its revocation will take place. Each branch digest b represents a valid soft- or hard-revocation for a single pseudonym. An `OR` operation may have up to 8 inputs, that's why it can be necessary to have multiple layers of these.

It is possible to calculate these by executing the commands in a trial session of the TC, but we showcase this by calculating it on the host. We start by initializing our variables and continue to define our very first branch digest: the activation. Recall that we have to write something to the index before it can be used. To ensure this is only done once, we utilize the Authorization Key's use-limit property and allow an initial write to the index, assuming the AK signs it. We continue into a loop where we iterate over all revocable pseudonyms. For each of the keys, we create two leaf digests, l_1 and l_2. We also initiate S_{data} and H_{data}, representing the parameters used in the `SetBits` command: the bits being set. We increment the `anonymizer` and set the H_{data} to that, thereby ensuring the parameter hash is unique. Now that we have anonymized the data, we can set the respective hard- and soft revocation bits in the data and continue calculating the revocation hashes. We see $k.sri.name$, the pseudonym k's soft revocation index's name, and hri representing the hard revocation index. These might be the same index, depending on the number of revocation domains.

With this data, we can calculate the soft- and hard revocation leaf digests, l_1, l_2, and then calculate the branch digest b. This is inserted into β, the list of all branch digests, and then we loop again. When all b's is calculated and inserted into β, we can calculate the final digest to be authorized during the activation of the revocation index.

Figure 6: Structure of the policy to be authorized



4.4 Activate Revocation Index

Before we can use the index, we must write to it, in this case, set everything to zero. Recall that the policy is built to allow a write to the index, if the AK provides a signature over the command to execute. The policy is not authorized yet, so the first thing the host does is prepare the values to be hashed and signed: hashing the final policy and generating the command parameter hash for the initial write and hashing this. It then continues to gain access to the authorizing key by providing the ACI's secret, inherently incrementing the authorization count. This process is redone for the hashed command parameter hash.

Now the host can get a signature over the policy and asking for a verification ticket by verifying the signature. A ticket guarantees this particular TC has verified the signature, thereby guaranteeing the command and its parameters have been authorized. This ticket can be saved and is needed every time `PolicyAuthroize` is required.

To gain writing authorization, the host initiates a new session and executes one of the index's valid policies, namely `PolicySigned` with the previously acquired signature over the zero-write command parameter hash. The current session digest should now match

the branch digest b_0, and the host executes PolicyOR with β. Assuming the TC verifies this, it will replace the session digest with a concatenation of all provided branch digests in β, which should be the index's authorized policy. The host finally executes PolicyAuthorize with the previously acquired ticket and signature. The TC verifies the ticket, the signature, and finally, the session digest matches the authorized digest. In this case, the TC replaces the session digest with the authorization key's name: the policy digest for the index. The host can now execute a write operation, and the index has now been activated.

While it is possible for the host to write anything to the index, it will not have any incentive to write anything but zeroes, as it would set pseudonyms in an initially revoked state. At the end of this phase, the index has been written to (it has been activated). The final policy calculated in the previous step has been authorized. It is now ready to be used to manage the revocation states of pseudonyms. As with the primordial ACI, the index has been initialized with an authorization limit and count and is immutable as the previous authorization key does not have any authorizations left. The index can now act as a protector for a new authorization key to regulate the use-count of this, and an additional revocation index can now be initialized.

4.5 Initialize New Authorization Counter Index

To initiate a new ACI, we must guarantee immutability by using an old AK, as it will only have a single authorization left, based on the previously defined ACI. So we start by creating a policy digest based on PolicySigned and the name of the old AK. Then we define a bit-space with that policy. As we should have a single authorization left in our authorization key, we can now create this key. Recall, to use this key, we must execute PolicySecret and provide the PIN to the previous ACI, thereby allowing us to use the key one last time. Therefore, we provide a signature by the old key and execute the PolicySigned command that will allow us then to write the authorization limit to the new ACI, if the TC can verify the signature.

4.6 Revocation

Upon receiving a (potential) revocation message, the host loads the corresponding RA's public key into the TC. The received message contains the public pseudonym key, pk_{ps}, which needs to be revoked, and a revocation hash and signature; i.e., of the respective revocation index, r_{index} and (hard or soft) revocation bit, r_{bit}. Recall that the policy for gaining write-access to the revocation index includes both PolicySigned (verifying that the message originates from the correct RA responsible for the trust domain where the vehicle also belongs to) and PolicyCpHash (correct revocation bit(s)): both these policies must be correctly satisfied before the TC allows the successful revocation. Executing these two, should produce a valid leaf digest (the result of the hashing done in the TC after PolicyCpHash is executed, also noted as l_x), validated by executing PolicyOR with a reference list of the possible leaf digest (λ) for that specific branch. Suppose the leaf digest matches a digest in the reference list. In that case, the session digest is replaced by a hash of the reference list: the branch digest, verified by an additional PolicyOR. We depict the execution of these policy commands in

Figure 7: Activate Revocation Index

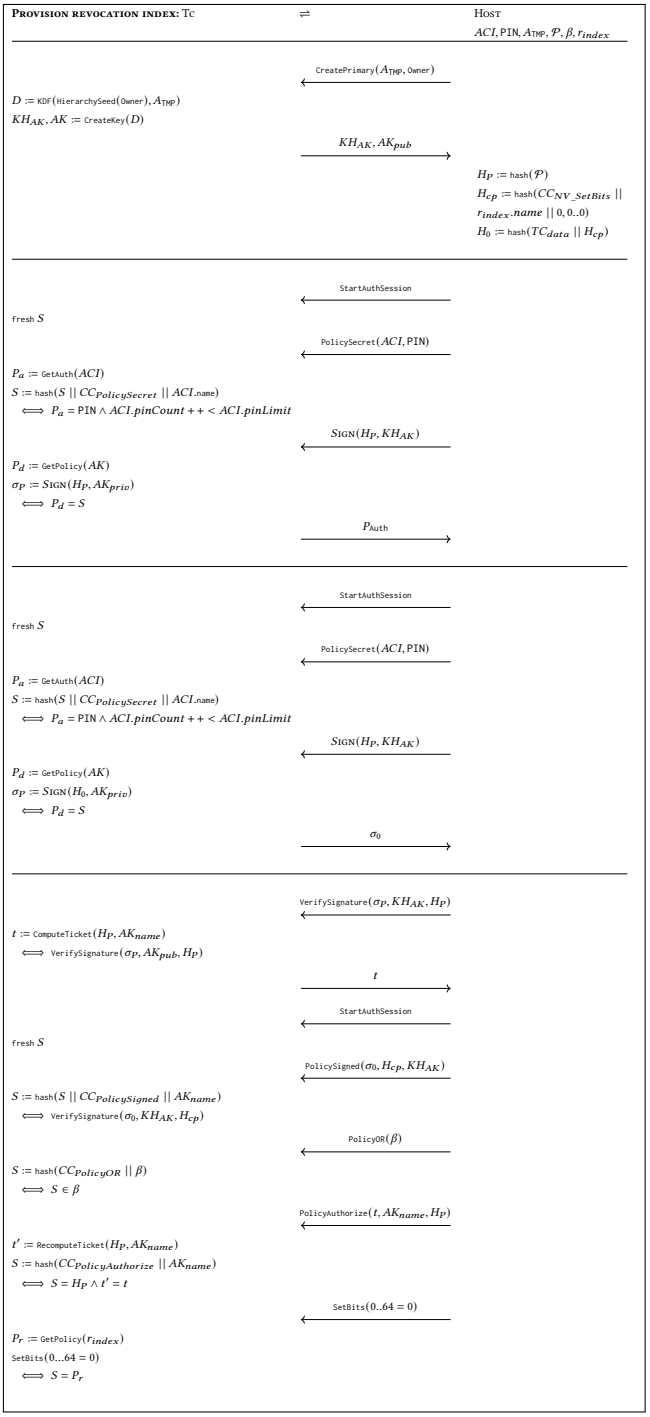
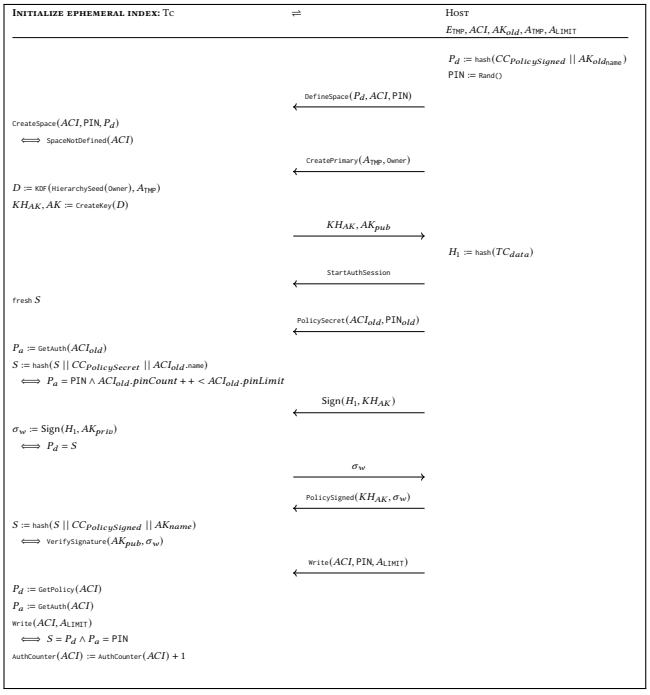


Figure 9 where it is highlighted that PolicyOR will replace the current session digest with a hashed concatenation of all provided reference branch digests (β) *if and only if* the current session digest is in the provided reference. If the reference list is unaltered, the

Figure 8: Initialize New Authorization Counter Index

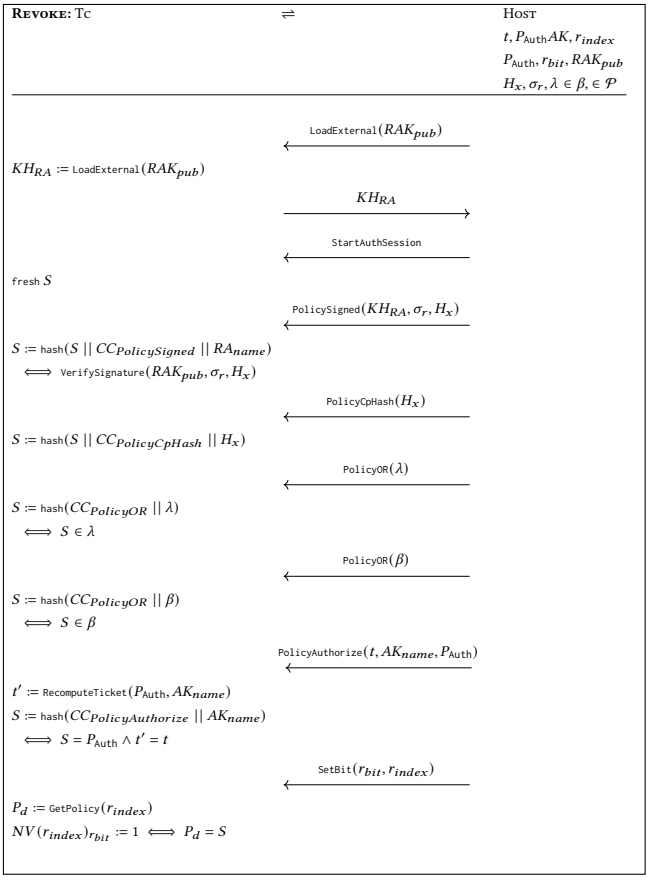


hashed concatenation should be the authorized policy, which is verified by executing `PolicyAuthorize` with a ticket (that proves the authorization), the name of the authorizing key, and, of course, the authorized policy. Suppose the session digest matches the approved policy: In that case, the session digest is changed to the hash of the command code of `PolicyAuthorize` and the authorizing key's name, which is the policy for the revocation index. The host can now execute a write to the index, but only with the parameters used in the `PolicyCpHash` command, ensuring the correct bits are set; hence the right pseudonym(s) are revoked.

Once all required pseudonyms, and their DAA key pairs, are deleted, the TC responds to the vehicle with a signed revocation confirmation σ_{rvk} which is then sent to the RA. Upon reception of the revocation confirmation, the RA verifies that this is signed by the same TC that issued the pseudonym certificate that was revoked, thus, implying that the correct vehicle has revoked itself. The entire signature can be verified using the DAA Verify operation as being signed by the TC that belongs to the misbehaving vehicle. By the end of this protocol, there are strong guarantees that the vehicle in question has been revoked without the need of any pseudonym resolution. The RA has verifiable evidence, from the vehicle, that it has performed the revocation enforced by the TC.

We have to note here that in case an attacker intercepts this revocation message, or a malicious vehicle host blocks the revocation message intended for the TC, then the revocation process will not be triggered and the vehicle's TC will not respond back with a revocation confirmation (note that this is also an issue for REWIRE [6] and O-TOKEN [25]). In order for revocation to take effect in this case, the TC needs to detect that this has occurred. This can be achieved by a *heartbeat* mechanism, such that the TC periodically

Figure 9: Revoking Vehicle's Pseudonyms & DAA Key Pairs



expects either a revocation message or a heartbeat (which may be a revocation intended for some other TC, or else a timed message). Revocation messages and heartbeats include information about the period they are intended for, thus, a heartbeat for one period cannot be used at a different time. They are signed by the RA so they cannot be tampered with or spoofed, and only one message is generated by the RA for each time period. Failure to receive a heartbeat message (or a series of messages so as to allow possible limited connectivity) can act as indication for potential misbehaviour that can also trigger revocation by the TC. In order to improve the safety level provided, this mechanism can make use of the types of heartbeat messages already provided for monitoring the status of one-hop vehicular topologies so as to produce indistinguishable communications and diminish the revocation vulnerability window existing in conventional CRLs [12].

5 PERFORMANCE EVALUATION

In this section, we evaluate analytically the computational complexity and overhead posed by our protocol and compare its performance with the current state-of-the-art in revocation solutions, being proposed by ETSI [5], based on the use of PKIs [6, 10, 19]. Recall that in most existing approaches to revocation, first, information about the misbehaving vehicle's long-term credentials (i.e., pseudonyms) is disseminated to the other vehicles through CRLs

or other means [8]. This is supported only if the leveraged pseudonym scheme supports *resolution* of participants' long-term identities from their pseudonyms; thus, enabling message- and identity-based hard revocation. As aforementioned in Section 2, besides such approaches being detrimental to the protection of all participating vehicles privacy, they also suffer from scalability issues - in the context of *bandwidth limitation* and *network latency* as continuous connection to the infrastructure RA(s) is needed.

In our DAA-enabled C-ITS, the RA broadcasts the revocation message which is received by all vehicles since the hosts are required to forward them to their trusted component, therefore, minimizing the bandwidth requirement; i.e., the vehicles do not need to periodically connect to the RA for "pulling" the latest version of the revocation blacklist. Even in the case of a threat model where vehicles are also allowed to block messages intended for the TC, this will lead to the RA re-broadcasting the revocation message until it receives a revocation confirmation (Section 4.6) - which poses an additional network overhead, however, minimal since the size of the revocation message is 128 bits.

In what follows, we consider all of the core phases as detailed in Section 4, including the Initialization of the ACI and the Revocation Index (RI) and the Activation of the RI followed by the actual (hard- or soft-) revocation process. To analyze the computational complexity, we divide the operations into two classes - (1) offline, and (2) online. All the operations which can be either pre-computed or not need to be executed in real-time are classified as offline operations. These include the computations at the TPM and the vehicle host needed for both **bootstrapping the entire revocation process**, i.e., initializing the Authorization and Revocation Indexes (Section 4.1) as well as registering the revocation hashes to the RA (Sections 4.3 and 4.4), and for **generating hard- and soft-revocation bits linked to a newly produced set of pseudonyms** - this phase, as detailed in Section 4.5, is invoked when the vehicle needs to securely create a new bunch of short-term anonymous credentials (more pseudonyms are needed to support the privacy-preserving authentication and verification of broadcasted CAM messages), thus, a new ACI needs to be created and initialized in a secure and immutable way. The operations which need to be performed in real-time are classified as online operations. These include the computations at the TPM and the vehicle host for performing the actual hard- and/or soft-revocation based on the received revocation message from the RA; more specifically, the execution of a series of policy authorization commands so as to check the revocation parameters received by the RA (Section 4.6).

5.1 Implementation Results

Evaluation Environment Setup & Testing Methods: Implementation was straightforward once the protocols were designed and written in terms of the appropriate TPM calls (see Tables 2 to 5) which also constitutes one of the novelties of this work since, to the best of our knowledge, it is one of the first complete instantiations of such a strong and provable revocation mechanism. The protocols were implemented in C/C++, using the GNU compiler and the IBM implementation of a TPM software stack (IBM TSS v. 1.6.0) [15].

The experiments' goal is to verify that the proposed solution is functional and outline the phases needed to support it, and most importantly, verify that near-constant revocation time is achievable even with numerous pseudonyms managed by a wide set of vehicles. It is, therefore, implemented as a single binary with multiple entities since we opted out from considering the *network latency* induced by vehicular mobility which usually implies volatile network connectivity. In all cases, our goal is to provide strong evidence on efficient revocation service provision, based on the use of DAA. and demonstrate its efficiency in comparison to other DAA-based approaches where the process execution time and overhead is linear to the size of the revocation list [16].

Table 2: Initialize Primordial Authorization Counter Index

Activity	Mean	± (95% CI)
Total Application Stack	578.20 ms	0.87 ms
Total TPM Stack	583.92 ms	0.67 ms
TPM2_CreatePrimary	259.69 ms	0.27 ms
TPM2_DefineSpace	30.04 ms	0.22 ms
TPM2_Sign	86.86 ms	0.20 ms
TPM2_StartAuthSession	18.22 ms	0.20 ms
TPM2_PolicySigned	132.57 ms	0.20 ms
TPM2_NV_Write	38.05 ms	0.30 ms
TPM2_FlushContext	9.14 ms	0.19 ms
TPM2_FlushContext	9.35 ms	0.21 ms

The results are generated using a laptop with Intel(R) Core(TM) i7-8665U CPU @ 1.90-2.11GHz. The protocols were then tested on two hardware-based trusted component platforms: an Infineon SLB9670 TPM [14] and a Nuvoton TPM so as to conduct a detailed investigation of the parameters that may affect the execution time of our revocation protocols. As can be seen in Section A.2, there is a strong interdependence of the optimal correctness of the results to the execution environmental setup and more specifically on the type of TC leveraged and subsequently on the crypto engine provided by the respective TC.

Performance Analysis: Although not the focus of the paper, we also opted to measure the actual timings for each one of the DAA Phases in order to provide a more comprehensive overview of the entire pseudonym lifecyle; from the creation to its secure and privacy-preserving revocation (when needed). The DAA JOIN and ISSUE protocols take up around 820 ms while the creation and certification of a pseudonym key (DAA CREATE) takes up 420ms. The DAA SIGN operation is rather fast and requires 80ms whereas the DAA VERIFY is split into two operations: the verification of the pseudonym key takes up 200ms and the verification of the ECDSA signature takes up 10ms.

As aforementioned, the most performance heavy operations, of our protocol, are the offline operations for initializing both the ACI and RI as well as registering the revocation hashes to the RA. Initializing the ACI, it's evident that the TPM is responsible for most of the incured time overhead (Table 2). Indeed, the host's only operation is a simple hashing operation for the policy digest. Interestingly enough, the accumulated time for TPM execution is larger than the whole operation. This might be due to the multiple initialization and deinitialization of the internal timing interfaces used. However, the combined time of creating the primordial ACI is

Table 3: Initialize Revocation Index

Activity	Mean	± (95% CI)
Total Application Stack	317.81 ms	0.53 ms
Total TPM Stack	312.88 ms	0.42 ms
TPM2_NV_ReadPublic	12.68 ms	0.19 ms
TPM2_CreatePrimary	260.62 ms	0.42 ms
TPM2_FlushContext	9.63 ms	0.19 ms
TPM2_DefineSpace	29.94 ms	0.23 ms

Table 4: Activate Revocation Index

Activity	Mean	± (95% CI)
Total Application Stack	920.28 ms	1.02 ms
Total TPM Stack	931.99 ms	0.15 ms
TPM2_CreatePrimary	260.26 ms	0.27 ms
TPM2_StartAuthSession	17.78 ms	0.24 ms
TPM2_PolicySecret	25.01 ms	0.19 ms
TPM2_Sign	96.85 ms	0.30 ms
TPM2_FlushContext	9.23 ms	0.19 ms
TPM2_StartAuthSession	17.37 ms	0.25 ms
TPM2_PolicySecret	24.91 ms	0.19 ms
TPM2_Sign	97.28 ms	0.32 ms
TPM2_FlushContext	9.18 ms	0.19 ms
TPM2_VerifySignature	132.71 ms	0.21 ms
TPM2_StartAuthSession	17.94.67 ms	0.29 ms
TPM2_PolicySigned	132.82 ms	0.0.20 ms
TPM2_PolicyOR	9.89 ms	0.19 ms
TPM2_PolicyAuthorize	25.72 ms	0.21 ms
TPM2_FlushContext	9.32 ms	0.19 ms
TPM2_NV_SetBits	36.80 ms	0.26 ms
TPM2_FlushContext	8.90 ms	0.18 ms

Table 5: Revocation - Soft (S) and Hard (H)

Activity	Mean (S)	Mean (H)	± (95% CI)
Total App. Stack	327.71 ms	323.91 ms	0.93/0.98 ms
Total TPM Stack	349.66 ms	348.28 ms	0.71/0.65 ms
TPM2_LoadExternal	92.29 ms	92.31 ms	0.22/0.22 ms
TPM2_StartAuthSession	17.73 ms	17.56 ms	0.25/0.22 ms
TPM2_PolicySigned	133.62 ms	132.90 ms	0.22/0.20 ms
TPM2_PolicyCpHash	9.18 ms	8.90 ms	0.19/0.18 ms
TPM2_PolicyOR	9.51 ms	9.55 ms	0.19/0.19 ms
TPM2_PolicyOR	9.87 ms	9.75 ms	0.19/0.19 ms
TPM2_PolicyAuthorize	25.57 ms	25.66 ms	0.19/0.19 ms
TPM2_NV_SetBits	34.06 ms	33.31 ms	0.35/0.25 ms
TPM2_FlushContext	9.35 ms	9.20 ms	0.19/0.18 ms
TPM2_FlushContext	9.04 ms	9.14 ms	0.19/0.19 ms

rather efficient and applicable to environments where this is needed to be executed multiple times. If new ACIs are needed, this time will increase slightly, as the previous AK's policies must be satisfied. These include starting a new session and executing `PolicySigned` and can be estimated to add a timing overhead of 45ms.

Moving towards the initialization of the revocation index (Table 3), this requires even less resources. This is mainly due to the limited operations taking place: the only actual operation being executed is the creation of the revocation index itself. This is shown in the context of a "worst-case" scenario where the ACI has to be read, and the Authorization Key has to be recreated; in actual operations, the AK would not have been flushed from the TPM in the previous stage. The time required for the host operations is slightly higher since more hashing operations are needed for creating the policy digest (yet it is small enough to be able to run even in resource-constrained vehicle ECUs).

Activating the revocation index is the heaviest phase of the protocol as seen in Table 4. Obviously, the TPM is again consuming most of the required resources. As described in Section 4.4, the AK's policy has to be satisfied twice: first for signing (authorizing) the final digest and secondly for providing the signature for the initial write in the revocation bits. However, recall that this is an offline operation which means that there is no need for a vehicle to wait till the previously created bunch of pseudonyms runs out before creating and activating new pseudonyms and their RBs.

Finally, and more interestingly, we can see that both soft- and hard-revocation takes an equal amount of time, as shown in Table 5. This is mainly due to the fact that the timing required is independent from the number of bits to be set; 1 for hard revocation or multiple in the case of a soft revocation as it is essentially an OR operation of 64 bits. Even if soft- and hard revocation bits lie in different indexes (Section 3.1), this will still be the case, as hard revocation works on a single index, and the same goes for a soft revocation.

6 CONCLUSIONS

In this paper, we proposed a novel revocation scheme based on the use of trusted computing technologies and more specifically the Direct Anonymous Attestation (DAA) protocol. This secure and privacy-preserving scheme supports trustworthy vehicle-local verification, thus, overcoming the challenges of current solutions that have been proposed in the standards based on the use of traditional PKIs. We have shown that our protocol achieves a significant performance improvement over the prior state of the art by verifying that near-constant revocation time is achievable even when multiple pseudonyms are entailed. We have evaluated all of the internal protocol phases through a qualitative analysis as well as through an actual implementation on two TPM variants, i.e., an Infineon and Nuvoton cryptoprocessor.

REFERENCES

[1] 5GAA Automotive Association. Oct. 2020. Privacy-by-Design Aspects of C-V2X. White Paper. Available online at https://5gaa.org/wp-content/uploads/2020/11/5GAA_White-Paper_Privacy_by_Design_V2X.pdf.

[2] Ernie Brickell, Jan Camenisch, and Liqun Chen. 2004. Direct Anonymous Attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*. 132–145.

[3] Liqun Chen, Dan Page, and Nigel P. Smart. 2010. On the Design and Implementation of an Efficient DAA Scheme. In *Smart Card Research and Advanced*

Application, Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 223–237.
[4] Liu Chunli and Tang Li Fang. 2012. The Application Mode in Urban Transportation Management Based on Internet of Things. In *Proceedings of the 2nd International Conference on Electric Technology and Civil Engineering (ICETCE)* (Three Gorges, China).
[5] European Commission. June 2018. Certificate Policy for Deployment and Operation of European Cooperative Intelligent Transport Systems (C-ITS).
[6] David Förster, Hans Löhr, Jan Zibuschka, and Frank Kargl. 2015. REWIRE – Revocation Without Resolution: A Privacy-Friendly Revocation Mechanism for Vehicular Ad-Hoc Networks. In *Trust and Trustworthy Computing*.
[7] D. Förster, F. Kargl, and H. Löhr. 2014. PUCA: A pseudonym scheme with user-controlled anonymity for vehicular ad-hoc networks (VANET). In *IEEE Vehicular Networking Conference (VNC)*. 25–32.
[8] Thanassis Giannetsos and Ioannis Krontiris. 2019. Securing V2X Communications for the Future: Can PKI Systems Offer the Answer?. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19)* (Canterbury, CA, United Kingdom).
[9] Stylianos Gisdakis, Thanassis Giannetsos, and Panos Papadimitratos. 2014. SPPEAR: Security & Privacy-preserving Architecture for Participatory-sensing Applications. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks* (Oxford, United Kingdom) *(WiSec '14)*. ACM, New York, NY, USA, 39–50.
[10] Stylianos Gisdakis, Marcello Lagana, Thanassis Giannetsos, and Panos Papadimitratos. 2013. SEROSA: SERvice oriented security architecture for Vehicular Communications. In *VNC*. IEEE, 111–118.
[11] Philippe Golle and Kurt Partridge. 2009. On the Anonymity of Home/Work Location Pairs. In *Proceedings of the 7th International Conference on Pervasive Computing* (Nara, Japan) *(Pervasive '09)*. 390–397.
[12] J. J. Haas, Yih-Chun Hu, and K. P. Laberteaux. 2011. Efficient Certificate Revocation List Organization and Distribution. *IEEE J.Sel. A. Commun.* 29, 3 (March 2011), 595–604.
[13] C. Hicks and F. D. Garcia. 2020. A Vehicular DAA Scheme for Unlinkable ECDSA Pseudonyms in V2X. In *2020 IEEE European Symposium on Security and Privacy (EuroS P)*. 460–473. https://doi.org/10.1109/EuroSP48549.2020.00036
[14] Infineon Technologies AG. [n.d.]. Iridium SLB 9670 TPM2.0 Linux. https://www.infineon.com/cms/en/product/evaluation-boards/iridium9670-tpm2.0-linux// [Online; accessed 03-May-2019].
[15] International Business Machines. [n.d.]. IBM's TPM 2.0 TSS Version 1119. https://sourceforge.net/projects/ibmtpm20tss/ [Online; accessed 03-May-2019].
[16] Vireshwar Kumar, He Li, Noah Luther, Pranav Asokan, Jung-Min (Jerry) Park, Kaigui Bian, Martin B. H. Weiss, and Taieb Znati. 2018. Direct Anonymous Attestation with Efficient Verifier-Local Revocation for Subscription System. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (Incheon, Republic of Korea) *(ASIACCS '18)*. Association for Computing Machinery, New York, NY, USA, 567–574.
[17] Virendra Kumar, Jonathan Petit, and William Whyte. 2017. Binary Hash Tree Based Certificate Access Management for Connected Vehicles. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '17)*. 145–155.
[18] M. E. Nowatkowski, J. E. Wolfgang, C. McManus, and H. L. Owen. 2010. The effects of limited lifetime pseudonyms on certificate revocation list size in VANETS. In *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*. 380–383.
[19] J. Petit, F. Schaub, M. Feiri, and F. Kargl. 2015. Pseudonym Schemes in Vehicular Networks: A Survey. *IEEE Communications Surveys Tutorials* 17, 1 (2015), 228–255.
[20] Marcos A. Simplicio, Eduardo Lopes Cominetti, Harsh Kupwade Patil, Jefferson E. Ricardini, and Marcos Vinicius M. Silva. 2019. ACPC: Efficient revocation of pseudonym certificates using activation codes. *Ad Hoc Networks* 90 (2019).
[21] Trusted Computing Group. [n.d.]. Trusted Computing Platform Alliance (TCPA) main specification. http://www.trustedcomputinggroup.org.
[22] Trusted Computing Group. [n.d.]. Trusted Platform Module Library Family "2.0" Specification - Parts 1-4 and Code, Revision 1.59. https://trustedcomputinggroup.org/resource/tpm-library-specification/.
[23] J. Whitefield, L. Chen, T. Giannetsos, S. Schneider, and H. Treharne. 2017. Privacy-enhanced capabilities for VANETs using direct anonymous attestation. In *2017 IEEE Vehicular Networking Conference (VNC)*. 123–130.
[24] Jorden Whitefield, Liqun Chen, Frank Kargl, Andrew Paverd, Steve Schneider, Helen Treharne, and Stephan Wesemeyer. 2017. Formal Analysis of V2X Revocation Protocols. In *International Workshop on Security and Trust Management*, Giovanni Livraga and Chris Mitchell (Eds.). Springer International Publishing, 147–163.
[25] J. Whitefield, L. Chen, F. Kargl, A. Paverd, S. Schneider, H. Treharne, and S. Wesemeyer. 2017. Formal Analysis of V2X Revocation Protocols. In *Security and Trust Management - 13th International Workshop, STM* (Oslo, Norway), Vol. 10547. Springer.
[26] W. Whyte, A. Weimerskirch, V. Kumar, and T. Hehn. 2013. A security credential management system for V2V communications. In *2013 IEEE Vehicular Networking Conference ((VNC'13)*. 1–8.
[27] L. Xi, D. Feng, Y. Qin, F. Wei, J. Shao, and B. Yang. 2014. Direct Anonymous Attestation in practice: Implementation and efficient revocation. In *2014 Twelfth Annual International Conference on Privacy, Security and Trust*. 67–74.
[28] Zhang Xiong, Hao Sheng, WenGe Rong, and Dave E. Cooper. 2012. Intelligent transportation systems for smart cities: a progress review. *Science China Information Sciences* (2012).

A APPENDIX

A.1 Building Blocks

A.1.1 Trusted Components. Trusted Components are reliable, tamper-evident, and secure processing units and can be present both in software or hardware. They can provide a variety of functions, such as cryptographic operations, key storage, and authentication. In this paper, we are showcasing the use of a Trusted Component by using a Trusted Platform Module (TPM), a physical hardware chip that provides both cryptographic functions, secure long- and short-term storage, and policy-based safeguards. In the following sections, we outline the essential properties of the TPM, enabling it to support vehicular revocation.

A.1.2 Key Storage and Hierarchies. A TPM hierarchy contains a cryptographic seed that forms a foundation for key generation. Using the seed and input parameters (key template), the TPM can generate a *Primary Key*, a key that de facto lies in the volatile memory but can be made persistent. As long as the template and seed remain the same, the key is recreatable ad hoc - an essential property due to the TPM's limited memory size. The TPM can create a new key as a child of a primary key, and these will always be unique. The parent key will encrypt such a key's private area and eject it onto the host, meaning it is unusable outside the TPM (a public part of the key is still functional, e.g., to verify signatures). The primary key must be present in the TPM memory to decrypt the key's private part, making it functional. The private area of a key can contain a password for use, and the public area can contain a policy, which we will talk about in the next section. The hash of the public area is the key's fingerprint and is called the *name* of the key.

A.1.3 Policies and Sessions. The TPM 2.0 specification [22] allows one to define a policy and require specific assertions or actions to take place before access to a protected object is allowed. The policy associated with an object is represented internally with a single statistically unique digest value known as the *policy digest*. Access to all objects making use of this enhanced authorization takes place via a session-based authorization procedure, during which the caller issues a sequence of policy commands to the TPM. Each policy command is an assertion that a particular statement is true in order for the policy to be satisfied. In this case, the policy command modifies a digest value associated with the session, characteristic of the particular policy expressed via the sequence of policy commands. This running accumulation of the digest value is called the *session digest*. Multiple policy commands will form a unique session digest based on the policy parameters and the TPM's internal checks. After the policy command sequence has been completed, the final value of the session digest is compared to the policy digest of the object being accessed. A match indicates that the sequence of invoked

Table 6: Experiment with two Nuvoton TPMs

Command	NPCT650	NPCT750
TPM2_CreatePrimary	65.68 ms	45.50 ms
TPM2_StartAuthSession	5.04 ms	10.70 ms
TPM2_PolicySecret	4.29 ms	11.60 ms
TPM2_Sign	204.38 ms	27.30 ms
TPM2_VerifySignature	263.68 ms	53.90 ms
TPM2_FlushContext (Session)	3.16 ms	9.80 ms
TPM2_FlushContext (Key)	2.71 ms	11.10 ms

policy commands satisfies the assertions expressed by the policy, and authorization is granted.

A.2 Experimentation Summary

In general, we see fast execution, and most of the work is done by the underlying trusted component. This implies that these timings are close to the most optimal, as optimizing the code will only have a non-essential impact. Furthermore, it is evident that re-creating the AK is a relatively heavy operation and should be kept in TPM volatile memory for as long as needed. Regarding a large number of pseudonyms, more `PolicyOR`'s are needed during revocation. This timing represents the first 6 pseudonyms, though just by adding two levels more (20ms added), we can support $6 \cdot 8^2 = 384$ pseudonyms. Adding one more (30ms) will support $6 \cdot 8^3 = 3072$ pseudonyms.

Interestingly, we noticed relatively large timings when it came to creating primary keys and verifying signatures. Therefore, an experiment was concluded in a different environment with two different TPMs. The following timings are acquired from a Dell desktop, x86, Ubuntu16.04 Xenial.

The results in Table 6 is an obvious example that both the environment and TPM will have an impact on the timings. In the latter example, we saw faster timings on creating keys but slower timings on signing and verifying signatures using an older Nuvoton TPM. The more modern Nuvoton TPM reduced the key operation's timings by a great deal, though more time is taken on non-key operations. This can be suspected due to Nuvoton potentially adding hardware support for ECC operations in their newer silicon. It is a prime example of timings being dependent on the physical implementation. These timings are extracted from the application layer, environmental factors such as operating system, workloads, and communication busses (I^2C, SPI, LPC) can impact timings. Despite the additional uncertainty revolving around timings in desktop environments, the revocation timings retain a small operation time. With only minimal added time when including an immense number of pseudonyms, it significantly improves traditional public key infrastructures, see section 8. Creating new pseudonyms dynamically is an operation that one must assume to happen often. This will include the three preparation phases: initializing a new ACI, initializing and activating a new revocation index. The new ACI is estimated to have only a slight increase in time, why the preparation to create new pseudonyms only takes in the order of around two seconds. As the host can execute these operations at any time, the host should do it before running out of pseudonyms, for example, when half of the existing pseudonyms have been used.

Algorithm 1 Calculate Final Policy Digest

(1) Initialize $\mathcal{P}$ as an array of hashes (capable of holding n hashes where n is the number of pseudonyms) and `anonymizer` as clear byte. Calculate activation branch digest $b_0 := H(H(b_0 \,||\, CC_{PolicySigned} \,||\, AK_{name}))$ and add to β
(2) For $k \in \mathcal{K}$:
  (a) Initialize l_1, l_2 as a two digest buffers. Initialize S_{data} and H_{data} as 8 byte buffers and set all bytes to zero.
  (b) Set bit identified by $k.s_{bit}$ high in byte number 8 in S_{data}
  (c) Increment `anonymizer` and set H_{data} to be the binary representation if it. Set bit identified by $k.h_{bit}$ high in byte number 8 in H_{data}
  (d) Compute soft- and hard revocation hashes
  $H_s := H(CC_{NV\_SetBits} \,||\, x, \,||\, k.sri.name \,||\, s_{data})$
  $H_h := H(CC_{NV\_SetBits} \,||\, k.hri.name, \,||\, k.hri.name \,||\, h_{data})$
  (e) Compute soft revocation leaf digest as
  $l_1 := H(H(l_1 \,||\, CC_{PolicySigned} \,||\, k.RA.name))$
  $l_1 := H(l_1 \,||\, CC_{PolicyCpHash} \,||\, H_s)$
  (f) Compute hard revocation leaf digest as
  $l_2 := H(H(l_2 \,||\, CC_{PolicySigned} \,||\, k.RA.name))$
  $l_2 := H(l_2 \,||\, CC_{PolicyCpHash} \,||\, H_h)$
  (g) Add branch digest $b := H(CC_{PolicyOR} \,||\, l_1 \,||\, l_2)$ to β
(3) Compute `finalPolicy` $:= H(CC_{PolicyOr} \,||\, \beta_1 \,||\, \beta_2 ... \,||\, \beta_n)$
(4) **Output** `finalPolicy`

A.3 Towards Near Zero-Trust Assumptions

Assuming near-zero trust assumptions for the vehicle requires additional validation of a number of processes that are executed in the host so as to be able to protect against compromised vehicle hosts (e.g., ECUs) that try to manipulate the parameters given to the attached trusted component. This essentially considers a Dolev-Yao adversarial model which essentially allows an adversary to monitor and modufy all interactions between the host and the TC. The critical operation to verify is the management of the policies generated on the host and that these have been correctly calculated for protecting the appropriate indexes and keys linked to active pseudonyms (an adversary can create a policy for inactive pseudonyms in which case a revocation message will be received and handled correctly but without any actual revocation results). The second operation to protect is the content of the ACI in order to validate the authorization limit. Finally, pseudonyms should be verified as to be governed by the correct RIs whose revocation hashes have been calculated correctly. These proofs can be done using the TPMs certification functionality and validated by a trusted entity, such as the Issuer. However, in order to acquire a high level of trust, the trusted party should not immediately release the correct pseudonym certificates. Instead, it should verify that the AK has been used to replace an ephemeral key and use its last authorization to write to the next-round ACI. Once finalized, it is impossible for the host to misuse the last authorization, and the system can be trusted completely without considering any other inherent trust assumptions.