

Scaling Pseudonymous Authentication for Large Mobile Systems

Mohammad Khodaei
Networked Systems Security Group
Stockholm, Sweden
khodaei@kth.se

Hamid Noroozi
Networked Systems Security Group
Stockholm, Sweden
hnoroozi@kth.se

Panos Papadimitratos
Networked Systems Security Group
Stockholm, Sweden
papadim@kth.se

ABSTRACT

The central building block of secure and privacy-preserving Vehicular Communication (VC) systems is a Vehicular Public-Key Infrastructure (VPKI), which provides vehicles with multiple anonymized credentials, termed *pseudonyms*. These pseudonyms are used to ensure message authenticity and integrity while preserving vehicle (thus passenger) privacy. In the light of emerging large-scale multi-domain VC environments, the efficiency of the VPKI and, more broadly, its scalability are paramount. By the same token, preventing misuse of the credentials, in particular, Sybil-based misbehavior, and managing “*honest-but-curious*” insiders are other facets of a challenging problem. In this paper, we leverage a state-of-the-art VPKI system and *enhance* its functionality towards a highly-available, dynamically-scalable, and resilient design; this ensures that the system remains operational in the presence of benign failures or resource depletion attacks, and that it dynamically *scales out*, or possibly *scales in*, according to request arrival rates. Our full-blown implementation on the Google Cloud Platform shows that deploying large-scale and efficient VPKI can be cost-effective.

KEYWORDS

VANETs, VPKI, Security, Privacy, Availability, Scalability, Resilient, Micro-service, Container Orchestration, Cloud.

1 INTRODUCTION

In Vehicular Communication (VC) systems, vehicles beacon Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs) periodically, at high rates, to enable transportation safety and efficiency. It has been well-understood that VC systems are vulnerable to attacks and that the privacy of their users is at stake. As a result, security and privacy solutions have been developed by standardization bodies (IEEE 1609.2 WG [51] and ETSI [45]), harmonization efforts (C2C-CC [75]), and projects (SeVeCom [53, 71, 73], PRESERVE [76], and CAMP [2, 88]). A consensus towards using Public Key Cryptography (PKC) to protect Vehicle-to-Vehicle (V2V)/Vehicle-to-Infrastructure (V2I) (V2X) communication is reached: a set of short-lived anonymized certificates, termed *pseudonyms*, are issued by a Vehicular Public-Key Infrastructure (VPKI), e.g., [54, 60, 88], for registered vehicles. Vehicles switch from one pseudonym to a non-previously used one towards message unlinkability, as pseudonyms are per se inherently unlinkable. Pseudonymity is conditional, in the sense that the corresponding long-term vehicle identity can be retrieved by the VPKI when needed, e.g., if vehicles deviating from system policies.

Deploying a VPKI differs from a traditional Public-Key Infrastructure (PKI), e.g., [6, 13, 15]. One of the most important factors

is the PKI dimension, i.e., the number of registered “users” (vehicles) and the multiplicity of certificates per user. According to the US Department of Transportation (DoT), a VPKI should be able to issue pseudonyms for more than 350 million vehicles across the Nation [1]. Considering the average daily commute time to be 1 hour [1] and a pseudonym lifetime of 5 minutes, the VPKI should be able to issue at least 1.5×10^{12} pseudonyms per year, i.e., 5 orders of magnitude more than the number of credentials the largest current PKI issues (10 million certificates per year [88]). Note that this number could be even greater for the entire envisioned Intelligent Transport Systems (ITSs) ecosystem, e.g., including pedestrians and cyclists, Location Based Services (LBSs) [45, 74, 83] and vehicular social networks [52]. More so, outside the VC realm, there is an ongoing trend towards leveraging short-lived certificates [85] for the Internet: web servers request new short-lived certificates, valid for a few days [85]. This essentially diminishes the vulnerability window, e.g., if a single Certification Authority (CA) were compromised [85], or if a large fraction of certificates needed to be revoked after the latest Certificate Revocation List (CRL) was distributed among all entities [39, 57, 67].

With emerging large-scale multi-domain VC environments [45, 51, 61, 74, 75], the efficiency of the VPKI and, more broadly, its scalability are paramount. Vehicles could request pseudonyms for a long period, e.g., 25 years [63]. However, extensive pre-loading with millions of pseudonyms per vehicle for a long period is computationally costly and inefficient in terms of utilization [60]. Moreover, in case of revocation [39, 57, 67], a huge CRL should be distributed among all vehicles due to long lifespan of the credentials, e.g., [63]: a sizable portion of the CRL is irrelevant to a receiving vehicle and can be left unused, i.e., wasting of significant bandwidth for CRL distribution [57, 84]. Alternatively, each vehicle could interact with the VPKI regularly, e.g., once or a few times per day, not only to refill its pseudonym pool but also to fetch the latest revocation information¹. However, the performance of a VPKI system can be drastically degraded under a clogging Denial of Service (DoS) attack [54, 60], thus, compromising the availability of the VPKI entities. Moreover, a *flash crowd* [32], e.g., a surge in pseudonym acquisition requests during rush hours, could render the VPKI unreachable, or drastically decrease its quality of service.

The cost of VPKI unavailability is twofold: security (degradation of road safety) and privacy. An active malicious entity could prevent other vehicles from accessing the VPKI to fetch the latest revocation information. Moreover, signing CAMs with the private keys corresponding to expired pseudonyms, or the Long Term Certificate (LTC), is insecure and detrimental to user privacy. Even

¹Note that Cellular-V2X provides reliable and low-latency V2X communication with a wide range of coverage [28, 30, 31]; thus, network connectivity will not be a bottleneck.

though one can refill its pseudonym pool by relying on anonymous authentication primitives, e.g., [35, 36, 56, 72], the performance of the safety-related applications could be degraded. For example, leveraging anonymous authentication schemes for the majority of vehicles results in causing 30% increase in cryptographic processing overhead in order to validate CAMs [56]. Thus, it is crucial to provide a highly-available, scalable, and resilient VPKI design that could efficiently issue pseudonyms in an *on-demand* fashion² [55, 65].

Considering a multi-domain development of VC systems, with a multiplicity of service providers, each vehicle could obtain pseudonyms from various service providers. The acquisition of multiple simultaneously valid (sets of) pseudonyms would enable an adversary to inject multiple erroneous messages, e.g., hazard notifications, as if they were originated from multiple vehicles, or affect protocols based on voting, by sending out false, yet authenticated, information. Even though there are distributed schemes to identify Sybil [43] nodes, e.g., [47, 90], or mitigate this vulnerability by relying on Hardware Security Modules (HSMs) [71], a VPKI system should prevent such credentials misuse on the infrastructure side, e.g., [54, 60]. However, when deploying such a system, e.g., [38, 68], on the cloud, a malicious vehicle could repeatedly request pseudonyms; in fact, requests might be delivered to different replicas of a micro-service, releasing multiple simultaneously valid pseudonyms. Mandating a centralized database, shared among all replicas to ensure *isolation* and *consistency* of all transactions, would mitigate such a vulnerability. However, this contradicts highly efficient and timely pseudonyms provisioning for large-scale mobile systems.

Contributions: In this paper, we leverage and *enhance* a state-of-the-art VPKI, and propose a *VPKI as a Service (VPKIaaS)* system towards a highly-available, dynamically-scalable, and fault-tolerant (highly-resilient) design, ensuring the system remains operational in the presence of benign failures or any resource depletion attack (clogging a DoS attack). Moreover, our scheme eradicates Sybil-based misbehavior when deploying such a system on the cloud with multiple replicas of a micro-service without diminishing the pseudonym acquisition efficiency. All procedures of deployment and migration to the cloud, e.g., bootstrapping phase, initializing the micro-services, pseudonym acquisition process, monitoring health and load metrics, etc., are fully automated. Through extensive experimental evaluation, we show that the VPKIaaS system could dynamically scale out, or possibly scale in³, based on the VPKIaaS system workload and the requests' arrival rate, so that it can comfortably handle *unexpected* demanding loads while being cost-effective by systematically allocating and deallocating resources. Our experimental evaluation shows a 36-fold improvement over prior work [38]: the processing delay to issue 100 pseudonyms for [38] is approx. 2010 ms, while it is approx. 56 ms in our system. Moreover, the performance of [60] drastically decreases when there is a surge in the pseudonym request arrival rates; on

the contrary, our VPKIaaS system can comfortably handle demanding loads request while efficiently issuing batches of pseudonyms.

In the rest of the paper, we describe background and related work (Sec. 2) and the system model and objectives (Sec. 3). We then explain the VPKIaaS system, detailing security protocols (Sec. 4), and provide a qualitative analysis (Sec. 5), followed by a quantitative analysis (Sec. 6), before the conclusion (Sec 7).

2 BACKGROUND AND RELATED WORK

A VPKI can provide vehicles with valid pseudonyms for a long period, e.g., 25 years [63]. However, extensive preloading with millions of pseudonyms per vehicle for such a long period is computationally costly, inefficient in terms of utilization and cumbersome for revocation [57, 61]. On the contrary, several proposals suggest more frequent Vehicle-to-VPKI interactions, namely *on-demand* schemes, e.g., [46, 54, 60, 81]. This strategy provides more efficient pseudonym utilization and revocation, thus being effective in fending off misbehavior. But, for on-demand pseudonym acquisition, one needs to design (and deploy) an efficient and scalable system while being resilient against any resource depletion attack. Even though VPKI systems may handle large-scaled distributed scenarios, e.g., [38], there is lack of dynamic scalability (i.e., dynamically scale out/in according to the arrival rates) and resilient to a resource depletion attack, e.g., a Distributed DoS (DDoS) attack. Beyond a significant performance improvement over [38], our VPKIaaS implementation is highly-available, dynamically-scalable, and fault-tolerant.

Sybil-based [43] misbehavior can seriously affect the operation of VC systems, as multiple fabricated non-existing vehicles could pollute the network by injecting false information. For example, an adversary with multiple valid pseudonyms, termed here a *Sybil* node, could create an illusion of traffic congestion towards affecting the operation of a traffic monitoring system, or broadcast fake misbehavior detection votes [77, 78, 80], or disseminate Spam to other users in a vehicular social network [52]. The idea of enforcing non-overlapping pseudonym lifetimes was first proposed in [71]. This prevents an adversary from equipping itself with multiple valid identities, and thus affecting protocols of collection of multiple inputs, e.g., based on voting, by sending out redundant false, yet authenticated, information. Even though this idea has been accepted, a number of proposals, e.g., [63, 88], do not prevent a vehicle from obtaining simultaneously valid pseudonyms via multiple pseudonym requests. The existence of multiple pseudonym issuers deteriorate the situation: a vehicle could request pseudonyms from multiple service providers, while each of them is not aware whether pseudonyms for the same period were issued by any other service provider. One can mitigate this vulnerability by relying on an HSM, ensuring all signatures are generated under a single valid pseudonym at any time. There are also distributed schemes to detect Sybil nodes based on radio characteristics and triangulation, e.g., [47, 90]; such strategies are application-dependent, e.g., this cannot guarantee the operation of a traffic monitoring system from an adversary who disseminates multiple traffic congestion messages, each signed under a distinct "fake" private key.

²Unlike issuing short-lived certificates [85] for the Internet that responses can be cached, issuing on-demand pseudonyms cannot be precomputed: each vehicle requests new certificates with a different public key, important for unlinkability/privacy.

³In the cloud terminology, scaling in/out, termed *horizontal* scaling, refers to replicating a new instance of a service, while scaling up/down, termed *vertical* scaling, refers to allocating/deallocating resources for an instance of a given service.

V-tokens [81] prevents a vehicle from obtaining multiple simultaneously valid pseudonyms due to having service providers communicating with each other, e.g., a distributed hash table. SEC-MACE [60] (including its predecessors [54, 55]) prevents Sybil-based misbehavior on the infrastructure side without the need for an additional entity, i.e., extra interactions or intra-VPKI communications. More specifically, it ensures each vehicle has one valid pseudonym at any time in a multi-domain environment. However, when deploying such a system on the cloud, a malicious vehicle could repeatedly request pseudonyms, hoping that requests are delivered to different replicas of a micro-service, thus obtaining multiple simultaneously valid pseudonyms, e.g., [38, 68]. Unlike such schemes, our VPKIaaS scheme prevents Sybil-based misbehavior on the cloud-deployed infrastructure: it ensures that each vehicle can only have one valid pseudonym at any time in a multi-domain VC environment; more important, it does not affect timely issuance of pseudonyms.

The VPKI entities are, often implicitly, assumed to be fully trustworthy. Given the experience from recent mobile applications, e.g., [48, 64, 66], the adversarial model is extended from fully trustworthy to *honest-but-curious* VPKI servers, notably in [60, 88]. Such honest-but-curious entities may subvert the security protocols and deviate from system policies if gained an advantage without being identified, e.g., inferring user sensitive information [58, 59, 87, 89]. Outside the VC realm, there are different proposals for PKI to be resilient against *compromised* insiders. Such schemes rely on signing a certificate by more than a threshold number of CAs, e.g., [44, 62]; however, such schemes cannot be used by VC systems. For example, issuing a certificate in [44] takes approximately 2 minutes and it varies with the number of required CAs. Obviously, this contradicts with *on-demand* pseudonym acquisition strategies for VC systems, e.g., [55, 56, 60, 65], which necessitate efficient pseudonym provisioning.

3 SYSTEM MODEL AND OBJECTIVES

3.1 Overview and Assumptions

A VPKI consists of a set of Certification Authorities (CAs) with distinct roles: the Root CA (RCA), the highest-level authority, certifies other lower-level authorities; the Long Term CA (LTCA) is responsible for the vehicle registration and the Long Term Certificate (LTC) issuance, and the Pseudonym CA (PCA) issues pseudonyms for the registered vehicles. Pseudonyms have a lifetime (a validity period), typically ranging from minutes to hours; in principle, the shorter the pseudonym lifetime is, the higher the unlinkability and thus the higher privacy protection can be achieved. We assume that each vehicle is registered only with its *Home-LTCA (H-LTCA)*, the *policy decision and enforcement point*, reachable by the registered vehicles. Without loss of generality, a *domain* can be defined as a set of vehicles in a region, registered with the H-LTCA, subject to the same administrative regulations and policies [61, 70]. There can be several PCAs, each active in one or more domains; any legitimate, i.e., registered, vehicle is able to obtain pseudonyms from any PCA, the pseudonym provider (as long as there is a trust established between the two CAs). Trust between two domains can be established with the help of the RCA, or through cross certification.

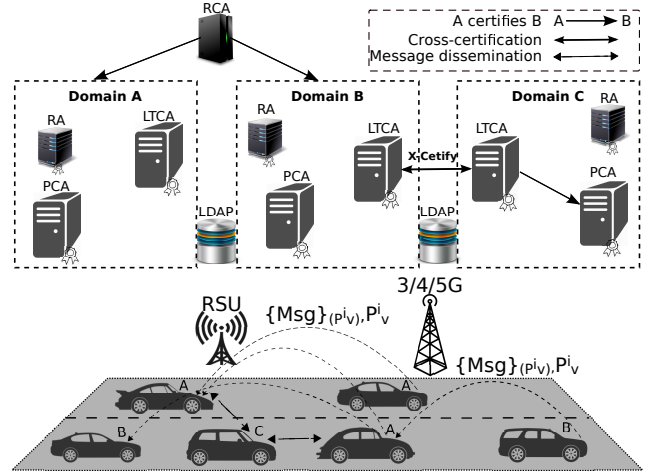


Figure 1: A VPKI Overview for Multi-domain VC Systems.

Each vehicle interacts with the VPKI entities to obtain a batch of pseudonyms, each having a corresponding short-term private key, to sign and disseminate their mobility information, e.g., CAMs or DENMs, time- and geo-stamped, periodically or when needed as a response to a specific event. Fig. 1 shows an overview of a VPKI with three domains, A, B and C. Domains A and B have established trust with the help of a higher level authority, i.e., the RCA, while domains B and C have established security association by cross certification. The vehicles in the figure are labeled with the domains they are affiliated to. A vehicle registered in domain A digitally signs outgoing messages with the private key, k_v^i , corresponding to P_v^i , which signifies the current valid pseudonym signed by the PCA. The pseudonym is then attached to the signed messages to enable verification by any recipient. Upon reception, the pseudonym is verified before the message itself (signature validation). This process ensures communication authenticity, message integrity, and non-repudiation. Vehicles switch from one pseudonym to another one (non-previously used) to achieve unlinkability, thus protecting sender’s privacy as the pseudonyms are inherently unlinkable.

Each vehicle “decides” when to trigger the pseudonym acquisition process based on various factors [55]. Such a scheme requires sparse connectivity to the VPKI, but it facilitates an On-Board Unit (OBU) to be *preloaded* with pseudonyms proactively, covering a longer period, e.g., a week or a month, should the connectivity be expected heavily intermittent. A universally fixed interval, Γ , is specified by the H-LTCA and all pseudonyms in that domain are issued with the lifetime (τ_p) aligned with the VPKI clock [60]. As a result of this policy, at any point in time, all the vehicles transmit using pseudonyms that cannot be distinguished based on their issuance time thanks to this time alignment.

All vehicles (OBUs) registered are equipped with HSMs, ensuring that private keys never leave the HSM. Moreover, we assume that there is a misbehavior detection system, e.g., [34], that triggers revocation. The Resolution Authority (RA) can initiate a process to resolve and revoke all pseudonyms of a misbehaving vehicle [69]: it interacts with the corresponding PCAs and LTCA (a detailed protocol description, e.g., in [54, 60]) to resolve and revoke all credentials issued for a misbehaving vehicle. Consequently, the misbehaving

vehicle can no longer obtain credentials from the VPKI. The VPKI is responsible for distributing the CRLs and notifying all legitimate entities about the revocation, e.g., [57]. We further assume that the cloud service providers are honest and they provide a service with the desired Service Level Agreement (SLA); in terms of secret management, we assume that the cloud service providers are fully trustworthy.

3.2 Adversarial Model and Requirements

We extend the general adversary model in secure vehicular communications [60, 70] to include an *honest-but-curious* service provider, i.e., a PCA that attempts to gain advantages towards its goal, e.g., profiling users. In addition, in the context of this work, malicious PCAs could try to (i) issue multiple sets of (simultaneously valid) pseudonyms for a legitimate vehicle, or (ii) issue a set of pseudonyms for a non-existing (illegitimate) vehicle, or (iii) fraudulently accuse different vehicles (users) during a pseudonym resolution process. A deviant LTCA could attempt to map a different LTC during the resolution process, thus misleading the system. In our adversarial model, we assume that the LTCA does not misbehave by unlawfully registering illegitimate vehicles, i.e., issuing fake LTCs, but it could be tempted to issue fake *authorization tickets*, to be used during pseudonym acquisition process⁴. The RA can also continuously initiate pseudonym validation process towards inferring user sensitive information. Our adversarial model considers multiple VPKI servers collude, i.e., share information that each of them individually infers with the others, to harm user privacy.

In a multi-PCA environment, malicious (compromised) clients raise two challenges. First, they could repeatedly request multiple simultaneously valid pseudonyms, thus misbehaving each as multiple registered legitimate-looking vehicles. Second, they could degrade the operations of the system by mounting a clogging DoS attack against the VPKI servers. *External adversaries*, i.e., unauthorized entities, could try to harm the system operations by launching a DoS (or a DDoS) attack, thus degrading the availability of the system. But they are unable to successfully forge messages or ‘crack’ the employed cryptosystems and cryptographic primitives.

Security and privacy requirements for V2X communications have been extensively specified in [70], and additional requirements for VPKI entities in [60] and the CRL distribution in [57]. Beyond the aforementioned requirements, we need to thwart Sybil-based attacks when deploying VPKIaaS system on the cloud (without degrading efficient pseudonym issuance). At the same time, we need to ensure that the VPKIaaS system is highly-available and dynamically-scalable: the system *dynamically* scales out, or possibly scales in, according to the requests’ arrival rate, to handle any demanding load while being cost-effective by systematically allocating and deallocating resources. Moreover, we need to ensure that the scheme is resilient to any resource depletion attack.

⁴During the registration process, the H-LTCA registers a vehicle upon receiving a request from the corresponding Original Equipment Manufacturer (OEM), i.e., to fraudulently register a vehicle, two entities must collude. But, in order to issue a fake ticket, the H-LTCA could do it without interacting with any other entity.

4 VPKI SERVICES OVERVIEW & SECURITY PROTOCOLS

In the registration phase, each H-LTCA registers vehicles within its domain and maintains their long-term identities. At the bootstrapping phase, each vehicle needs to discover the VPKI-related information, e.g., the available PCAs in its home domain, or the desired Foreign-LTCA (F-LTCA) and PCAs in a foreign domain, along with their corresponding certificates. To facilitate the overall intra-domain and multi-domain operations, a vehicle first finds such information from a Lightweight Directory Access Protocol (LDAP) [82] server. This is carried out without disclosing the real identity of the vehicle. We presume connectivity to the VPKI, e.g., via Roadside Units (RSUs) or Cellular-V2X; should the connectivity be intermittent, vehicle, i.e., the OBU, could initiate pseudonym provisioning proactively based on different parameters, e.g., the number of remaining valid pseudonyms, the residual trip duration, and the networking connectivity.

The H-LTCA authenticates and authorizes vehicles over a mutually authenticated Transport Layer Security (TLS) [42] tunnel. This way the vehicle obtains a *native ticket* (*n-tkt*) from its H-LTCA while the targeted PCA or the actual pseudonym acquisition period is hidden from the H-LTCA; the ticket is anonymized and it does not reveal its owner’s identity (Protocol 5 and Protocol 6 in the Appendix). The ticket is then presented to the intended PCA, over a unidirectional (server-only) authenticated TLS, to obtain pseudonyms (Protocol 1).

When the vehicle travels in a foreign domain, it should obtain new pseudonyms from a PCA operating in that domain; otherwise, the vehicle would stand out and be more easily traceable (linkable). The vehicle first requests a *foreign ticket* (*f-tkt*) from its H-LTCA (without revealing its targeted F-LTCA) so that the vehicle can be authenticated and authorized by the F-LTCA. In turn, the F-LTCA provides the vehicle with a new ticket (*n-tkt*), which is native within the domain of the F-LTCA to be used for pseudonym acquisition in that (foreign) domain. The vehicle then interacts with its desired PCA to obtain pseudonyms. Obtaining an *f-tkt* is transparent to the H-LTCA: the H-LTCA cannot distinguish between native and foreign ticket requests. This way, the PCA in the foreign domain cannot distinguish native requesters from foreign ones. For liability attribution, our scheme enables the RA, with the help of the PCA and the LTCA, to initiate a resolution process, i.e., to resolve a pseudonym to its long-term identity. Each vehicle can interact with any PCA, within its home or a foreign domain, to fetch the CRL [57] and perform Online Certificate Status Protocol (OCSP) [54] operations, authenticated with a current valid pseudonym.

4.1 VPKI as a Service (VPKIaaS)

We migrate the VPKI on the Google Cloud Platform (GCP) [22] for the availability, reliability, and dynamic scalability of the VPKI system under various circumstances. Fig. 2 illustrates a high-level abstraction of the VPKIaaS architecture on a managed Kubernetes cluster [25] on GCP.⁵ A set of Pods will be created for each microservice, e.g., LTCA or PCA, from their corresponding container

⁵Note that the RCA entity is assumed to be off-line, thus not included in this abstraction.

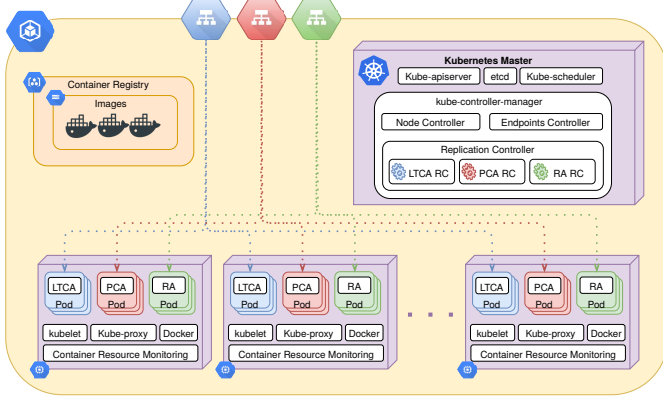


Figure 2: A High-level Overview of VPKIaaS Architecture.

images, facilitating their horizontal scalability. When the rate of pseudonym requests increases, the Kubernetes master, shown on the top in Fig. 2, schedules new Pods or kills a running Pod in case of benign failures, e.g., system faults or crashes, or resource depletion attacks, e.g., a DoS attack. The Pods could be scaled out to the number, set in the deployment configuration, or scaled out to the amount of available resources enabled by Kubernetes nodes.

Each Pod publishes two types of metrics: *load* and *health*. The load metric values are generated by a resource monitoring service, which facilitates horizontal scaling of a micro-service: upon reaching a threshold of a pre-defined load, replication controller replicates a new instance of the micro-service to ensure a desired SLA. The health metric ensures correct operation of a micro-service by persistently monitoring its status: a faulty Pod is killed and a new one is created. In our VPKIaaS system, we define CPU usages as the load metric. In order to monitor the health condition of a micro-service, dummy requests (dummy tickets for the LTCA micro-services and dummy pseudonyms for the PCA micro-services) are locally queried by each Pod⁶.

4.2 Security Protocols

In this section, we provide the detailed description of pseudonym acquisition processes (Protocol 1) and pseudonym issuance validation process (Protocol 2) in order to identify misbehaving PCA issuing fraudulent pseudonym. Furthermore, in order to mitigate Sybil attacks on the side of VPKIaaS system, we propose two protocols (Protocols 3 and 4): an in-memory key-value Redis database [14] is shared among all replicas of a micro-service, to facilitate efficient validation of tickets and pseudonyms requests. Table 1 shows the notation used in the security protocols.

4.2.1 Pseudonym Acquisition Process (Protocol 1). Each vehicle first requests an anonymous ticket [54, 55] from its H-LTCA, using it to interact with the desired PCA to obtain pseudonyms; due to lack of space, we provide the detailed ticket acquisition process in Appendix. Upon reception of a valid ticket, it generates Certificate Signing Requests (CSRs) with Elliptic Curve Digital Signature Algorithm (ECDSA) public/private key pairs [45, 51] and

⁶A dummy ticket request is constructed by an LTCA Pod to validate the correctness of ticket issuance procedure while a dummy pseudonym request is constructed by a PCA Pod to ensure the correctness of pseudonym issuance procedure. Such dummy requests cannot be used by a compromised Pod to issue fake pseudonyms (see Sec. 5).

Table 1: Notation used in the protocols

$(P_v^i)_{pca}, P_v^i$	a pseudonym signed by the PCA
(LK_v, Lk_v)	long-term public/private key pairs
(K_v^i, k_v^i)	pseudonymous public/private key pairs
$Id_{req}, Id_{res}, Id_{ca}$	request/response/CA unique identifiers
$(msg)_{\sigma_v}$	a signed message with the vehicle's private key
N, Rnd	nonce, a random number
t_{now}, t_s, t_e	fresh/current, starting, and ending timestamps
$n-tkt, f-tkt$	native ticket, foreign ticket
$H()$	hash function
$Sign(Lk, msg)$	signing a message with the private key (Lk)
$Verify(LK, msg)$	verifying a message with the public key
τ_P	pseudonym lifetime
Γ	interacting interval with the VPKI
IK	identifiable key
V	vehicle
ζ, χ	temporary variables

Protocol 1 Issuing Pseudonyms (by the PCA)

```

1: procedure ISSUEPSNYMS(Req)
2:   Req → (Idreq, Rndn-tkt, tktσLTCA, {(Kv1)σkv1, ..., (Kvn)σkvn}, N, tnow)
3:   Verify(LTCA, (tkt)σLTCA)
4:   tktσLTCA → (SN, H(IdPCA || Rndtkt), IKtkt, ts, te, Exptkt)
5:   H(Idthis-pca || Rndn-tkt)  $\stackrel{?}{=} H(Id_{pca} || Rnd_{n-tkt})$ 
6:   Rndv ← GenRnd()
7:   for i=1 to n do
8:     Begin
9:       Verify(Kvi, (Kvi)σkvi)
10:      IKPvi ← H(IKtkt || Kvi || tsi || tei || Hi(Rndv))
11:      if i = 1 then
12:        SNi ← H(IKPvi || Hi(Rndv))
13:      else
14:        SNi ← H(SNi-1 || Hi(Rndv))
15:      end if
16:      ζ ← (SNi, Kvi, IKPvi, tsi, tei)
17:      (Pvi)σpca ← Sign(Lkpca, ζ)
18:    End
19:   return (Idres, {(Pv1)σpca, ..., (Pvn)σpca}, Rndv, N+1, tnow)
20: end procedure

```

sends the request to the PCA. Vehicle-LTCA is over mutually authenticated TLS [42] tunnels (or Datagram TLS (DTLS) [79]) while the vehicle-PCA communication is over a unidirectional (server-only) authenticated TLS (or DTLS); this ensures that the PCA does not infer the actual identity of the requester.

Having received a request, the PCA verifies the ticket signed by the H-LTCA (assuming trust is established between the two) (steps 1.2–1.3). The PCA then decapsulates the ticket and verifies the pseudonym provider identity (step 1.4–1.5). Then, the PCA generates a random number (step 1.6) and initiates a proof-of-possession protocol to verify the ownership of the corresponding private keys by the vehicle (step 1.9). Then, it calculates the “*identifiable key*”, $IK : H(IK_{tkt} || K_v^i || t_s^i || t_e^i || H^i(Rnd_v))$ (step 1.10). This essentially prevents a compromised PCA from mapping a different ticket during resolution process, or identifies a malicious PCA if issued a pseudonym without a valid ticket received. The PCA implicitly correlates a batch of pseudonyms belonging to each requester (steps 1.11–1.15).

Protocol 2 Pseudonym Issuance Validation Process

- (1) $V_j : P_v^i \leftarrow (SN^i, K_v^i, IK_{P_v^i}, t_s^i, t_e^i)$
- (2) $V_j : \zeta \leftarrow (P_v^i)$
- (3) $V_j : (\zeta)_{\sigma_v} \leftarrow \text{Sign}(P_v^i, \zeta)$
- (4) $V_j \rightarrow \text{RA} : (Id_{req}, (\zeta)_{\sigma_v}, t_{now})$
- (5) $\text{RA} : \text{Verify}(P_v, (\zeta)_{\sigma_v})$
- (6) $\text{RA} : \zeta \leftarrow (P_v^i)$
- (7) $\text{RA} : (\zeta)_{\sigma_{ra}} \leftarrow \text{Sign}(Lk_{ra}, \zeta)$
- (8) $\text{RA} \rightarrow \text{PCA} : (Id_{req}, (\zeta)_{\sigma_{ra}}, LTC_{ra}, N, t_{now})$
- (9) $\text{PCA} : \text{Verify}(LTC_{ra}, (\zeta)_{\sigma_{ra}})$
- (10) $\text{PCA} : (tkt, Rnd_{IK_{P_v^i}}) \leftarrow \text{Resolve}(P_v^i)$
- (11) $\text{PCA} : \chi \leftarrow (SN_{p_i}, tkt_{\sigma_{ltca}}, Rnd_{IK_{P_v^i}})$
- (12) $\text{PCA} : (\chi)_{\sigma_{pca}} \leftarrow \text{Sign}(Lk_{pca}, \chi)$
- (13) $\text{PCA} \rightarrow \text{RA} : (Id_{res}, (\chi)_{\sigma_{pca}}, N+1, t_{now})$
- (14) $\text{RA} : \text{Verify}(LTC_{pca}, \chi)$
- (15) $\text{RA} : (SN_{p_i}, tkt_{\sigma_{ltca}}, Rnd_{IK_{P_v^i}}) \leftarrow \chi$
- (16) $\text{RA} : \text{Verify}(LTC_{ltca}, tkt_{\sigma_{ltca}})$
- (17) $\text{RA} : H(Id_{PCA} || Rnd_{tkt}, IK_{tkt}, t_s^i, t_e^i, Exp_{tkt}) \leftarrow tkt$
- (18) $\text{RA} : H(IK_{tkt} || K_v^i || t_s^i || t_e^i || Rnd_{IK_{P_v^i}}) \stackrel{?}{=} IK_{P_v^i}$

This essentially enables efficient distribution of the CRL [57]: the PCA only needs to include one entry per batch of pseudonyms without compromising their unlinkability. Finally, the PCA issues the pseudonyms by signing it using its private key (steps 1.16–1.17) and delivers the response (step 1.19).

4.2.2 Pseudonym Issuance Validation Process (Protocol 2).

Upon receiving a request for misbehavior identification, e.g., multiple suspicious traffic congestion alerts sent to a traffic monitoring system, an entity could send a request to the RA to validate the pseudonym issuance process of a “suspicious” pseudonym (step 2.1–2.4). The RA validates the request and interacts with the corresponding PCA that issued the pseudonym, to provide evidence for the pseudonym issuance procedure; in fact, this process ensures that an actual vehicle requested the pseudonym by providing a valid ticket, also guarantees the PCA did not issue a pseudonym for an illegitimate vehicle (step 2.5–2.8).

Upon receiving the request, the PCA validates the request, and provides the corresponding ticket and $Rnd_{IK_{P_v^i}}$, used to issue the pseudonym. The response is signed by the PCA sent back to the RA (step 2.8–2.13). Upon receiving the response, the RA verifies it, facilitates validating the ticket using the public key of the LTCA, and checks $H(IK_{tkt} || K_v^i || t_s^i || t_e^i || Rnd_{IK_{P_v^i}}) \stackrel{?}{=} IK_{P_v^i}$ (step 2.14–2.18). If the hash calculation results in the same hash values, it confirms that the pseudonym has been issued based on a valid ticket, i.e., properly issued by the LTCA. Moreover, it ensures the PCA could not have issued the pseudonym for a *non-existing* vehicle. Note that upon performing pseudonym issuance validation process, the actual identity of a vehicle is not disclosed, i.e., user privacy is strongly protected. Further security and privacy analysis in Sec. 5.

4.3 Mitigating Sybil Attacks on the VPKIaaS

Multiple replicas of a micro-service interact with the same database to accomplish their operations, e.g., all replicas of LTCAs should

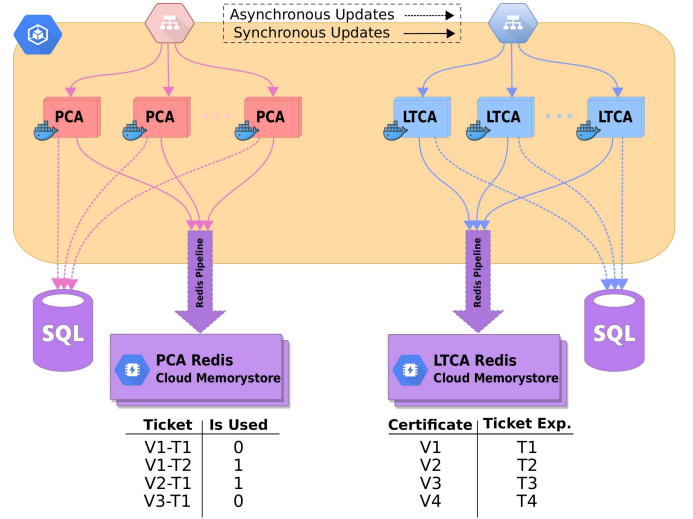


Figure 3: VPKIaaS Memorystore with Redis and MySQL.

interact with the same database to store information about tickets they issue. The same way, all replicas of PCAs interact with a single database to validate an authorization ticket and store information corresponding to issued pseudonyms. Micro-services could opt in to utilize their shared MySQL database either synchronously or asynchronously⁷. Asynchronous interaction of the micro-services and the shared database would result in efficient pseudonyms issuance. However, a malicious vehicle could repeatedly submit requests. If the micro-services do not synchronously validate tickets and pseudonym requests, one can obtain multiple sets of pseudonyms if the requests were delivered to different replicas. On the other hand, synchronous interaction of the micro-services and the shared database would prevent issuing multiple sets of pseudonym for a given requester, thus, eradicating the Sybil-based misbehavior. However, it would drastically diminish the performance of the system, notably timely on-demand issuance of pseudonyms. The performance of the relational database, e.g., MySQL, used in [60], can be highly degraded by synchronized interactions, e.g., [41]. Moreover, scaling out the Pods to handle a large volume of workload while relying on a single shared MySQL database becomes a *single point of failure*, questions the practicality of such a scheme (to be highly-available and dynamically-scalable).

In order to systematically mitigate the aforementioned vulnerability, we propose a hybrid design by considering two separate databases. Fig. 3 shows the Memorystore of the VPKIaaS: an in-memory key-value database as a service on GCP compatible with the Redis [14] protocol, and a relational database, e.g., MySQL. Each Pod of a micro-service synchronously interacts with the Redis database⁸ to validate a request towards thwarting Sybil attacks. Upon validating a request, the tickets and pseudonyms are issued and the corresponding information are stored in the relational database

⁷A synchronous interaction with a database implies enforcing limits on accessing to a resource by locking it to ensure the consistency of all transactions. An asynchronous interaction, though, implies that requests are proceeded without waiting to complete a transaction; the execution will happen later via an asynchronous callback function.

⁸Note that MySQL and Redis could both be single point of failures if not offered as a highly-available and dynamically-scalable service. However, a distributed cluster of MySQL will be a bottleneck in our scenario because relational databases are slow in nature, especially if the setup is synchronous. The Redis cluster, though, is an in-memory key-value database which offers very fast insertion and query.

Protocol 3 Ticket Request Validation (by the LTCA using Redis)

```
1: procedure VALIDATE_TICKET_REQ( $SN_{LTC}^i, tkt_{start}^i, tkt_{exp}^i$ )
2:   ( $value^i$ )  $\leftarrow$  RedisQuery( $SN_{LTC}^i$ )
3:   if  $value^i == NULL$  OR  $value^i <= tkt_{start}^i$  then
4:     RedisUpdate( $SN_{LTC}^i, tkt_{exp}^i$ )
5:      $Status \leftarrow IssueTicket(\dots)$   $\triangleright$  Invoking ticket issuance procedure
6:     if  $Status == False$  then
7:       RedisUpdate( $SN_{LTC}^i, value^i$ )  $\triangleright$  Reverting  $SN_{LTC}^i$  to  $value^i$ 
8:       return ( $False$ )  $\triangleright$  Ticket issuance failure
9:     else
10:      return ( $True$ )  $\triangleright$  Ticket issuance success
11:    end if
12:  else
13:    return ( $False$ )  $\triangleright$  Suspicious to Sybil attacks
14:  end if
15: end procedure
```

asynchronously. Such a hybrid design mitigates Sybil attacks without diminishing the overall performance of the pseudonym acquisition process: the time-consuming validation through the rational database is replaced by an efficient validation through the Redis database.

4.3.1 LTCA Sybil Attack Mitigation (Protocol 3). The LTCA, the *policy decision and enforcement point* in a domain, issues tickets with non-overlapping intervals, i.e., vehicles cannot obtain tickets with overlapping lifetime. Upon receiving a ticket request, each LTCA micro-service Pod should check if a ticket was issued to the requester during that period. Enforcing such a policy ensures that no vehicle would obtain more than a single valid ticket towards requesting multiple simultaneously valid pseudonyms. Furthermore, each ticket is implicitly bound to a specific PCA by the vehicle; as a result, the ticket cannot be used more than once or be used for other PCAs. Each LTCA micro-service Pod stores the serial number of the vehicle's LTC (as the key) and the expiration time of its current ticket (as the value) on the Redis database. Upon receipt of a new request for obtaining a ticket, each micro-service creates a Redis pipeline to validate the ticket (step 3.2). A Redis pipeline entails a list of commands guaranteed to be executed sequentially without interruption.

The Redis pipeline checks the existence of the serial number of an LTC in the database; if it exists, it validates if the request interval overlaps with the previously recorded entry (step 3.3); the request is marked to be *malicious* if the serial number exists in the database and the requested ticket start time (tkt_{start}) is less than the expiration time of the already existed ticket. Otherwise, the Redis pipeline updates the corresponding entry (or adds a new entry if not existed) with the new ticket expiration time (step 3.4). Then, the procedure for ticket issuance will be invoked (step 3.5, i.e., Protocol 6 in Appendix). In case of any failure during the ticket issuance, the ticket expiration value will be rolled back (steps 3.6–3.8). The Redis pipeline is executed on a single thread and it is guaranteed to sequentially execute the commands; thus, even if all replicas of the LTCA received a ticket request from the same vehicle, Redis ensures that only one ticket request will be served and the rest of them will be denied.

4.3.2 PCA Sybil Attack Mitigation (Protocol 4). The PCA issues pseudonyms with non-overlapping lifetimes in order to ensure that

Protocol 4 Pseudonym Request Validation (by the PCA using Redis)

```
1: procedure VALIDATE_PSEUDONYM_REQ( $SN_{tkl}^i$ )
2:   ( $value^i$ )  $\leftarrow$  RedisQuery( $SN_{tkl}^i$ )
3:   if  $value^i == NULL$  OR  $value^i == False$  then
4:     RedisUpdate( $SN_{tkl}^i, True$ )
5:      $Status \leftarrow IssuePsnym(\dots)$   $\triangleright$  Invoking pseudonym issuance
6:     if  $Status == False$  then
7:       RedisUpdate( $SN_{tkl}^i, False$ )  $\triangleright$  Reverting  $SN_{tkl}^i$  to  $False$ 
8:       return ( $False$ )  $\triangleright$  Pseudonym issuance failure
9:     else
10:      return ( $True$ )  $\triangleright$  Pseudonym issuance success
11:    end if
12:  else
13:    return ( $False$ )  $\triangleright$  Suspicious to Sybil attacks
14:  end if
15: end procedure
```

no vehicle is provided with more than one valid pseudonym at any given point in time. However, in order to fully eradicate Sybil-based misbehavior, the PCA micro-service should ensure that each ticket is used only once to issue a set of pseudonyms for a requester. In other words, the VPKIaaS system should ensure that different replicas of the PCA micro-service never issue more than a set of pseudonyms for a ticket. All replicas of the PCA share a Redis Memorystore with the ticket serial number (as the key) and a boolean data type (as the value). If the ticket serial number does not exist, or if it exists with a boolean data type value of false, the ticket was not used.

Upon receipt of a pseudonym acquisition request, each Pod of the PCA micro-service creates a Redis pipeline to validate the ticket (step 4.2). If the key (SN_{tkl}) does not exist or the value is false (step 4.3), Redis updates the database with the value of true and the procedure for issuing pseudonyms will be invoked (step 4.5, i.e., Protocol 1). In case of failure during the pseudonym acquisition process, the corresponding value for the ticket will be set to false in the Redis database, i.e., rolling back the transaction, to ensure the consistency of the pseudonym issuance procedure (steps 4.6–4.8). If the value corresponding to the key (SN_{tkl}) is true, the request for obtaining a set of pseudonyms should be denied (step 4.13).

5 QUALITATIVE ANALYSIS

A detailed security and privacy analysis on the requirements for VPKI entities can be found in [57, 60]. Here, we compile security and privacy analysis for deploying a VPKIaaS system on the cloud, and we discuss additional facts of the problem. A detailed description on secret management in the cloud can be found in Appendix.

5.1 Security and Privacy Analysis

Sybil-based misbehavior: A malicious vehicle could attempt to repeatedly request to obtain multiple tickets from the LTCA, and/or aggressively request multiple sets of pseudonyms from the PCA. However, all replicas of a micro-service share a Redis Memorystore to validate every request. Thus, any suspicious request can be *instantaneously* validated through the Redis Memorystore (without interacting with the MySQL, which would be relatively more time-consuming). Redis is executed on a single thread and the pipeline is guaranteed to sequentially execute the commands; thus, even

if all replicas of a micro-service, e.g., the PCA, received a pseudonym request from one vehicle at the same time, the VPKIaaS system would serve only one pseudonym request and the rest of them would be denied. Therefore, the VPKIaaS ensures an efficient ticket and pseudonym provisioning while preventing any vehicle from obtaining multiple tickets or sets of pseudonyms towards a Sybil-based misbehavior. The ramification of the Redis service failure depends on the action taken after the failure, i.e., *fail open* or *fail close*. In case of fail open, Sybil attacks would be possible, as the VPKIaaS system would provide vehicles with spurious pseudonyms. Later, it invalidates the erroneously issued credentials by adding them to the CRL. In case of fail close, the VPKIaaS system stops issuing credentials until the failure gets resolved.

Alternatively, a single deviant PCA could issue multiple simultaneously valid pseudonyms for a given vehicle, or issue pseudonyms for an entity without any valid ticket issued by the LTCA. However, upon performing pseudonym validation process, the RA requests the corresponding PCA to validate a pseudonym. Each pseudonym requires to have a valid pseudonym identifiable key (IK_{p_i}). Thus, a malicious PCA can be identified and would then be evicted from the VPKI system if it issued a pseudonym without a valid ticket provided. Note that when performing the pseudonym issuance validation process, the actual identity of the pseudonym owner is not disclosed to the PCA or the RA, i.e., user privacy is preserved. Moreover, no entity can infer user sensitive information by continuously conducting pseudonym issuance validation process towards harming user privacy. We emphasize here that our VPKIaaS scheme does not prevent a malicious PCA from issuing multiple sets of fake pseudonyms; rather, our scheme facilitates efficient identification of a misbehaving PCA by cross-checking the pseudonym issuance procedure in a privacy-preserving manner. To ensure correct operation of a micro-service, each Pod frequently requests a dummy ticket or pseudonym. Since such operations are executed in isolation within the Pod, the issued dummy tickets and pseudonyms cannot leave the Pod. Moreover, each issued pseudonym can be cross-checked towards identifying suspicious compromised entity.

DDoS attacks on the VPKIaaS system: Compromised internal entities or external adversaries could try to harm the system operations by launching a DoS (or a DDoS) attack, thus degrading the availability of the system. A rate limiting mechanism prevents them from compromising the availability of the system; moreover, the system flags misbehaving users, thus evicting them from the system. External adversaries could launch a DDoS attack by clogging the LTCA with fake certificates, or the PCA with bogus tickets. In fact, such misbehaving entities attempt to compromise the availability of the VPKI entities by mandating them to excessively validate the signature of fake LTCs or bogus tickets, i.e., performing a signature flooding attack [50].

We achieve high-availability and fault-tolerance in the face of a benign failure by exploiting the Kubernetes master to kill the running (faulty) Pod, e.g., in case of system faults or crashes, and create a new Pod. In case of resource depletion attacks, the Kubernetes master scales out the Pods to handle such demanding loads. At the same time, a puzzle technique, e.g., [29, 33], can be employed as a mitigation approach, e.g., [60]: each vehicle is mandated to visit

a pre-defined set of Pods, in a pre-determined sequential order to solve a puzzle. As a result, the power of an attacker is degraded to the power of a legitimate client, thus, an adversary cannot send high-rate spurious requests to the VPKI. On the side of the infrastructure, there are DDoS mitigation techniques at different network layers, provided by various cloud service providers.

Synchronization among the VPKI entities: Lack of synchronization between the LTCA and the PCA could affect the pseudonym issuance process, e.g., a PCA would not issue pseudonyms for a seemingly ‘expired’ ticket. However, mildly drifting clocks of the VPKI entities can hardly affect the operation, because the pseudonym lifetimes and periods for pseudonym refills (Γ) are in the order of minutes, typically. It suffices to have VPKI entities periodically synchronizing their clocks. For example, if the accuracy of an Real Time Clock (RTC) is 50 parts-per-million (ppm), i.e., 50×10^{-6} , and the maximum accepted error in timestamp is 50 ms, then each entity should synchronize its clock every 16 minutes ($\frac{50 \times 10^{-3} \text{sec}}{50 \times 10^{-6} \text{ppm}}$).

6 QUANTITATIVE ANALYSIS

Experimental setup: We leveraged a state-of-the-art VPKI system [60] and restructured its source code to fit in a micro-services architecture, e.g., through containerization, automation, bootstrapping of services. We further added health and load metric publishing features, to be used by an orchestration service to scale in/out accordingly. We built and pushed Docker images for LTCA, PCA, RA, MySQL, and Locust [18], an *open source load testing tool*, to the Google Container Registry [9]. Isolated namespaces and deployment configuration files are defined before Google Kubernetes Engine (GKE) v1.10.11 [10] cluster runs the workload. We configured a cluster of five Virtual Machines (VMs) (n1-highcpu-32), each with 32 vCPUs and 28.8GB of memory. The implementation is in C++ and we use FastCGI [49] to interface Apache web-server. We use XML-RPC [16] to execute a remote procedure call on the cloud. The VPKIaaS interface is language-neutral and platform-neutral, as we use Protocol Buffers [11] for serializing and de-serializing structured data. For the cryptographic protocols and primitives (ECDSA and TLS), we use OpenSSL with ECDSA-256 key pairs according to the ETSI (TR-102-638) [45] and IEEE 1609.2 [51] standards; other algorithms and key sizes are compatible in our implementation.

To facilitate the deployment of the VPKIaaS, we created all VPKIaaS configuration in YAML language [17], applicable to deploy on any cloud provider which offers *Kubernetes As A Service*, e.g., GCP [22] and Amazon Web Service (AWS) (aws.amazon.com). For our experiments, we deployed our VPKIaaS on the GKE. We also used other GCP services: *Memorystore* [20], *Prometheus* [26], and *Grafana* [23]. The Memorystore service is a Redis-compatible [14] service which acts as in-memory key-value data store (see Fig. 3). Prometheus is a feature-rich metric service which collects all the metrics of the Kubernetes cluster and the applications running on it into a time-series database. We use Grafana to visualize the metrics collected by Prometheus and monitor the *system under test*. Prometheus and Grafana are deployed as prepared applications from the GCP marketplace [27] on the Kubernetes cluster. Moreover, we leveraged Locust [18], deployed on the Kubernetes cluster, to synthetically generate a large volume of pseudonym requests.

Table 2: Experiment Parameters.

Parameters	Config-1	Config-2
total number of vehicles	1000	100, 50,000
hatch rate	1	1, 100
interval between requests	1000-5000 ms	1000-5000 ms
pseudonyms per request	100, 200, 300, 400, 500	100, 200, 500
LTCA memory request	128 MiB	128 MiB
LTCA memory limit	256 MiB	256 MiB
LTCA CPU request	500 m	500 m
LTCA CPU limit	1000 m	1000 m
LTCA HPA	1-40; CPU 60%	1-40; CPU 60%
PCA memory request	128 MiB	128 MiB
PCA memory limit	256 MiB	256 MiB
PCA CPU request	700 m	700 m
PCA CPU limit	1000 m	1000 m
PCA HPA	1-120; CPU 60%	1-120; CPU 60%

Metrics: To evaluate the performance of our VPKIaaS system, we measure the latency to obtain pseudonyms under different scenarios and configurations for a large-scale mobile environment. More specifically, we evaluate the performance of the system with (and without) flash crowds to illustrate its *high-availability*, *robustness*, *reliability*, and *dynamic-scalability*. We demonstrate the performance of our VPKIaaS system by emulating a large volume of synthetic workload. Table 2 shows the configurations used in our experiments, with *Config-1* referring to a ‘normal’ vehicle arrival rate and *Config-2* for a *flash crowd* scenario. Experiments with *Config-1* indicates that every 1-5 seconds, a new vehicle joins the system and requests a batch of 100-500 pseudonyms. To emulate a flash crowd scenario, i.e., *Config-2*, beyond having vehicles joining the system based on *Config-1*, 100 new vehicles join the system every 1-5 seconds and request a batch of 100-200 pseudonyms.

Remark: Assuming the pseudonyms are issued with non-overlapping intervals (important to mitigate Sybil-based misbehavior), obtaining 100 and 500 pseudonyms per day implies pseudonyms lifetimes of 14.4 minutes ($\tau_P = 14.4$ min.) or 3 minutes ($\tau_P = 172.8$ sec), respectively. According to actual large-scale urban vehicular mobility dataset, e.g., Tapas-Cologne [86] or LuST [40], the average trip duration is within 10-30 minutes. Moreover, according to the US DoT, the average daily commute time in the US is around 1 hour [1]. Thus, requesting 100 pseudonyms per day would cover 24 hours trip duration with each pseudonym lifetime of approx. 15 minutes. We evaluate the performance of our VPKIaaS system under such seemingly extreme configurations.

6.1 Large-scale Pseudonym Acquisition

Fig. 4.a illustrates the Cumulative Distribution Function (CDF) of the single ticket issuance processing delay (executed based on Config-1 in Table 2); as illustrated, 99.9% of ticket requests are served within 24 ms: $F_x(t = 24 \text{ ms}) = 0.999$, i.e., $Pr\{t \leq 24 \text{ ms}\} = 0.999$. Fig. 4.b shows the CDF of processing latency for issuing pseudonyms with different batches of pseudonyms per request as a parameter. For example, with a batch of 100 pseudonyms per request, 99.9% of the vehicles are served within less than 77 ms ($F_x(t = 77 \text{ ms}) = 0.999$). Even with a batch of 500 pseudonyms per request, the VPKIaaS system can efficiently issue pseudonyms:

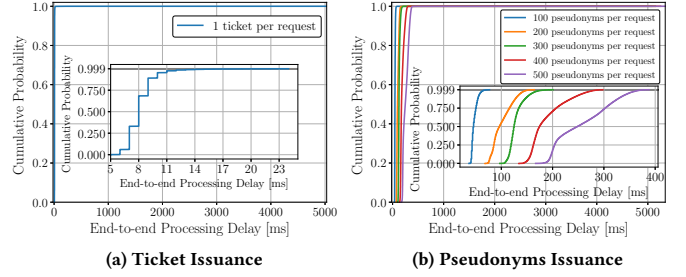


Figure 4: (a) CDF of end-to-end latency to issue a ticket. (b) CDF of end-to-end processing delay to issue pseudonyms.

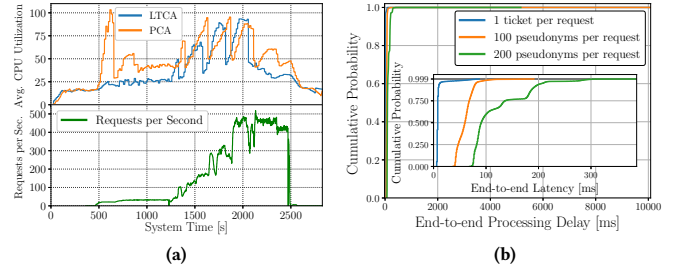


Figure 5: VPKIaaS system in a flash crowd load situation. (a) CPU utilization and the number of requests per second. (b) CDF of processing latency to issue tickets and pseudonyms.

$F_x(t = 388 \text{ ms}) = 0.999$. The results confirm that the VPKIaaS scheme is efficient and scalable: the pseudonym acquisition process incurs low latency and it efficiently issues pseudonyms for the requesters.

6.2 VPKIaaS with Flash Crowd Load Pattern

Fig. 5 shows the performance of the VPKIaaS when a surge in pseudonym acquisition requests happens to the VPKIaaS (executed based on Config-2 in Table 2, with 100 pseudonyms per request for Fig. 5.a). We assess CPU utilization of the LTCA and the PCA Pods (Fig. 5.a top) and the total number of pseudonyms requests per second (Fig. 5.a bottom). When the number of requests per second increases, the average CPU utilization would rise; however, when CPU utilization hits 60% threshold, defined in the Horizontal Pod Autoscalers (HPAs) [24], the LTCA and the PCA deployment would horizontally scale to handle demanding loads, thus the average CPU utilization drops upon scaling out.

Fig. 5.b shows the end-to-end processing latency to obtain tickets and a batch of 100 or 200 pseudonyms in a flash crowd situation. The processing latency to issue a single ticket is: $F_x(t = 87 \text{ ms}) = 0.999$; to issue a batch of 100 pseudonyms per request, the processing latency is: $F_x(t = 192 \text{ ms}) = 0.999$. In comparison with processing delay under ‘normal’ conditions (Fig. 4), the processing latency of issuing a single ticket increases from 24 ms to 87 ms; the processing latency to issue a batch of 100 pseudonyms increased from 77 ms to 192 ms. Thus, even under such a highly demanding request rate, the VPKIaaS system issues credentials efficiently.⁹

⁹The total number of vehicles requesting 100 pseudonyms (under Config-2 in Table 2) is 398,870 and the VPKIaaS system issued approximately 40 millions pseudonyms within 2,500 seconds; with such an arrival rate, the VPKIaaS system would issue 0.5×10^{12} pseudonyms per year. Obviously, this number is lower than the one

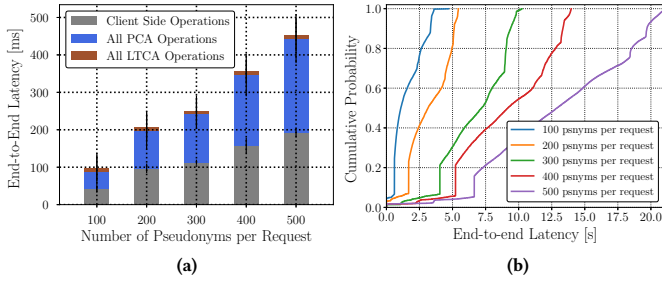


Figure 6: VPKIaaS system with flash crowd load pattern. (a) Average end-to-end latency to obtain pseudonyms. (b) CDF of end-to-end latency, observed by clients.

Fig. 6.a shows the latency for each system component to obtain different batches of pseudonyms per request (Config-2 in Table 2). Our VPKIaaS system outperforms prior work [38]: the processing delay to issue 100 pseudonym for [38] is approx. 2010 ms, while it is approx. 56 ms in our system, i.e., achieving a 36-fold improvement over prior work [38]. Fig. 6.b illustrates the average end-to-end latency to obtain pseudonyms, observed by clients. As we can see, during a surge of requests, *all* vehicles obtained a batch of 100 pseudonyms within less than 4,900 ms (including the networking latency). Obviously, the shorter the pseudonym lifetime, the higher the workload on the VPKI, thus the higher the end-to-end latency. Note that serving requests under a flash crowd scenario at this rate (Config-2 in Table 2) implies that our VPKIaaS system would serve 720,000 vehicles joining the system within an hour. Thus, even under such flash crowd load pattern, our VPKIaaS system can comfortably handle such a high demand of requests.

6.3 Dynamic-scalability of the VPKIaaS

In this scenario, we demonstrate the performance of our VPKIaaS system, notably its reliability and dynamic scalability. To emulate a large volume of workload, we generated synthetic workload using 30 containers, each with 1 vCPU and 1GB of memory (executed based on Config-2 in Table 2). Fig. 7.a shows the average CPU utilizations of the LTCA and PCA Pods (observed by HPA) as well as the total number of requests per second. Fig. 7.b shows how our VPKIaaS system dynamically scales out or scales in according to the rate of pseudonyms requests. The numbers next to the arrows show the number of LTCA and PCA Pod replicas at any specific system time. As illustrated, the number of PCA Pods starts from 1 and it gradually increases; at system time 1500, there is a surge in pseudonym requests, thus the number of PCA Pods increased to 80. Note that issuing a ticket is more efficient than issuing pseudonyms; thus, the LTCA micro-service scaled out only up to 4 Pod replicas.

6.4 VPKIaaS Performance Comparison

We compare our VPKIaaS scheme with a *baseline* scheme [38], which implements a VPKI according to the ETSI architecture. More precisely, each vehicle requests pseudonyms from an authorization authority; the request is forwarded to the enrollment authority to

mentioned in Sec. 1, i.e., 1.5×10^{12} . Note that this is a proof of concept of the implementation and evaluation of the VPKIaaS system; by allocating more resources and increasing the pseudonym request rates, the VPKIaaS system would issue even further pseudonyms.

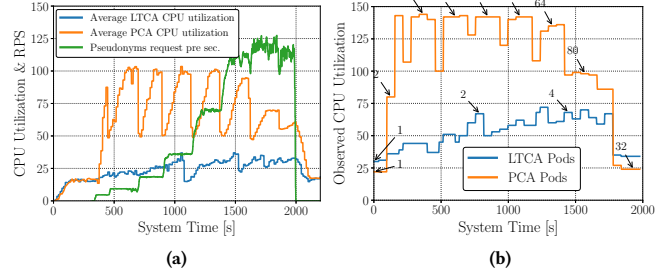


Figure 7: Each vehicle requests 500 pseudonyms (CPU utilization observed by HPA). (a) Number of active vehicles and CPU utilization. (b) Dynamic scalability of VPKIaaS system.

check and validate the request. Upon a successful validation, the authorization authority issues the pseudonyms and sends them back to the vehicle. Using the similar setup to have a meaningful and direct comparison, we achieve a 36-fold improvement over the baseline scheme: under normal conditions, the processing delay to issue 100 pseudonyms for the baseline scheme is approx. 2010 ms, while it is approx. 56 ms in our VPKIaaS system. Even under a flash crowd scenario (based on Config-2), the processing delay to issue 100 pseudonyms is approx. 71 ms, i.e., 28-fold improvement. Furthermore, unlike the VPKI system in [38], our implementation supports dynamic scalability, i.e., the VPKI scales out, or scales in, based on the arrival rate of pseudonyms requests.

Moreover, in order to handle a large volume of workload, SECMACE [60] requires to statically allocate resources to the VPKI. In case of an unpredictable surge in the arrival rates or being under a DDoS attack, the performance of SECMACE would drastically decrease. Furthermore, when deploying SECMACE on the cloud, a malicious vehicle could repeatedly request to obtain pseudonyms towards performing Sybil-based misbehavior. On the contrary, our VPKIaaS system can comfortably handle requests with unexpected arrival rate while being *efficient* in issuing pseudonyms, being *resilient* against Sybil and resource depletion attacks, and being cost-effective by systematically allocating and deallocating resources.

7 CONCLUSION

Paving the way for the deployment of a secure and privacy-preserving VC system relies on deploying a special-purpose VPKI. However, its success requires extensive experimental evaluation, to ensure viability (in terms of performance and cost). We leverage a state-of-the-art VPKI, enhance its functionality, and migrate it into the GCP to illustrate its availability, resiliency, and scalability towards a cost-effective VPKI deployment. Through extensive security and privacy analysis, we show that the VPKIaaS system fully eradicates Sybil-based misbehavior without compromising the efficiency of the pseudonym acquisition process. All these investigations would catalyze the deployment of the central building block of secure and privacy-preserving VC systems.

ACKNOWLEDGEMENT

Work supported by the Swedish Foundation for Strategic Research (SSF) SURPRISE project and the KAW Academy Fellowship Trustworthy IoT project.

REFERENCES

- [1] 2014. V2V Communications: Readiness of V2V Technology for Application. National Highway Traffic Safety Administration, DOT HS 812 014.
- [2] 2016. Vehicle Safety Communications Security Studies: Technical Design of the Security Credential Management System. <https://bit.ly/2CA1WbV>.
- [3] 2018. AWS Certificate Manager. <https://aws.amazon.com/certificate-manager/>.
- [4] 2018. AWS CloudTrail. <https://aws.amazon.com/cloudtrail/>.
- [5] 2018. Cloud Identity & Access Management. <https://cloud.google.com/iam/>.
- [6] 2018. Comodo Certification Authority. <https://ssl.comodo.com/>.
- [7] 2018. FIPS 140-2 Level 3 Non-Proprietary Security Policy. <https://bit.ly/2R1XUeH>.
- [8] 2018. GCP Cloud Audit Logging. <https://cloud.google.com/logging/docs/audit/>.
- [9] 2018. Google Container Registry. <https://cloud.google.com/container-registry/>.
- [10] 2018. Google Kubernetes Engine v1.9.6. <https://bit.ly/2I8MjJx>.
- [11] 2018. Google Protocol Buffer. <https://bit.ly/1mlSy49>. Accessed April 25, 2018.
- [12] 2018. Kubernetes Namespaces. <https://bit.ly/2DjOw5d>.
- [13] 2018. Let's Encrypt Stats. <https://letsencrypt.org/>.
- [14] 2018. Redis, In-memory Data Structure Store, Used as a Database. <https://redis.io/>.
- [15] 2018. Symantec SSL/TLS Certificates. <https://symc.ly/2Mp8Mpe>.
- [16] 2018. XML-RPC for C/C++. <https://bit.ly/2R0pMCS>. Accessed April 25, 2018.
- [17] 2018. YAML API Reference. <https://learn.getgrav.org/advanced/yaml>.
- [18] 2019. An Open Source Load Testing Tool. <https://locust.io/>.
- [19] 2019. AWS CloudHSM. <https://aws.amazon.com/cloudhsm/>.
- [20] 2019. Cloud Memorystore. <https://cloud.google.com/memorystore/>.
- [21] 2019. Google Cloud HSM. <https://cloud.google.com/hsm/>.
- [22] 2019. Google Cloud Platform. <https://cloud.google.com/gcp/>.
- [23] 2019. Grafana. <https://grafana.com/>.
- [24] 2019. Horizontal Pod Autoscaler. <https://bit.ly/2Q8Ri1u>.
- [25] 2019. Kubernetes: Production-Grade Container Orchestration. kubernetes.io/
- [26] 2019. Prometheus. <https://prometheus.io/>.
- [27] 2019. Prometheus & Grafana: Google Cloud Marketplace. <https://bit.ly/2Q8Ri1u>.
- [28] K. Abboud et al. 2016. Interworking of DSRC and Cellular Network Technologies for V2X Communications: A Survey. *IEEE TVT* 65, 12 (July 2016), 9457–9470.
- [29] M. Abliz and T. Znati. 2009. A Guided Tour Puzzle for Denial of Service Prevention. In *IEEE ACSAC*. Honolulu, HI, 279–288.
- [30] M. Agiwal et al. 2016. Next Generation 5G Wireless Networks: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 1617–1655.
- [31] J. G. Andrews et al. 2014. What Will 5G Be? *IEEE JSAC* 32, 6 (2014), 1065–1082.
- [32] I. Ari et al. 2003. Managing Flash Crowds on the Internet. In *IEEE/ACM MAS-COTS*. Orlando, FL, USA, 246–249.
- [33] T. Aura et al. 2001. DoS-Resistant Authentication with Client Puzzles. In *Proceedings of Security Protocols Workshop*. New York, USA.
- [34] N. Bißmeyer. 2014. *Misbehavior Detection and Attacker Identification in Vehicular Ad-Hoc Networks*. Ph.D. Dissertation. Technische Universität.
- [35] G. Calandriello et al. 2007. Efficient and Robust Pseudonymous Authentication in VANET. In *ACM VANET*. New York, USA, 19–28.
- [36] G. Calandriello et al. 2011. On the Performance of Secure Vehicular Communication Systems. *IEEE TDSC* 8, 6 (Nov. 2011), 898–912.
- [37] L. J. Carnahan and M. E. Smid. 1994. *Security Requirements for Cryptographic Modules*. Technical Report.
- [38] P. Cincilla et al. 2016. Vehicular PKI Scalability-Consistency Trade-Offs in Large Scale Distributed Scenarios. In *IEEE VNC*. Columbus, Ohio, USA.
- [39] J. Clark and et al. 2013. SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements. In *IEEE SnP*. Berkeley, USA.
- [40] L. Codeca et al. 2015. Luxembourg SUMO Traffic (LuST) Scenario: 24 Hours of Mobility for Vehicular Networking Research. In *IEEE VNC*. Kyoto, Japan.
- [41] Brian F Cooper et al. 2010. Benchmarking Cloud Serving Systems with YCSB. In *ACM SoCC*. Indianapolis, Indiana, USA, 143–154.
- [42] T. Dierks. 2008. The transport layer security protocol version 1.2. (Aug. 2008).
- [43] J. R Douceur. 2002. The Sybil Attack. In *ACM Peer-to-peer Systems*. London, UK.
- [44] L. Dykciik et al. 2018. BlockPKI: An Automated, Resilient, and Transparent Public-Key Infrastructure. *arXiv preprint arXiv:1809.09544* (Sep. 2018).
- [45] ETSI. 2009. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions.
- [46] L. Fischer et al. 2006. Secure Revocable Anonymous Authenticated Inter-vehicle Communication (SRAAC). In *ESCAR*. Berlin, Germany.
- [47] Philippe Golle, Dan Greene, and Jessica Staddon. 2004. Detecting and correcting malicious data in VANETs. In *ACM VANET*. Philadelphia, PA, USA, 29–37.
- [48] D. Goodin. 2011. New Hack on Comodo Reseller Exposes Private Data.
- [49] Paul Heinlein. 1998. FastCGI. *Linux journal* 1998, 55es (1998), 1.
- [50] H-C. Hsiao et al. 2011. Flooding-Resilient Broadcast Authentication for VANETs. In *ACM Mobile Computing and Networking*. Las Vegas, Nevada, USA.
- [51] IEEE-1609.2. 2016. IEEE Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages. (Mar. 2016).
- [52] H. Jin, M. Khodaei, and P. Papadimitratos. 2016. Security and Privacy in Vehicular Social Networks. In *Vehicular Social Networks*. Taylor & Francis Group.
- [53] Antonio K. [n. d.]. Security Architecture and Mechanisms for V2V/V2I, SeVe-Com.
- [54] M. Khodaei et al. 2014. Towards Deploying a Scalable & Robust Vehicular Identity and Credential Management Infrastructure. In *IEEE VNC*. Paderborn, Germany.
- [55] M. Khodaei et al. 2016. Evaluating On-demand Pseudonym Acquisition Policies in Vehicular Communication Systems. In *ACM IoV-Vol*. Paderborn, Germany.
- [56] M. Khodaei et al. 2017. RHyTHM: A Randomized Hybrid Scheme To Hide in the Mobile Crowd. In *IEEE VNC*. Torino, Italy.
- [57] M. Khodaei et al. 2018. Efficient, Scalable, and Resilient Vehicle-Centric Certificate Revocation List Distribution in VANETs. In *ACM WiSec*. Stockholm, Sweden.
- [58] M. Khodaei et al. 2018. Poster: Mix-Zones Everywhere: A Dynamic Cooperative Location Privacy Protection Scheme. In *IEEE VNC*. Taipei, Taiwan.
- [59] M. Khodaei et al. 2018. POSTER: Privacy Preservation through Uniformity. In *ACM WiSec*. Stockholm, Sweden, 279–280.
- [60] M. Khodaei et al. 2018. SECMAACE: Scalable and Robust Identity and Credential Management Infrastructure in Vehicular Communication Systems. *IEEE TITS* 19, 5 (May 2018), 1430–1444.
- [61] M. Khodaei and P. Papadimitratos. 2015. The Key to Intelligent Transportation: Identity and Credential Management in Vehicular Communication Systems. *IEEE Vehicular Technology Magazine* 10, 4 (Dec. 2015), 63–69.
- [62] T H-J. Kim et al. 2013. Accountable Key Infrastructure (AKI): A Proposal for a Public-key Validation Infrastructure. In *ACM WWW*. Rio de Janeiro, Brazil.
- [63] Virendra Kumar et al. 2017. Binary Hash Tree based Certificate Access Management for Connected Vehicles. In *ACM WiSec*. Boston, USA.
- [64] John Leyden. 2011. Inside 'Operation Black Tulip': DigiNotar hack analysed. <https://bit.ly/2REVJ8Q>.
- [65] Zhendong Ma et al. 2008. Pseudonym-on-demand: A New Pseudonym Refill Strategy for Vehicular Communications. In *IEEE VTC*. Calgary, BC, 1–5.
- [66] Neil McAllister. 2013. Browser makers rush to block fake Google.com security cert. <https://bit.ly/2QXANoo>.
- [67] Patrick McDaniel and Aviel Rubin. 2000. A Response to "Can We Eliminate Certificate Revocation Lists?". In *FC (Springer)*. Berlin, Heidelberg, 245–258.
- [68] H. Noroozi et al. 2018. DEMO: VPKIaaS: A Highly-Available and Dynamically-Scalable Vehicular Public-Key Infrastructure. In *ACM WiSec*. Stockholm, Sweden.
- [69] P. Papadimitratos. 2008. "On the road" - Reflections on the Security of Vehicular Communication Systems. In *IEEE ICVES*. Columbus, OH, USA.
- [70] Panagiotis Papadimitratos et al. 2006. Securing Vehicular Communications-Assumptions, Requirements, and Principles. In *ESCAR*. Berlin, Germany.
- [71] Panagiotis Papadimitratos et al. 2007. Architecture for Secure and Private Vehicular Communications. In *IEEE ITST*. Sophia Antipolis, 1–6.
- [72] P. Papadimitratos et al. 2008. Impact of Vehicular Communication Security on Transportation Safety. In *IEEE INFOCOM MOVE*. Phoenix, AZ, USA, 1–6.
- [73] Panagiotis Papadimitratos et al. 2008. Secure Vehicular Communication Systems: Design and Architecture. *IEEE CommMag* 46, 11 (Nov. 2008), 100–109.
- [74] P. Papadimitratos et al. 2009. Vehicular Communication Systems: Enabling Technologies, Applications, and Future Outlook on Intelligent Transportation. *IEEE Communications Magazine* 47, 11 (Nov. 2009), 84–95.
- [75] PKI-Memo. 2011. C2C-CC. <http://www.car-2-car.org/>.
- [76] PRESERVE-Project. 2015. www.preserve-project.eu/.
- [77] Maxim Raya et al. 2007. Eviction of Misbehaving and Faulty Nodes in Vehicular Networks. *IEEE JSAC* 25, 8 (Oct. 2007), 1557–1568.
- [78] Steffen Reidt et al. 2009. The Fable of the Bees: Incentivizing Robust Revocation Decision Making in Ad Hoc Networks. In *ACM CCS*. Chicago, Illinois, US.
- [79] Eric Rescorla et al. 2012. Datagram Transport Layer Security V.1.2. (Jan. 2012).
- [80] Sushmita Ruj and others Cavenaghi. 2011. On Data-Centric Misbehavior Detection in VANETs. In *IEEE VTC*. San Francisco, CA, USA, 1–5.
- [81] F. Schaub, F. Kargl, Z. Ma, and M. Weber. 2010. V-tokens for Conditional Pseudonymity in VANETs. In *IEEE WCNC*. Sydney, Australia.
- [82] J. Sermersheim. 2006. Lightweight Directory Access Protocol (LDAP). (2006).
- [83] R. Shokri et al. 2014. Hiding in the Mobile Crowd: Location Privacy through Collaboration. *IEEE TDSC* 11, 3 (May 2014), 266–279.
- [84] Marcos Simplicio et al. 2018. ACPC: Efficient Revocation of Pseudonym Certificates using Activation Codes. *Elsevier Ad Hoc Networks* (July 2018).
- [85] E. Topalovic et al. 2012. Towards Short-lived Certificates. *IEEE Oakland Web 2.0 Security and Privacy (W2SP)* (May 2012).
- [86] S. Uppoor et al. 2014. Generation and Analysis of a Large-scale Urban Vehicular Mobility Dataset. *IEEE TMC* 13, 5 (May 2014), 1061–1075.
- [87] C. Vaas et al. 2018. Nowhere to Hide? Mix-Zones for Private Pseudonym Change using Chaff Vehicles. In *IEEE VNC*. Taipei, Taiwan.
- [88] W. Whyte, A. Weimerskirch, V. Kumar, and T. Hehn. 2013. A Security Credential Management System for V2V Communications. In *IEEE VNC*. Boston, MA.
- [89] B. Wiedersheim et al. 2010. Privacy in Inter-vehicular Networks: Why Simple Pseudonym Change is not Enough. In *WONS*. Kranjska Gora, Slovenia, 176–183.

APPENDIX

Protocol 5 Ticket Request from the LTCA (by the vehicle)

```

1: procedure REQTicket( $t_s, t_e$ )
2:    $Rnd_{tk_t} \leftarrow GenRnd()$ 
3:    $\zeta \leftarrow (Id_{req}, H(Id_{PCA} || Rnd_{tk_t}), t_s, t_e)$ 
4:    $(msg)_{\sigma_v} \leftarrow Sign(Lk_v, \zeta)$ 
5:   return  $((msg)_{\sigma_v}, LTC_v, N, t_{now})$ 
6: end procedure

```

Protocol 6 Issuing a Ticket (by the LTCA)

```

1: procedure ISSUETicket( $(msg)_{\sigma_v}, LTC_v, N, t_{now}$ )
2:   Verify( $LTC_v, (msg)_{\sigma_v}$ )
3:    $Rnd_{IK_{tk_t}} \leftarrow GenRnd()$ 
4:    $IK_{tk_t} \leftarrow H(LTC_v || t_s || t_e || Rnd_{IK_{tk_t}})$ 
5:    $\zeta \leftarrow (SN, H(Id_{PCA} || Rnd_{tk_t}), IK_{tk_t}, t_s, t_e, Exp_{tk_t})$ 
6:    $(tk_t)_{\sigma_{ltca}} \leftarrow Sign(Lk_{ltca}, \zeta)$ 
7:   return  $(Id_{res}, (tk_t)_{\sigma_{ltca}}, Rnd_{IK_{tk_t}}, N + 1, t_{now})$ 
8: end procedure

```

Ticket Acquisition Process (Protocols 5 and 6). Assume the OBU decides to obtain pseudonyms from a specific PCA. It first interacts with its H-LTCA to obtain a valid ticket. To conceal the actual identity of its desired PCA from the LTCA, it calculates the hash value of the concatenation of the specific PCA identity with a random number¹⁰ (steps 5.1–5.2). The vehicle prepares the request and signs it under the private key corresponding to its LTC (step 5.3–5.4) before returning the ticket request (step 5.5). It will then interact with the LTCA over a bidirectional authenticated TLS.

Upon reception of the ticket request, the LTCA verifies the LTC (thus authenticating and authorizing the requester) and the signed message (step 6.2). The LTCA generates a random number ($Rnd_{IK_{tk_t}}$) and calculates the “ticket identifiable key” (IK_{tk_t}) to bind the ticket to the LTC as: $H(LTC_v || t_s || t_e || Rnd_{IK_{tk_t}})$ (steps 6.3–6.4); this prevents a compromised LTCA from mapping a different LTC during the resolution process. The LTCA then encapsulates (step 6.5), signs (step 6.6), and delivers the response (step 6.7).

Secret Management

Secret management is a concern towards deploying services in the cloud. Passwords, secret keys, and private keys cannot be simply integrated (hard-coded) into the services, e.g., the source code or the configuration files. Having services deployed on the cloud, each service fetches the needed secrets according to role-based access control. In this section, we review best practices, provided by cloud service providers. Note that deploying services on the cloud typically implies trusting cloud service providers, notably in terms of secret management.

Amazon Web Service (AWS): AWS offers several services regarding secret management on the cloud. The most common service is Key Management Service (KMS), which offers a key management service on FIPS 140-2 validated HSMs [19] as the way to create, import, store, and rotate keys within AWS. The KMS of the AWS

only supports Advanced Encryption Standard (AES)-256. Through role-based access control policies for key management, one can be ensured that the secret key is only accessible by the authorized service, which initiated the process. As the applications and services will fetch the secrets from the KMS whenever they need, changing the secret key will not affect the operations of the services because they will fetch a new secret in the next iteration. The KMS provides automatic secret rotation, which can be enabled by the service. KMS can also be integrated with CloudTrail [4], logging access to the secret keys. CloudTrail logs must be configured with proper actions, besides raising an alarm in case of suspicious activities, e.g., rotating the key if illegitimate access to the secret key. Beyond KMS, there are other services for secret management, specifically designed to hold secret strings for the use in Relational Database Service (RDS) services of AWS. AWS also offers AWS Certificate Manager (ACM), providing a traditional certificate management [3].

Google Cloud Platform (GCP): GCP offers a key management service similar to AWS. Unlike AWS, GCP supports various cryptographic algorithms and primitives, e.g., AES-256, RSA-2048, RSA-3072, RSA-4096, Elliptic Curve Cryptography (ECC)-P256, and ECC-P384. In order to protect the secrets according to compliance standards, e.g., Federal Information Processing Standard (FIPS) 140-2, the KMS can be integrated with the HSM [21] service provided by the cloud service provider according to the FIPS 140-2 Level 3 [7, 37]. Accessing the secret keys can be restricted using Identity & Access Management (IAM) policies [5]. The cloud IAM service facilitates fine-grained access control to a service, e.g., defining a role to enable encryption using a certain KMS service and assigning the role to a specific (*authorized*) micro-service. Thus, the system ensures that only the specified micro-service can access the KMS instance without being able to access other cryptographic materials. Similar to CloudTrail in AWS, the GCP provides an Audit Logging Service [8] in order to monitor activities, e.g., accessing the data, as well as logging the system events for auditing purposes.

Kubernetes: Kubernetes is an orchestration service, responsible for orchestrating micro-services. Secret management in Kubernetes is different from the ones that cloud providers would offer. Kubernetes offers a secret management system for micro-services. Thus, a micro-service can leverage only the secret management system within the Kubernetes, or alternatively, it can interact with the secret management services offered by the cloud service provider, in which the Kubernetes instances operate.

Secret Management in VPKIaaS: In order to offer a cloud agnostic solution for the VPKIaaS system, the Kubernetes secret volume suits our solution the best. However, the contents of the volume are encrypted using the KMS of the cloud service provider. During the bootstrapping phase, each Pod of a micro-service fetches its encrypted private key from its local volume; it then queries the KMS of the cloud provider to decrypt the private key (according to role-based access control). To protect the secrets, i.e., the key-pairs used by the VPKI entities, each micro-service leverages its own secret volume in its own namespace [12]. A namespace is an isolated environment with all classes of elements operating in it. However, to protect the secret volumes, the keys can be encrypted using the KMS of the cloud service provider, depending on the choice of deployment.

¹⁰The storage cost for these random numbers is reasonably cheap, e.g., 264 million vehicles with average trip duration of 1 hour require 32 GB per day (25\$ per month).