Authenticity of LiDAR Points in Autonomous Vehicle "Certified Control" Architecture

Giulio Capolino, Carolina Ortega Perez,
Cynthia Liu, Valerie Richmond

May 2020

1 Introduction

In the past decade, the automobile and technology industries have been reshaped by the push for autonomous vehicles (AVs). The revolution is still young, but has already revealed novel security challenges. Here we explore one of those challenges, as it relates to a proposed AV safety architecture.

1.1 Certified Control[1]

A typical AV architecture is shown in Figure 1. The sensors send their data to a controller, which does the complex work of perception and control, often using machine learning. The controller then sends its chosen action to the actuators, which physically execute the action. In this setup, each of the three components is in the "trusted base." This means that, were any single component to be infiltrated, the entire system would be compromised. This is concerning, especially due to the complexity of the controller. The controller may be internet-connected or otherwise particularly susceptible to malicious infiltration, so it is preferable to exclude it from the trusted base.

[1]The development of the certified control architecture, described here, is in no part ours. We attribute [2] for this concepts in this section.

Figure 1: Typical AV architecture.

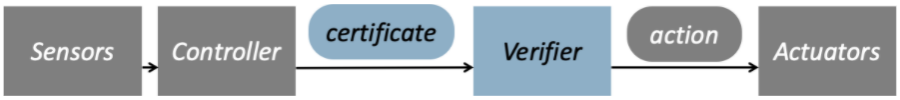

Figure 2: Certified control architecture. Blue components are the additions of certified control; gray components are typical for all AV architectures.

Professor Daniel Jackson's Software Design Group (CSAIL SDG), with support from Toyota Research Institute, has developed a novel AV software architecture called "certified control," which removes the controller from the trusted base. In the architecture, an independent component, which we'll call the "verifier,"[2] oversees safety and intervenes when danger is detected. The verifier receives a subset of sensor data, called a certificate, from the car's controller. The certificate serves as the controller's proof to the verifier that the action it chose is safe. Figure 2 shows this architecture.

As an example, consider a controller that decides to drive the vehicle ahead at the same speed. Say that it makes this decision based on data from LiDAR sensors. LiDAR sensors produce point clouds using pulsed lasers, and are one of the most common sensors used for perception and control in AVs today, so we focus on them for the scope of this project. The controller may then produce a certificate which contains LiDAR points that lie on the flat road ahead. The certificate can be thought of as the "core" subset of the sensor data—that subset of the data which truly makes a difference to the safety of the action. Those LiDAR points argue that there is no obstacle along the path ahead; each LiDAR reading provides direct physical evidence of an uninterrupted line from

[2]Termed by Jackson the "monitor" or "interlock."

the LiDAR unit to the point of reflection. Together, a collection of such LiDAR points, covering the cross section of the path ahead with appropriate density, indicates absence of an obstacle in the vehicle's lane. In this way, the certificate can convince the verifier of the safety of its chosen action.

Because generating such a certificate is usually much harder than checking it (a classic computer science gap), the verifier remains relatively simple and small. In this way, the architecture achieves a small trusted base, which includes only the sensors, the verifier, and the actuators, while excluding the vulnerable controller.

A critical aspect of the architecture is that the controller is not able to spoof sensor data to the verifier. That is, the data given to the verifier in the certificate must indeed be from the sensors. Otherwise an adversarial controller could choose a dangerous action, but construct a fake certificate to convince the verifier of its safety. We explore this authenticity requirement.

1.2 Contributions

In this paper, we explore the authenticity requirement of LiDAR points passed in a certified control certificate. We first contextualize the problem with a threat model, then describe the implementation of four signature schemes (some previously-existing and some self-created) in a simulated certified control architecture, and finally evaluate the schemes with an emphasis on efficiency. Code for the simulation and for each of the schemes can be found at `https://github.com/capoling/857-lidar-security`.

2 Threat Model

The threat model against which these LiDAR signature schemes are meant to protect is one in which LiDAR points emanate from a trusted source (the sensor), travel through an untrusted channel (the controller), and arrive at the verifier. As mentioned, it is assumed that the controller may be connected to

the internet, or otherwise be susceptible to malicious interference. It is assumed that the controller can perform efficient polynomial time algorithms[3].

It is assumed that secure bridge networks [1] are the mode of communication between components within the system, and that these bridge networks only exist between components expected and allowed to communicate, such that outsiders cannot send messages to components without infiltrating a particular component. That is, the verifier is not itself internet-connected, or otherwise able to receive any messages except for those from the controller. In this way, we mitigate the risk of denial of service attacks in which a non-adversarial controller's communication with the verifier can be blocked by a barrage of outsider messages.

It is assumed that the sensor cannot receive any incoming messages, including from the verifier. This eliminates some risk to the sensor, but also introduces some complexity, described in Section 3.1.1.

It is assumed that the verifier and the sensor unit have synchronized internal clocks (or even a shared clock). In each of the signature schemes, a supposed signature includes a plaintext timestamp. The verifier should, in addition to the cryptographic checks required for the scheme, confirm that the timestamp is sufficiently close to its own internal clock time. Times which are too far from the current time should be considered stale and rejected. This prevents replay attacks (in which old properly-signed LiDAR points can be sent again at inappropriate times, to construct an out-of-date picture of the landscape). The required closeness of time should be determined based on the speed of transmission from the sensor to the verifier, but we imagine it can be relatively conservative, since successful replay attacks (ones which compromise the safety of the vehicle actions) require the ability to reuse relatively old data in order to fool the controller—one's environment while driving does not generally change very quickly. For greater protection against replay attacks, however, we rec-

[3]but does not, of course, have the ability to break computationally intractable problems like the discrete log problem, on which standard cryptographic protocols rely—that is, the threat model does not encompass a controller with quantum computing abilities!

ommend the use of Stratum-1 clocks (highly precise clocks often used in AV systems) which provide guarantees on the delay between synchronized systems.

Our threat model does not include physical attacks which may affect the sensor readings. For example, we do not protect against an adversary placing reflective surfaces on the road, against which LiDAR rays may bounce erratically.

The signature schemes are meant to achieve verifiable authenticity of the LiDAR points. We do not require confidentiality, or any other security property, of the points.

3 Implementations

3.1 Simple Hashing

The first signature scheme is a simple hashing scheme in which the sensor and monitor share a hardcoded secret, s. The signature of a LiDAR point M at a timestamp t is a triple of the time, the point itself, and a hash of the concatenation of the point, time, and secret key[4]:

$$Sig(M, t) = t, M, H(M \| t \| s)$$

To verify a supposed signature $(t, M, hash)$, the monitor itself computes the hash of the concatenation of the pieces with its secret key. If the hash matches that in the triple, it accepts the signed point and time; otherwise, it rejects.

For this implementation, we use the SHA256 hash function. Given the assumption that SHA256 is collision resistant, it is difficult to find two hash input $x = M \| t \| s$, $x' = M' \| t' \| s'$ which yield the same hash. This is especially difficult in practice, since an adversary would not be satisfied with just any hash collision—they likely want $t \approx t'$ (so that the falsified triple's timestamp is in the future and soon enough to be part of a well-timed deceptive certificate), and would want a particular value of M' (in order to use it to construct a meaningfully deceptive certificate).

[4] $\|$ here represents concatenation.

The secret key in this scheme need not be impractically long to make it computationally difficult for an adversary to find. Imagine a malicious controller. It may query the verifier with triples $(t, M, H(M\|t\|x))$, where x is the controller's guess at the secret key, for multiple values of x. If the secret key is 120 bits, as it is in our implementation, the controller at worst needs to query for all 2^{120} possible keys. With each key taking a few milliseconds, that's 2^{120} queries * 3ms $\approx 10^{35}$ queries * 10^{-13} centuries = 10^{22} centuries. And that's not to mention the fact that the malicious controller would only be able to make such queries while the vehicle is in operation.

3.1.1 Requirement for a Shared Secret

For the above scheme, the sensor and the verifier require a shared secret. Without two-way communication between the components, this shared secret must be achieved via hardcoding into the hardware (done at vehicle assembly time). There is precedent for such a scheme: it is analogous to the hardcoding of trusted root certificate authorities on computers at manufacturing time. If two-way communication between the components were available, however, a secure key exchange protocol, such as Diffie-Hellman, could be leveraged. This would be more fault tolerant, since it would enable easy regeneration of the secret key if a breach were detected.

3.2 RSA with Timestamp

Our second attempt at security involved signing each LiDAR coordinate using a slightly modified version of RSA. In this scheme, the sensor is expected to inherently contain a public-secret key pair either hardcoded in its implementation, or a key-pair generator, as per RSA. Furthermore, this implementation expects that both controller and the verifier have access to a data structure for storing 'marked' timestamps (as well as a synchronized or shared clock). The message that the controller signs using the general RSA procedure is comprised of the LiDAR point coordinates and the timestamp from the shared internal clock.

The internal clock gets marked with the timestamp via the data structure (not elaborated in this paper given that it is implementation-specific). Once the controller has signed the message, the verifier's decryptipn procedure mimics the encryption one by taking the message, which has been publicly shared along with the encrypted signature, as well as the marked timestamp from the internal clock that it has access to, and checks that the hash of "point‖timestamp" matches the hash outputted using the public key. Specifically, the scheme is comprised of:

1. **Generator.** Given the parameters above:
   (a) RSA key pair generator inherently embedded in sensor (or hard-coded key-pair) creating pairs in the following form:
   (b) Secret key: (n, d)
   (c) Public key: (n, e)
2. **Signing.**:
   (a) m = LiDAR coordinates (x,y,z) || timestamp. This marks the internal clock at that timestamp
   (b) H(m) = hashing m using SHA256
   (c) s = $(H(m)^d)$ mod n
   (d) Output: ((x,y,z), s)
3. **Verifying.** Given ((x,y,z), s):
   (a) m' = (x,y,z) — timestamp retrieved from internal clock.
   (b) H(m') = hashing m' using SHA256
   (c) $d_h = (s^e)$ mod n
   (d) Outputs: $d_h == H(m')$

This procedure prevents many active attacks because, in order for an attack to succeed, the attacker would need to have access to the internal clock embedded in the AV's hardware.

However, the RSA encryption and decryption operations are not the most performance-friendly, thus possibly causing drawbacks in coordinates processing times. This could be improved by using batches of signed coordinates rather than individual signed ones. This would put more pressure on the reliability of the scheme, though, as one spoofed point is much less dangerous than a whole batch of spoofed points. Furthermore, it puts responsibility on the security of the internal clock, as well as the impossibility for an attacker to replicate such timestamps.

3.3 Ed25519

For our third signature scheme, we used Ed25519, which is a deterministic public-key signature system. An Edwards-curve Digital Signature Algorithm (EdDSA) algorithm is a digital signature scheme using a variant of Schnorr signature scheme using twisted Edward curves (a type of elliptic curves). Ed25519 is a type of EdDSA that uses Curve25519, which gives the scheme its name. Similar to the previous schemes, here we also signed a concatenation of a LiDAR point and its timestamp. This allows the verifier to confirm the data was obtained within a small frame of time. The following is general overview of Ed25519 and how we used it to sign our messages.

1. **Parameters.**

   Let E be an elliptic curve and B a point defined on E. Furthermore, define nB as the n-th multiple of B in E. Finally, let l be a prime.[5]

2. **Generator.** Given the parameters above:

   (a) SK is a random 256-bit string.

   (b) Let SHA-512$(SK) = (h_0, h_1, \ldots, h_{511})$.

[5]These variables have more constraints and have some specific values in Ed25519. For more details on the construction of the signature scheme, refer to [3].

(c) Define integer

$$a = 2^{254} + \sum_{3 \leq i \leq 253} 2^i h_i \in \{2^{251}, 2^{251} + 8, \ldots, 2^{255} - 8\}.$$

(d) $PK = aB$.

3. **Signing.** Given $N = (M||t)$:

   (a) Let $r = \text{SHA-512}(h_{256}, h_{257}, \ldots, h_{511}||N)$.

   (b) Define $R = rB$.

   (c) Let $S = (r + \text{SHA-512}(R||SK||N)a) \mod l$.

   (d) Return signature $(R||S)$.

4. **Verifying.** Given $N = (M||t)$ and $Sig(N) = (R||S)$:

   (a) Verify that the timestamp t is within a small range of time difference.

   (b) Verify that $8SB = 8R + 8\text{SHA-512}(R||PK||N)PK$ (in E).

   (c) Accept if (a) and (b) are true. Else, reject.

To implement this scheme, we used two python libraries. The first one was python-ed25519 (`github.com/warner/python-ed25519`). The second library was Nacl, from libsodium (`github.com/jedisct1/libsodium`).

Besides performance, some other advantages of Ed25519 are the fact that it uses SHA-512 which is assumed to be a collision resistant hash function that is generally faster than others in its family. Another factor that enforces security is the use of elliptic curve cryptography, which needs smaller keys and signatures than RSA to achieve a similar level of security. Moreover, as it will described in Section 5, Ed25519 achieves good performance results, which is critical in real time systems. Finally, similar to RSA, it is safe against active attacks such as replay or spoofing attacks under the assumption that the internal clock is reliable, efficient, and safe.

3.4 DSS

In addition to these schemes, we also used DSS (Digital Signature Standard) and its associated DSA (Digital Signature Algorithm) [9]. Like all aforementioned schemes, we sign a timestamp concatenated with the lidar point data. Similar to what we did for simple hashing, verification focuses on the validity of the signature, and delegates matching internal timestamps to another part of the system. The scheme works as follows, and a^{-1} represents the multiplicative inverse of a.

1. **Domain Parameters**
   (a) A prime p. Government (NIST) recommends that the bit length of p to be 1024, 2048, or 3072.
   (b) A prime q, a factor of $p - 1$. NIST recommendations sets the bit length of q to be 160, 224, 256.
   (c) Generator g of the group $GF(p)$
   (d) All of these are shared among devices in the system, and in practice are usually hardcoded.
2. **Generation.** Given the domain parameters:
   (a) (PK, SK) = (g^x, x) for some $0 < x < q$ randomly generated
   (b) k a secret number unique to each message
3. **Signing.** Given lidar points M, timestamp t, and hash function h:
   (a) Put t on the AV's internal clock.
   (b) Let $r = (g^k \mod p) \mod q$
   (c) Let z =the leftmost min(o, N) bits of $h(t||M)$, where o is the natural length of the hash output and N is the bit length of q.
   (d) $s = (k^{-1}(z + xr)) \mod q$
   (e) return signature (s, r)

4. **Verification.** Receive message M_v, signature (s_v, r_v):

   (a) Check that $0 < r_v < q$ and $0 < s_v < q$. If either is false, the signature is invalid.

   (b) Let $w = s_v^{-1} \mod q$

   (c) Retrieve timestamp t from the internal clock. Let z_v =the leftmost min(o, N) bits of $h(t||M_v)$

   (d) Let $u_1 = z_v w \mod q$ and $u_2 = r_v w \mod q$. Compute $\nu = (g^{u_1} g^{x u_2} \mod p) \mod q$

   (e) The message is valid if $\nu = r_v$.

   The security of this scheme depends on the discrete log assumption.

4 Experimental Setup

Without SDG's physical robot car setup available, we created Python3 simulations for each of the relevant components of the certified control architecture.

First, we created two simulators for the LiDAR sensor. Both simulators are built to mimic the behavior of SDG's sensor, the Velodyne Puck VLP16. The first simulator generates pseudo-random points, within some reasonable ranges, and yields groups of them at specified time intervals. The second simulator draws from real data recorded using SDG's robot car and LiDAR sensor. The simulator "replays" the same sensor outputs that the real setup produced. The data was taken over a few seconds when the car was in motion.

We also created a very simple simulator for the controller. In a real AV system, the controller would house the complex machine learning perception algorithms used to decide which of the LiDAR points from the sensor should be included in a certificate, such that the certificate contains convincing evidence of the safety of the controller's chosen action. For our purposes, a controller which randomly chose a fixed-size subset of the LiDAR points sufficed.

We created a simulation setup which took points from the simulated sensor, and passed them to the controller to get the subset of points contained in the

certificate. After the certificate points were chosen (randomly) by the controller, they were signed by the chosen scheme. Then, the signed points were passed to the verifier for checking. This is visualized in the architectural diagram in Figure 2. We could tune the speed at which this process happened.

To ensure that each scheme only accepted properly signed points, we also passed jibberish signatures (or signatures using a different, false key) to the verifier, expecting the verifier to reject points. In all cases, the final signature scheme implementations appropriately accepted validly signed points and rejected invalidly signed points.

5 Performance Evaluation

After confirming the correctness of each of our implementations for signing, verifying valid signatures, and rejecting invalid signatures, we focused our evaluation on performance. We emphasize performance evaluation because the verifier receives batches of LiDAR every few milliseconds[6], and the AV needs to authenticate these points quickly in order to decide on safety-critical actions based on the changing environment.

An average certificate in the certified control architecture contains about 70 points. We therefore conservatively tested each signature scheme on the signing and verifying of 100 points (although the times would scale approximately linearly for other numbers of points). Figure 3 shows the resulting runtimes. Each LiDAR point was represented as a triple of three coordinates, each with about 15 significant figures. All times were measured on a 2016 Macbook Pro with a 2.6 GHz Quad-Core Intel Core i7 Processor and 16 GB RAM. Time includes initializing the verifier, signing each of the points, and verifying each of the points. Times do not include simulation time required to generate or load in the points.

As is somewhat expected, simple hashing far outperforms the other schemes, because the only significant computation it does is hash a value, which is very

[6]for SDG's Velodyne Puck sensor, every 50ms

Scheme	Time to sign 100 points (ms)	Time to sign and verify 100 points (ms)
Simple Hashing	0.648	1.360
Ed25519 (with Nacl)	5.590	18.107
Ed25519 (without Nacl)	52.171	250.001
DSS	296.887	805.532
RSA with timestamp	501.819	506.064

Figure 3: Comparison of signature scheme runtimes, in increasing order by the first column.

fast. The other more sophisticated methods took proportionally more time. It is interesting to note that, while all schemes in the figure take longer to sign *and* verify than just to sign, the discrepancy between those times varies widely. For RSA, for example, it barely increases; for other schemes, it increases by several factors. This is reflective of the varying degrees of computational setup required to verify, and is worth considering in the context of performance required for AVs.

Given the performance results, we recommend the use of either Ed25519 (with the Nacl package) or the simple hashing schemes to handle AV performance needs. Both of these schemes can comfortably handle authentication of the certificate fast enough.

6 Rotating Keys

In order to increase the security of our system, we considered implementing a key rotation scheme, in which we periodically change the keys used in the signature schemes. However, a problem we encountered was that our system does not allow two-way communication between the sensor and the verifier, which is why we originally decided to hardcode the secrets. This implies that if we implemented key rotation, the rotation and synchronization would need to

depend on the hardcoded secrets and the timer, since those are the values both, the sensor and the verifier, share.

It is highly improbable for the adversary to guess the secret key based on the signed messages he may observe. For this reason, we consider the greatest threat against the secret keys to be the adversary gaining access to the hardcoded information. However, if this happened, the adversary would also gain access to any other secrets we use in the key rotations. If the key rotation scheme does not have any secret, it becomes useless, since it is completely predictable by the adversary. For this reason, we decided not to include key rotations in our scheme.

7 Current State and Other Threats

Many resources have already been put into the development of AVs. Most security-focused research has been targeted at showing vulnerabilities of AV components. It has been shown that some LiDAR sensor systems, as well as some camera-based perception systems and machine learning processes are susceptible to attacks. Bas (2015) has exposed the weakness of AV cameras to blinding and auto controls confusion attacks, proving the feasibility of jamming, relay, and spoofing on the overall AV workflow [4]. Cao et al (2019) as well as Malik (2019) exposed vulnerabilities in the LiDAR sensing process, and created a successful model for allowing spoofed LiDAR points to be read and processed as normal points by the autonomous vehicle, highlighting the dangerous vulnerability that we try to address in this paper [5] [6]. Finally, many papers have underlined the vulnerability of AVs to adversarial attacks on their machine learning processes, as seen in Sharma et al (2019) [7]. Often very complex and creative, these attacks may fool the machine into very dangerous, sometimes unknown situations. Given these dangers, some literature has also covered potential solutions, especially specific to LiDAR's coordinates spoofing. Tayeb et al [8] implemented a spoofing-proof method utilizing a DUO authentication procedure involving public-key infrastructure and clock timestamps to

improve the security of GPS signals in AVs. This same workflow can easily be replicated with LiDAR points and is rather similar to what we discussed in our RSA with Timestamps section. Mingxi et al (2012) [10] propose a fast signature scheme that is secure to active attacks, and that can be replicated in the design processes of AVs, given their network reliability, and the homomorphic public cryptography focus. Other solutions include ML adversarial resiliency and higher-complexity camera architectures focused around the prevention of active attacks. In conclusion, though the resources currently allocated to AV's RD are enormous, the published literature underlines an alarming lack of proven security in such machines, calling for more studies and ways to ensure safety in public roads.

8 Conclusion

As explored above, our work far from solves the myriad security threats facing AV development. However, in this project we managed to address a specific security aspect of a particular AV safety architecture. First we introduced that architecture, certified control, and described a threat model in that context. We then detailed four signature schemes which enabled the architecture's verifier to perform the necessary authenticity check of LiDAR points. We simulated the certified control architecture in order to determine that at least two of the schemes were performant enough to be deployed in a real certified control AV system. Those schemes would prevent replay and spoofing attacks from fooling the AV's verifier.

References

[1] Bridge Networks. https://docs.docker.com/network/bridge/ May, 2020.

[2] Jackson, Daniel; Jonathan DeCastro; Soonho Kong; Dimitrios Koutentakis; Angela Leong Feng Ping; Armando Solar-Lezama; Mike Wang; Xin Zhang. "Certified Control for Self-Driving Cars." Workshop on The Design and Analysis of Robust Systems. July, 2019. http://jadecastro.github.io/DARS_2019.pdf.

[3] Daniel J. Bernstein; Niels Duif; Tanja Lange; Peter Schwabe; Bo-Yin Yang. "High-speed high-security signatures". August, 2012. https://ed25519.cr.yp.to/ed25519-20110926.pdf.

[4] Stottelaar, and Bas G.B. "Practical Cyber-Attacks on Autonomous Vehicles." Practical cyber-attacks on autonomous vehicles, January 1, 1970. https://essay.utwente.nl/66766/.

[5] Cao, et al. "Adversarial Sensor Attack on LiDAR-Based Perception in Autonomous Driving." ArXiv.org, 20 Aug. 2019, arxiv.org/abs/1907.06826.

[6] Changalvala, Raghu Malik, Hafiz. (2019). LiDAR Data Integrity Verification for Autonomous Vehicle. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2943207.

[7] P. Sharma, D. Austin and H. Liu, "Attacks on Machine Learning: Adversarial Examples in Connected and Autonomous Vehicles," 2019 IEEE International Symposium on Technologies for Homeland Security (HST), Woburn, MA, USA, 2019, pp. 1-7, doi: 10.1109/HST47167.2019.9032989.

[8] S. Tayeb et al., "Securing the positioning signals of autonomous vehicles," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, 2017, pp. 4522-4528, doi: 10.1109/BigData.2017.8258493.

[9] U.S. Department of Commerce, National Institute of Standards and Technology. (2013). "Digital Signature Standard (DSS)." Retrieved from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[10] Yang, Mingxi Yan, Wenjie Fang, Huajing. (2012). Fast Signature Scheme for Network Coding. Journal of Algorithms Computational Technology. 6. 10.1260/1748-3018.6.1.17.