

# Zero-Knowledge Proof-of-Location Protocols for Vehicle Subsidies and Taxation Compliance<sup>\*</sup>

Dan Bogdanov<sup>1</sup>, Eduardo Brito<sup>1,2</sup>, Annika Jaakson<sup>1,2</sup>, Peeter Laud<sup>1</sup>, and Raul-Martin Rebane<sup>1</sup>

<sup>1</sup> Cybernetica AS, Tallinn, Estonia

{dan.bogdanov, eduardo.brito, annika.jaakson, peeter.laud, raul-martin.rebane}@cyber.ee

<sup>2</sup> University of Tartu, Tartu, Estonia

**Abstract.** This paper introduces a new set of privacy-preserving mechanisms for verifying compliance with location-based policies for vehicle taxation, or for (electric) vehicle (EV) subsidies, using Zero-Knowledge Proofs (ZKPs). We present the design and evaluation of a Zero-Knowledge Proof-of-Location (ZK-PoL) system that ensures a vehicle’s adherence to territorial driving requirements without disclosing specific location data, hence maintaining user privacy. Our findings suggest a promising approach to apply ZK-PoL protocols in large-scale governmental subsidy or taxation programs.

**Keywords:** Zero-Knowledge Proofs · Location privacy · Environmental policy · Tax enforcement.

## 1 Introduction

In economic policy, taxes and subsidies are key tools to influence markets and the behaviour of economic agents. Taxes typically make activities more expensive, while subsidies reduce costs. In environmental policy, they are used to discourage activities that harm the environment and encourage those that mitigate pollution. For example, congestion and highway taxes aim to reduce vehicle traffic or fund infrastructure maintenance, while subsidies for electric vehicles (EVs) seek to lower costs and increase adoption, reducing emissions from internal combustion engines.

Implementing such schemes often requires collecting vehicle location data, which can inadvertently compromise individual privacy. Location data can easily be linked to specific drivers, raising concerns about disproportionate surveillance. Recognizing that privacy and policy enforcement need not be a zero-sum game, this paper explores novel privacy-preserving approaches to data-driven

<sup>\*</sup> This research has been funded by the Defense Advanced Research Projects Agency (DARPA) under contract HR0011-20-C-0083. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

*This is the extended version of the paper to appear in the Proceedings of the 5th International Workshop on Security and Privacy in Intelligent Infrastructures (SP2I 2025), held in conjunction with the 20th International Conference on Availability, Reliability and Security (ARES 2025).*

environmental policy. We are the first to (1) formalize vehicle subsidy and taxation tasks in ways that align naturally with privacy-enhancing technologies like zero-knowledge proofs (ZKPs); (2) demonstrate how existing ZKP techniques can efficiently address these tasks; and (3) analyze practical barriers to deployment, offering insights into real-world applicability of Zero-Knowledge Proof of Location (ZK-PoL) protocols.

## 2 Background

### 2.1 Location-Based Vehicle Taxes and Subsidies

**EV subsidies.** Governments started introducing tax rebates and grants to boost electric vehicle (EV) adoption [46]. Estonia’s 2019 program offered up to 5000€ (pre-tax) [34], with eligibility requiring: 1. driving at least 80,000 km within four years, 2. completing 80% of that distance in Estonia, and 3. using renewable energy for the first 80,000 km. The Environmental Investment Centre executed the program.<sup>3</sup> Compliance was monitored through periodic reporting or GPS tracking via third-party providers.

**Road use and emission charges.** Vehicles can be taxed via fuel sales or odometer readings [16], but these methods lack road-specific granularity. Toll booths, RFID transponders, and camera systems offer precision but require expensive infrastructure. GPS tracking enables flexible, lower-cost taxation but raises privacy concerns on data storage and disclosure. Other mechanisms enforce emission standards, such as London’s Ultra Low Emission Zone (ULEZ) [19].

**Balancing verifiability and privacy.** Location-based enforcement compromises transparency, user control, and unlinkability, effectively imposing a “privacy price.” Sustainable mobility should not require sacrificing fundamental privacy rights. Cryptographic tolling schemes [26] and differential privacy (DP) [27] show promise, but even obfuscated public data can leak patterns [2]. Billing aggregation and external storage also expose users to data sovereignty risks [26]. This motivates secure computation techniques that generate verifiable proofs without disclosing raw location data, even in obfuscated form.

### 2.2 Proof-of-Location and Zero-Knowledge Proofs

Localisation techniques like GPS enable accurate position determination [32] but do not guarantee trust or verifiability. Proof-of-Location (PoL) extends localisation by authenticating location claims, with applications in smart mobility, content delivery, financial regulation, and smart cities [10].

PoL combines witnessing and trust [31]: a location proof is a verifiable certificate attesting a Prover’s presence at a point in space and time, supported by trusted Witness devices. Solutions combine localisation (e.g., GPS), wireless communication (e.g., Bluetooth), and cryptographic primitives (e.g., public-key cryptography). Foundational system models [10] address basic PoL needs. Privacy-enhanced PoL can be achieved by layering Zero-Knowledge Proofs (ZKPs).

<sup>3</sup> In 2023, the mileage requirements were relaxed to “use the car mostly in Estonia.”

ZKPs allow a Prover to convince a Verifier of a statement’s validity without revealing why. Statements are typically modelled as arithmetic circuits or constraint systems defining a binary relation  $R$ , with a public instance  $x$  and private witness  $w$ . A classic example is Schnorr’s protocol [38], where knowledge of a private key is proven without revealing it. ZKPs ensure *completeness* (honest proofs succeed), *soundness* (cheating Provers fail), and *zero-knowledge* (protocol transcripts reveal nothing beyond validity) [42].

For PoL applications, we adopt a centralized architecture [10], where a trusted Witness device generates signed location claims used in ZK proofs. These claims are transformed into verifiable proof statements without exposing raw trajectories [18], enabling Zero-Knowledge Proof-of-Location (ZK-PoL) [49]. Our constructions specifically employ new geometric techniques to structure proofs, enabling privacy-preserving attestations of extended driving behaviour.

### 3 Use cases

#### 3.1 Location-based vehicle purchase subsidies

Our first use case for ZK-PoL protocols is proving adherence to location-based conditions for vehicle purchase subsidies. The vehicle owner acts as the Prover, and the subsidy authority as the Verifier. The goal is to prove that the vehicle has driven at least  $x$  km over  $T$  years, with at least  $p\%$  of that distance within specified geographical bounds. For example, from Section 2.1,  $x = 80\,000$ ,  $T = 4$ , and  $p = 80$  within Estonia.

The vehicle is equipped with a trusted GPS device, assumed to be non-removable and tamper-evident. The device logs coordinates whenever the vehicle is active, and signs each trail with a private key, with the corresponding public key known to the Verifier or registered in a trusted PKI. Only the Prover has access to the raw data, transmitted via a local channel such as Bluetooth. Coordinates are projected into planar x-y values to reduce computational costs in distance calculations, using accurate projections like EPSG:3301 [33] for Estonian territory.

After  $T$  years (or periodically), the Prover generates a ZKP that the driven distance and coverage criteria are satisfied, without revealing the underlying trip coordinates. The proof verifies that the computations are based on genuine GPS data by checking the digital signatures against the known public key. The Prover can validate proof completeness before submission.

From a security perspective, preventing tampering is crucial. Without safeguards, the Prover could remove the device for trips outside the bounds or attach it to another vehicle to inflate mileage. Reliable calibration is also essential to ensure trustworthy location data. From a usability perspective, the device must be sufficiently accurate from the Prover’s standpoint. Underestimations of distance, or misclassification of trips outside the target area, could cause valid subsidy claims to fail. Dispute resolution processes for such cases are beyond the scope of this paper.

### 3.2 Location-based vehicle taxation

Our second use case is proving the amount of road tax that must be paid over a given period (e.g., month or quarter). The vehicle owner acts as the Prover, and the tax authority as the Verifier. Given a taxation period  $T$  and a network of toll roads, the Prover must demonstrate their driven distance  $d$  on toll roads during  $T$ . The tax owed depends on which distance range  $(x, y]$  the mileage falls into. The Prover’s goal is to prove that their toll-road driving distance satisfies  $d \leq y$  for a selected bracket, minimizing their tax liability.

The setup mirrors the subsidy case: a trusted, tamper-evident GPS device is attached to the vehicle, logging and signing location data. At the end of each taxation period, the Prover generates a ZKP proving their distance falls within a chosen bracket without revealing trip details, and submits it to the Verifier. Proof completeness ensures the Prover can select the lowest eligible tax bracket.

Unlike the subsidy case, proof submission is mandatory each period, with penalties for non-submission. Trust in device attachment and calibration is critical to prevent fraud, such as detaching the device or leaving it stationary. As before, device accuracy is essential, though dispute resolution mechanisms are outside the scope of this paper.

## 4 System Model

Building on these mobility use cases, we now instantiate the ZK-PoL model, noting its applicability to other domains such as asset or personnel tracking. The system model (Figure 1) centres on three components: the Witness device, the Prover, and the Verifier.

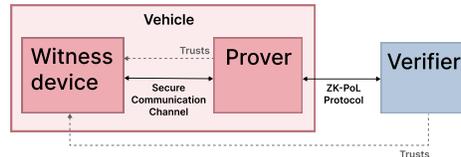


Fig. 1: ZK-PoL system model for mobility use cases.

The Witness device is a tamper-resistant or tamper-evident device coupled to the vehicle. Using its GPS module, it generates raw location claims, and signs them, without interpreting eligibility rules or computing taxes, allowing it to remain update-free even if policy changes. Trust is established between the Witness device and both the Prover and the Verifier, who rely on it to produce continuous, precise, and verifiable location data.

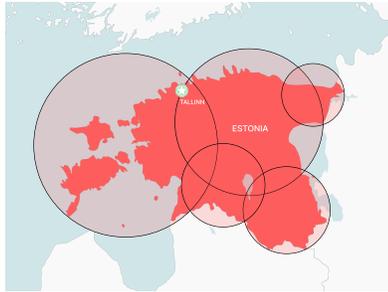
The Prover is any entity (owner, driver, or custodian) responsible for demonstrating location compliance. They establish a restricted communication channel with the Witness device to receive signed location data. Our model is not tied

to proximity constraints or specific communication ranges, unlike some related PoL work identified in Section 2.2. Importantly, proofs are bound to the vehicle’s identity via the Witness device, not the Prover’s personal identity. The Prover assembles the proof and manages subsequent interactions with the Verifier, including executing the ZKPs.

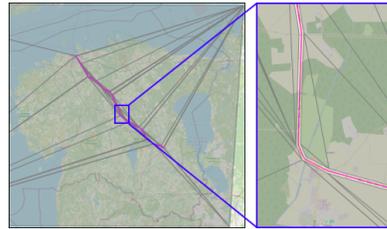
The Verifier trusts the Witness device’s public key  $pk$ , used to sign the location claims. The Prover’s proof asserts that a trajectory exists, signed and verifiable under  $pk$ , and satisfies the conditions for subsidy eligibility or applicable tax rates.

## 5 Zero-Knowledge Protocol Specification

Encoding ZK statements directly into low-level paradigms is difficult and error-prone, as these systems offer minimal abstraction and require bit-level reasoning. Higher-level approaches address this gap, including gadget APIs [24], DSLs [41,29,1], and general-purpose languages [15]. We use ZK-SecreC [4,5], a high-level imperative DSL with C++/Rust-like syntax, designed specifically for ZK proofs. It features an information flow type system [44,30] that separates Prover-private data and shared data, enforcing confidentiality and enabling efficient local computations. It also distinguishes compile-time and runtime values, supporting protocols that require relation preprocessing [23]. Local values injected into the proof must pass correctness checks to maintain soundness [39,45]. ZK-SecreC offers a standard library [7] of optimized primitives, including bit operations, fixed-point arithmetic [21,18], and efficient constructions for RAM and inequality proofs [8,45]. The compiler targets formats like SIEVE IR [11] and integrates with interactive ZKP systems [3,50] via a Rust trait [28], abstracting circuit construction through gates, constraints, and inputs.



(a) Approximation of Estonia by circles.



(b) Triangles covering Tallinn–Tartu road.

Fig. 2: Geographical approximations used for the EV subsidy (left) and highway tax (right) use cases.

### 5.1 EV subsidy use-case

The ZKP protocol goal in the EV subsidy case is for the Prover to demonstrate, with a list of coordinate points, that the total trail length exceeds a required distance, and a required percentage falls within specified geographical bounds.

We use a planar coordinate system to avoid expensive trigonometric operations. Geographical bounds are approximated by a union of circles (Figure 2a), which flexibly balances precision and computational cost. The proof data structure is shown in Figure 3a. Besides the technical parameters, discussed later, there are the values characterizing the *size* of the statement. Certain sizes have to be fixed; the running time of the protocol will depend (only) on these sizes. To simplify proofs, we assume:

- all coordinates and radii are positive integers,
- the trajectory has at most a known maximum number of points,
- the coordinate list is padded and hashed to hide the actual trail length.

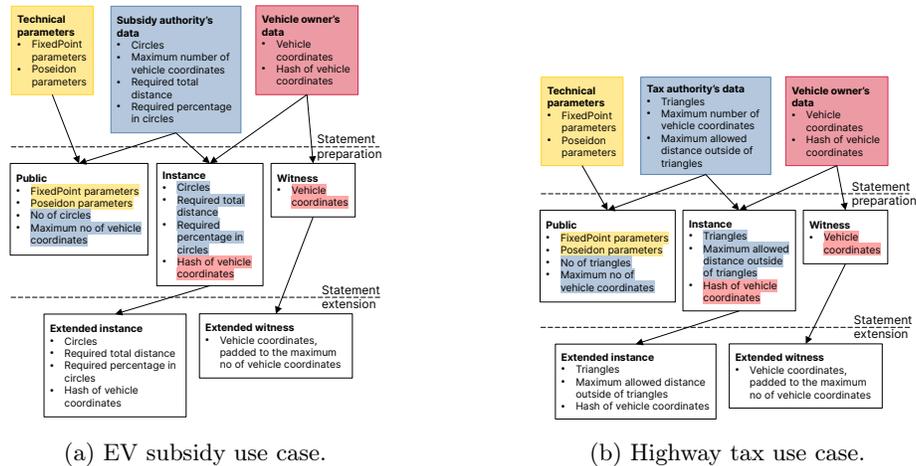


Fig. 3: Data models for the EV subsidy (left) and highway tax (right) use cases.

We presume that revealing the hash  $h_{\text{ex}}$  of the coordinate trail to the Verifier does not significantly compromise privacy. This hash, signed by the Witness device and transmitted via the Prover, binds the ZK proof to the Witness's public key, enabling the Verifier to independently verify authenticity in parallel to proof verification.

Algorithm 1 checks that the coordinate trail satisfies the subsidy conditions, using a subroutine from Algorithm 2 to determine circle inclusion. In our pseudocode, values managed by the ZK protocol are enclosed in double brackets, while locally managed values are not. Color coding indicates visibility: red for

**Input:** Parameters  $pp$  for Poseidon hash function  
**Input:** Centers  $(u_i, v_i)$  and radii  $r_i$  of circles ( $1 \leq i \leq n_{\text{circ}}$ )  
**Input:** Required distance  $d_{\text{req}}$  and percentage  $P_{\text{req}}$   
**Input:** Coordinates  $(x_i, y_i)$  on the trajectory ( $1 \leq i \leq n_{\text{traj}}$ )  
**Input:** Expected hash  $h_{\text{ex}}$  of the list of coordinates

```

1  $\llbracket tot \rrbracket \leftarrow 0, \llbracket cc \rrbracket \leftarrow 0$ 
2  $\llbracket x \rrbracket \leftarrow \text{wire}(x), \llbracket y \rrbracket \leftarrow \text{wire}(y)$ 
3  $\text{assert}(\text{hash}(pp, \llbracket x \rrbracket || \llbracket y \rrbracket) = \text{wire}(h_{\text{ex}}))$ 
4  $\llbracket u \rrbracket \leftarrow \text{wire}(u), \llbracket v \rrbracket \leftarrow \text{wire}(v)$ 
5 for  $i \leftarrow 1$  to  $n_{\text{circ}}$  do  $\llbracket s_i \rrbracket \leftarrow \text{wire}(r_i^2)$ ;
6  $\llbracket b\_pi \rrbracket \leftarrow \text{check\_inside}(\llbracket u \rrbracket, \llbracket v \rrbracket, \llbracket s \rrbracket, \llbracket x_1 \rrbracket, \llbracket y_1 \rrbracket)$ 
7 for  $i \leftarrow 2$  to  $n_{\text{traj}}$  do
8    $\llbracket b\_in \rrbracket \leftarrow \text{check\_inside}(\llbracket u \rrbracket, \llbracket v \rrbracket, \llbracket s \rrbracket, \llbracket x_i \rrbracket, \llbracket y_i \rrbracket)$ 
9    $\llbracket d_i \rrbracket \leftarrow \text{sqrt}((\llbracket x_i \rrbracket - \llbracket x_{i-1} \rrbracket)^2 + (\llbracket y_i \rrbracket - \llbracket y_{i-1} \rrbracket)^2)$ 
10   $\llbracket tot \rrbracket := \llbracket tot \rrbracket + \llbracket d_i \rrbracket$ 
11   $\llbracket cc \rrbracket := (\llbracket b\_pi \rrbracket \wedge \llbracket b\_in \rrbracket) ? (\llbracket cc \rrbracket + \llbracket d_i \rrbracket) : \llbracket cc \rrbracket$ 
12   $\llbracket b\_pi \rrbracket := \llbracket b\_in \rrbracket$ 
13  $\text{assert}(\text{wire}(d_{\text{req}}) \leq \llbracket tot \rrbracket)$ 
14  $\text{assert}(\llbracket tot \rrbracket * \text{wire}(P_{\text{req}}) \leq \llbracket cc \rrbracket * 100)$ 

```

**Algorithm 1:** Proof for the electric vehicle use-case.

**Input:** Centers  $(\llbracket u_i \rrbracket, \llbracket v_i \rrbracket)$  and squares of radii  $\llbracket s_i \rrbracket$  of circles ( $1 \leq i \leq n_{\text{circ}}$ )  
**Input:** Coordinates  $\llbracket x \rrbracket, \llbracket y \rrbracket$  of a point  
**Output:** Boolean  $\llbracket b \rrbracket$  indicating whether the point is inside at least one of the circles

```

1  $\llbracket b \rrbracket \leftarrow \text{false}$ 
2 for  $i \leftarrow 1$  to  $n_{\text{circ}}$  do
3    $\llbracket b \rrbracket := \llbracket b \rrbracket \vee ((\llbracket x \rrbracket - \llbracket u_i \rrbracket)^2 + (\llbracket y \rrbracket - \llbracket v_i \rrbracket)^2 \leq \llbracket s_i \rrbracket)$ 
4 return  $\llbracket b \rrbracket$ 

```

**Algorithm 2:** Checking that a point is inside a set of circles: `check_inside`.

Prover-only, green for shared data, and blue for public constants. Arithmetic and comparisons on ZK-managed values are realized securely by the underlying protocol, ensuring that cheating by the Prover is infeasible. Our conventions extend the notation commonly used for privacy-preserving computations, where one typically uses square brackets (either single or double) around a variable to denote that the value of this variable is handled by the cryptographic technology.

Variable types are not made explicit in the pseudocode but are deduced as follows: variables prefixed with  $b$  are booleans,  $h_{\text{ex}}$  is a hash output,  $pp$  contains hash parameters, and the rest are integers. All necessary primitives, including integer square roots (rounded toward zero), are provided by the ZK-SecreC standard library [7]. For the use of the square root, see also the discussion in Sec. 5.3.

We use the Poseidon hash function [22], optimized for field operations in ZKP systems, instead of general-purpose bit-based hashes like SHA-256, which would

incur costly bit extraction and manipulation overheads. Poseidon parameters are precomputed and included as public parameters.

In Algorithm 1, wire marks values as ZK protocol inputs, and `assert` enforces conditions during proof execution. If a wire argument is known to both parties, its value is trusted; if private to the Prover, it requires independent checks within the proof.

ZK protocols cannot branch on conditions that are also computed under ZK or depend on private values. Therefore, updating the distance in circles `cc` is non-trivial. We implement it using an *oblivious choice*, a ternary operation  $b ? x : y$  that returns  $x$  if  $b$  is true and  $y$  otherwise. For integers  $b, x, y$  with  $b \in \{0, 1\}$ , this can be computed as  $y + b(x - y)$ .

The proof statement structure (see Alg. 1) proceeds as follows. First, the Prover hashes the coordinate trail (padded with the last point) and asserts that it matches the expected instance hash. Then, it computes the total trail length (`tot`) and the length of segments where both endpoints lie within some circle (`cc`). Finally, it asserts:

$$\text{tot} \geq d_{req} \quad \text{and} \quad \text{cc} \geq \text{tot} \cdot \frac{P_{req}}{100}$$

If both assertions hold, the coordinate trail satisfies the subsidy rules. To compute `tot` and `cc` (see Alg. 1), we evaluate the Euclidean distance

$$d_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

for each pair of consecutive points, and check whether both points lie within some circle (not necessarily the same one). The inclusion check for a point  $(x_i, y_i)$  in a circle with centre  $(u_j, v_j)$  and radius  $r_j$  (see Alg. 2) is performed via:

$$(x_i - u_j)^2 + (y_i - v_j)^2 \leq r_j^2 \tag{1}$$

This comparison is done for each point and all circles  $j \in \{0, \dots, n_{traj}\}$ , accepting if any match. The values `tot` and `cc` are then:

$$\text{tot} = \sum_{i \in T} d_i \quad \text{cc} = \sum_{i \in S} d_i$$

where  $S \subset T$  includes indices  $i$  where both endpoints lie in some circle. Square roots are required to compute distances but are not needed for inclusion checks, which use squared distances and the inequality (1), thus saving computation under ZK.

## 5.2 Highway tax use-case

The goal of the ZKP statement in the highway tax use-case is to prove that no more than a specified distance of the Prover's coordinate trail lies along taxed roads (see Section 3.2). As in the subsidy use-case, an approximate representation is required. However, while subsidy proofs aim to show that points are

*inside* a designated area, here the Prover seeks to prove points are *outside* taxed roads. Thus, we represent the untaxed region instead by triangulating the area surrounding the highways. Being outside the taxed roads becomes equivalent to being inside one of these triangles.

Figure 2b illustrates this for the road between Tallinn and Tartu in Estonia: the road is marked in pink, and the surrounding area (including all of Estonia) is shown as gray triangles. A buffer margin around the road accounts for GPS noise and controls the precision-performance trade-off: tighter margins require more segments and thus more triangles, increasing proof cost.

The structure of the proof data is shown in Figure 3b. We adopt the same input constraints as in the EV use-case (Section 5.1), with one additional requirement: all points in the trail must either lie near the highway or within the public triangulation, i.e., within Estonian territory as depicted in Figure 2b.

<p><b>Input:</b> Parameters <math>pp</math> for Poseidon hash function  <b>Input:</b> Vertices <math>(X_{j,k}, Y_{j,k})</math> of triangles (<math>1 \leq j \leq n_{\text{tri}}, k \in \{1, 2, 3\}</math>)  <b>Input:</b> Maximum allowed on-highway distance <math>d_{\text{max}}</math>  <b>Input:</b> Coordinates <math>(x_i, y_i)</math> on the trajectory (<math>1 \leq i \leq n_{\text{traj}}</math>)  <b>Input:</b> Expected hash <math>h_{\text{ex}}</math> of the list of coordinates</p> <pre style="font-family: monospace; font-size: 0.9em;"> 1  <math>\llbracket tot \rrbracket \leftarrow 0, \llbracket hw \rrbracket \leftarrow 0</math> 2  <math>\llbracket x \rrbracket \leftarrow \text{wire}(x), \llbracket y \rrbracket \leftarrow \text{wire}(y)</math> 3  <math>\text{assert}(\text{hash}(pp, \llbracket x \rrbracket \parallel \llbracket y \rrbracket) = \text{wire}(h_{\text{ex}}))</math> 4  <math>\llbracket X \rrbracket \leftarrow \text{wire}(X), \llbracket Y \rrbracket \leftarrow \text{wire}(Y)</math> 5  <b>for</b> <math>i \leftarrow 1</math> <b>to</b> <math>n_{\text{traj}}</math> <b>do</b> 6    <math>\llbracket t_i \rrbracket \leftarrow \text{wire}(\text{find\_triangle}(x_i, y_i, X, Y))</math> 7    <math>\llbracket a_i \rrbracket \leftarrow \text{lookup}(\llbracket t_i \rrbracket, \llbracket X \rrbracket);</math> // <math> a_i  = 3</math> 8    <math>\llbracket b_i \rrbracket \leftarrow \text{lookup}(\llbracket t_i \rrbracket, \llbracket Y \rrbracket)</math> 9    <math>\llbracket c_i \rrbracket \leftarrow \text{check\_inside\_triangle}(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket x_i \rrbracket, \llbracket y_i \rrbracket)</math> 10   <b>if</b> <math>i &gt; 1</math> <b>then</b> 11     <math>\llbracket d_i \rrbracket \leftarrow \text{sqrt}((\llbracket x_i \rrbracket - \llbracket x_{i-1} \rrbracket)^2 + (\llbracket y_i \rrbracket - \llbracket y_{i-1} \rrbracket)^2)</math> 12     <math>\llbracket tot \rrbracket \leftarrow \llbracket tot \rrbracket + \llbracket d_i \rrbracket</math> 13     <math>\llbracket hw \rrbracket \leftarrow ((\llbracket c_{i-1} \rrbracket \wedge \llbracket c_i \rrbracket) ? (\llbracket hw \rrbracket + \llbracket d_i \rrbracket) : \llbracket hw \rrbracket)</math> 14  <math>\text{assert}(\llbracket tot \rrbracket - \llbracket hw \rrbracket \leq \text{wire}(d_{\text{max}}))</math> </pre>
---

**Algorithm 3:** Proof for the highway tax use case.

The pseudocode for the ZKP statement is given in Alg. 3, with a helper routine in Alg. 4. The conventions follow those in Alg. 1. The statement is more complex here, as checking triangle inclusion is more involved than for circles. A significant portion of computation — functions `find_triangle` and `get_bcoords` — is performed locally by the Prover. The results of these local computations are verified on the circuit, and the details of these computations do not affect the validity of the proof. The implementation details of `find_triangle` and `get_bcoords` are discussed in App. A.

As in Alg. 1, the ZK protocol begins by inputting the coordinate trail and asserting that its hash matches the one in the instance. It then computes the

**Input:** Vertices  $([a_1], [b_1]), ([a_2], [b_2]), ([a_3], [b_3])$  of a triangle  
**Input:** Coordinates  $[x], [y]$  of a point  
**Output:** Boolean indicating whether the point is inside the triangle

- 1  $[A] \leftarrow \Delta\text{area\_dbl}([a_1], [b_1], [a_2], [b_2], [a_3], [b_3])$
- 2  $(s, t) \leftarrow \text{get\_bcoords}(x, y, a_1, b_1, a_2, b_2, a_3, b_3)$
- 3  $[s] \leftarrow \text{wire}(s), [t] \leftarrow \text{wire}(t), [u] \leftarrow [A] - [s] - [t]$
- 4  $[x'] \leftarrow [u] \cdot [a_1] + [s] \cdot [a_2] + [t] \cdot [a_3]$
- 5  $[y'] \leftarrow [u] \cdot [b_1] + [s] \cdot [b_2] + [t] \cdot [b_3]$
- 6  $\text{assert}([x'] = [x] \cdot [A] \wedge [y'] = [y] \cdot [A])$
- 7 **return**  $0 \leq [s] \wedge 0 \leq [t] \wedge 0 \leq [u]$

**Algorithm 4:** Checking whether a point is inside a triangle:  
`check_inside_triangle`.

total trail length (`tot`) and the portion inside the triangle set (`hw`). Finally, it asserts that  $\text{tot} - \text{hw} \leq d_{\max}$ . If this check passes, the Prover is deemed to have driven at most  $d_{\max}$  on taxed roads.

Alg. 3 differs from Alg. 1 in the logic of checking whether a point is inside a region. While in Alg. 1 we simply performed the test for all circles and took the disjunction of results, in Alg. 3 Prover obviously picks a triangle and states that the point belongs to this particular triangle. The triangle-finding function `find_triangle` gives the index of that triangle (or an arbitrary index if the point does not belong to any triangle); this index  $t_i$  becomes another input to the ZK protocol. Next, we use the techniques of Oblivious RAM under ZK to locate the coordinates of the vertices of the chosen triangle; the function `lookup` gives us the  $x$ - and  $y$ -coordinates of this particular triangle. The function `lookup` can be implemented in various ways; our implementation (shown in Alg. 5) is based on expanding  $t_i$  to its characteristic vector and computing its scalar product with the vector of coordinates.

To check whether a point  $(x, y)$  lies inside a triangle  $[(a_1, b_1), (a_2, b_2), (a_3, b_3)]$  in ZK (Alg. 4), we use its *barycentric coordinates*<sup>4</sup>  $(s, t, u)$  with respect to the triangle's vertices. We use unnormalized coordinates, where  $s + t + u = A$  for all valid coordinate triples and for some positive constant  $A$ . These coordinates are computed locally by the Prover via `get_bcoords`, using only local inputs. The function `check_inside_triangle` then reconstructs the Cartesian point from  $(s, t, u)$  and asserts that it equals  $(x, y)$ . It returns true if and only if  $s, t, u \geq 0$ , meaning the point lies inside or on the boundary of the triangle.

We choose  $A$  as twice the area of the triangle  $[(a_1, b_1), (a_2, b_2), (a_3, b_3)]$ , which allows  $(s, t, u)$  to be computed without division. It is given by

$$\text{area\_dbl}(a_1, b_1, a_2, b_2, a_3, b_3) = \left| \det \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ 1 & 1 & 1 \end{pmatrix} \right| \quad (2)$$

<sup>4</sup> [https://en.wikipedia.org/wiki/Barycentric\\_coordinate\\_system](https://en.wikipedia.org/wiki/Barycentric_coordinate_system)

```

Input: Index  $\llbracket t \rrbracket$ 
Input: Single coordinates of vertices of triangles  $\llbracket X_{j,k} \rrbracket$  ( $1 \leq j \leq n_{\text{tri}}$ ,  $k \in \{1, 2, 3\}$ )
Output: Single coordinates  $\llbracket a_k \rrbracket$  of the selected triangle ( $k \in \{1, 2, 3\}$ )
1  $\llbracket s \rrbracket \leftarrow 0$ 
2 for  $i \leftarrow 1$  to  $n_{\text{tri}}$  do
3   if  $i = t$  then  $x_i \leftarrow 1$  else  $x_i \leftarrow 0$ ;
4    $\llbracket x_i \rrbracket \leftarrow \text{wire}(x_i)$ 
5   assert( $\llbracket x_i \rrbracket = 0 \vee \llbracket x_i \rrbracket = 1$ )
6    $\llbracket s \rrbracket := \llbracket s \rrbracket + \llbracket x_i \rrbracket$ 
7 assert( $\llbracket s \rrbracket = 1$ )
8  $\llbracket a_1 \rrbracket \leftarrow 0$ ,  $\llbracket a_2 \rrbracket \leftarrow 0$ ,  $\llbracket a_3 \rrbracket \leftarrow 0$ 
9 for  $i \leftarrow 1$  to  $n_{\text{tri}}$  do
10  for  $k \leftarrow 1$  to 3 do  $\llbracket a_k \rrbracket := \llbracket a_k \rrbracket + \llbracket x_i \rrbracket \cdot \llbracket X_{i,k} \rrbracket$ ;
11 return  $\llbracket a_1 \rrbracket, \llbracket a_2 \rrbracket, \llbracket a_3 \rrbracket$ 

```

**Algorithm 5:** Looking up the coordinates of vertices: lookup.

### 5.3 Possible optimizations

Several checks in our protocols can be optimized based on the Prover’s goals. For instance, Alg. 3 allows a relaxed behavior: when locating the triangle containing a point  $(x_i, y_i)$ , any incorrect index selection only disadvantages the Prover. If `check_inside_triangle` fails, the point is counted as outside, increasing the taxable distance. In both Alg. 1 and 3, the distance  $d_i$  between consecutive points is computed by ensuring

$$d_i^2 \leq (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 < (d_i + 1)^2,$$

Here,  $d_i$  is input by the Prover, and the ZK protocol checks the inequalities. Depending on the application, one of the checks may be omitted to reduce circuit size: in the highway tax use-case, where minimizing distance is the goal, only the upper bound matters; in the subsidy use-case, where demonstrating sufficient travel distance is important, only the lower bound is essential. If subsidy eligibility depended solely on distance inside circles, omitting the upper bound check would also be valid.

### 5.4 Security and Privacy Analysis

We summarize how our protocols fulfil key security and privacy requirements, while maintaining core cryptographic properties. A formal proof sketch is provided in App. B. Soundness, completeness, and zero-knowledge are guaranteed by:

- Correct cryptographic implementation of the underlying ZKP back-ends (e.g., `emp-zk` [47], Diet Mac’n’Cheese [20]),
- Correctness of the proof algorithms (Alg. 1-5),

- Enforcement of information flow and visibility policies by the ZK-SecreC type system [4, Thm. 3–5],
- Secure linkage of location data via the tamper-evident Witness device.

ZK-SecreC enables high-level construction of ZKPs while enforcing data visibility and input validation policies, critical to preserving privacy guarantees. The compiled circuits target interactive ZK backends, such as `emp-zk` and Diet Mac’n’Cheese, where proof generation proceeds through multiple rounds of communication rather than producing a single static proof artifact. Correctness at both the high-level and back-end levels is essential, as errors—e.g., in fixed-point arithmetic or unchecked values—could undermine soundness or zero-knowledge.

Our trust model assumes that the Witness device securely signs location data, and that raw GPS trails remain private to the Prover. The Verifier learns only whether compliance conditions are met, via privacy-preserving proofs. Unlike conventional systems (see Section 2) that expose full location logs or odometer readings, our approach preserves data and computational sovereignty while revealing no more than necessary. Compared to methods like differential privacy (DP) or secure multi-party computation (MPC), ZKPs enable individual compliance proofs without coordination [6] or statistical leakage [26].

## 6 Performance Evaluation

We prototyped both use cases to assess whether their performance with current ZKP technology is acceptable for real-world deployment. We used a Raspberry Pi 3B/4 with an embedded GPS module in place of the Witness device (in an actual deployment, this device should be tamper-resistant). The device contained a private signing key corresponding to a known public key. It captured coordinates every 30 seconds, consistent with commercial fleet tracking standards [43]. Test input sizes (200, 3600, and 43,800 points) correspond to typical single trips, monthly, and yearly driving durations [40].

In the EV subsidy use case, we approximated Estonia’s territory with 5 circles; in the highway-tax use case, the area along a 179 km highway (Tallinn–Tartu) was modelled using 248 triangles, achieving a maximal deviation of 50 meters from the real road. The number of shapes depends on the geographic area and tolerated error.

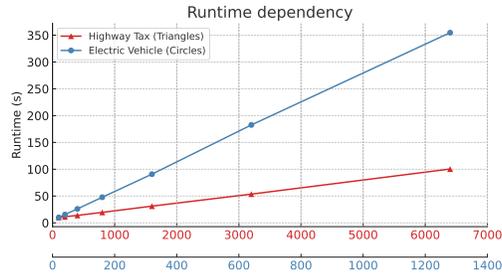
The Raspberry Pi sent signed location data over a secure Bluetooth connection to a Prover device. A native app handled Bluetooth configuration, loaded a WebAssembly-based executable via WebView, and executed the ZKP protocol with a remote Verifier server. Figure 4a shows the real-world prototype setup. Outside the browser, we tested `emp-zk` [47] and Diet Mac’n’Cheese (MnC) [20] back-ends, though comparisons are slightly biased due to better MnC integration. `Emp-zk` tests were run locally (0 ms delay). MnC tests were run under three network conditions: Fast (10 ms, 300 Mbit/s), Medium (80 ms, 100 Mbit/s), and Slow (200 ms, 10 Mbit/s). Verifiers used a SuperServer 6028R-TR (2×Intel Xeon E5-2640 v3, 128GB RAM), while Provers included a Lenovo ThinkPad T14s Gen 1 (Ryzen 7 PRO 4750U), a Google Pixel 5a, and a Raspberry Pi Compute Module

4. Details appear in Table 1. Due to differing protocol parameters, performance sensitivity to latency varies across devices.

We also evaluated how performance scales with the number of circles or triangles. Using single-trip input size, we varied the number of shapes and observed runtime trends (Figure 4b). For this test, the PC Prover ran both Prover and Verifier roles without network delay, isolating computational complexity. Results confirm a linear dependency on the number of shapes. For large shape counts, the highway-tax algorithm (Alg. 3) outperforms the EV subsidy algorithm (Alg. 1) in point-in-shape checking, due to use-case-specific optimizations. These trends are expected to hold for longer vehicle trajectories due to the linear complexity of the underlying algorithms (Section 5).



(a) Prototype setup: Android phone (Prover) and Raspberry Pi (Witness) connected via Bluetooth.



(b) Runtime vs. number of circles (EV) or triangles (Highway Tax) for area approximation.

Fig. 4: (a) Real-world prototype; (b) Local runtime performance.

## 7 Discussion

Recent advances in ZKP technology enable integration with identity systems [17]. Our prototype using ZK-SecreC tooling shows that identifiers, hashes, and ECDSA signatures can be incorporated with vehicle, driver, and GPS records. Fraud and anomaly detection could also be addressed with limited disclosure techniques as in MPC [6]. While ZKPs provide strong privacy and integrity guarantees, integration must include secure logging without exposing sensitive data. With successful deployments in voting and cryptocurrency, we anticipate commercial-grade ZK-PoL systems within 3–5 years, contingent on investment.

Hardware requirements include a tamper-evident GPS module securely coupled to the vehicle, capable of signing trip data and distinguishing between normal driving and transport scenarios. Existing commercial devices partially meet these needs, typically offering sealed enclosures, secure key storage, or tamper alerts. While we assume the device cannot be undetectably removed or spoofed,

	Pi 4 (F)	Pi 4 (M)	Pi 4 (S)	Phone (F)	Phone (M)	Phone (S)
EV trip	12.01 (0.15)	17.49 (0.09)	29.67 (0.20)	13.60 (0.15)	21.02 (0.44)	32.84 (0.07)
EV month	47.28 (0.33)	54.12 (0.20)	69.61 (0.44)	127.88 (0.88)	151.92 (2.49)	189.2 (1.90)
EV year	1229.03 (17.07)	1275.28 (15.18)	1309.39 (21.75)	1724 (16.43)	2078 (26.74)	2371 (27.93)
HW trip	25.57 (0.15)	31.52 (0.21)	45.72 (0.58)	36.44 (0.25)	43.98 (0.35)	54.33 (0.49)
HW month	323.55 (4.96)	336.82 (2.44)	387.70 (4.92)	369.39 (1.44)	388.72 (5.88)	403.26 (6.08)
HW year	-	-	-	-	-	-

	PC (F)	PC (M)	PC (S)	PC (emp)
EV trip	6.48 (1.49)	15.51 (1.49)	40.13 (0.80)	17.47
EV month	21.04 (2.17)	29.36 (4.59)	55.47 (1.64)	172.60
EV year	143.88 (4.40)	184.22 (8.76)	245.90 (5.00)	2131.36
HW trip	11.88 (2.07)	18.55 (2.18)	48.59 (1.00)	24.47
HW month	81.79 (2.11)	127.69 (8.28)	193.87 (12.34)	293.70
HW year	1052.06 (13.79)	1576.40 (97.22)	2226.76 (88.04)	3565.91

Table 1: Average performance (seconds) and standard deviation over 10 runs. For MnC, the prover was a Raspberry Pi 4, Phone, or PC under Fast (F), Medium (M), or Slow (S) networks. For emp-zk, the prover was a PC with a Fast network.

this trust anchor remains a key deployment consideration, and additional safeguards—such as secure mounting or periodic verification—may be required in practice. Offloading ZKP computation to the module could improve usability but raises complexity. Our benchmarks (Section 6) indicate that annual proofs (43,800 points) require roughly 36 minutes (EV subsidy) or 59 minutes (highway taxation), though proofs can be batched. These runtimes are tolerable for infrequent proofs and can be amortized over idle or charging periods. Further reductions are expected from future work on non-interactive back-ends, rollups, and hardware acceleration.

ZK-PoL protocols meet emerging public sector needs for transparent subsidy distribution [25] and help private entities prepare for regulatory shifts. A viable ecosystem will require coordination among service providers, vendors, and installers, plus interoperability standards to prevent fragmentation. By preserving user privacy and control [37], ZK-PoL offers a critical advantage as regulations evolve. Although proof generation demands user participation and computation time, these barriers could be mitigated by proof offloading, user-friendly apps, and back-end optimizations. Practical deployments will also need to address policy robustness, trust anchor management, and graceful failure handling.

## 8 Conclusion and Future Work

This paper presented the prototyping of privacy-preserving mechanisms for vehicle subsidy and taxation compliance, making three main contributions. First, we formalized subsidy and taxation proof tasks in a way that remains close to societal intuitions while being compatible with ZKPs. Second, we demonstrated that existing ZK techniques can efficiently address these tasks. Finally, we analysed the steps and barriers toward practical deployment.

We introduced and evaluated ZK-PoL protocols that verify compliance claims while preserving detailed location privacy. Beyond mobility, similar protocols could secure high-value asset tracking in supply chains [31], monitor smart city infrastructure [9], verify attendance at events [36], and authenticate autonomous systems' locations in industrial settings [48]. Applications in insurance are also promising, enabling verifiable incentives for low-risk driving and supporting usage-based models [14].

Large-scale adoption will require overcoming infrastructure, scalability, and interoperability challenges. Addressing these issues is critical for integrating ZK-PoL systems into the evolving landscape of smart mobility and privacy-aware digital services.

## References

1. Cosku Acay, Rolph Recto, Joshua Gancher, Andrew C. Myers, and Elaine Shi. Viaduct: an extensible, optimizing compiler for secure distributed programs. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 740–755. ACM, 2021.
2. Amirhossein Adavoudi Jolfaei, Andy Rupp, Stefan Schiffner, and Thomas Engel. Why privacy-preserving protocols are sometimes not enough: A case study of the brisbane toll collection infrastructure. *Proceedings on Privacy Enhancing Technologies*, 2024(1), 2024.
3. Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 92–122. Springer, 2021.
4. Dan Bogdanov, Joosep Jääger, Peeter Laud, Härmel Nestra, Martin Pettai, Jaak Randmets, Raul-Martin Rebane, Ville Sokk, Kert Tali, and Sandhra-Mirella Valdma. ZK-SecreC: a Domain-Specific Language for Zero-Knowledge Proofs. In *37th IEEE Computer Security Foundations Symposium, CSF 2024, Enschede, Netherlands, July 8-12, 2024*, pages 372–387. IEEE, 2024.
5. Dan Bogdanov, Joosep Jääger, Peeter Laud, Härmel Nestra, Martin Pettai, Jaak Randmets, Ville Sokk, Kert Tali, and Sandhra-Mirella Valdma. ZK-SecreC: a domain-specific language for zero knowledge proofs. *CoRR*, abs/2203.15448, 2022.
6. Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security*, pages 227–234, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
7. Dan Bogdanov, Joosep Jääger, Peeter Laud, Härmel Nestra, Martin Pettai, Jaak Randmets, Raul-Martin Rebane, and Ville Sokk. Zk-secrec compiler. <https://github.com/zk-secrec/compiler>, 2024.
8. Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology -*

- ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626. Springer, 2018.
9. Lorenz Bornholdt, Julian Reher, and Volker Skwarek. Proof-of-location: A method for securing sensor-data-communication in a byzantine fault tolerant way. In *Mobile Communication-Technologies and Applications; 24. ITG-Symposium*, pages 1–6. VDE, 2019.
  10. Eduardo Brito, Ammir Hadachi, Liina Kamm, and Ulrich Norbistrath. Decentralized Proof-of-Location systems for trust, scalability, and privacy in digital societies. *Scientific Reports*, 15(1):1–20, 2025.
  11. Paul Bunn, David Darais, Daniel Genkin, Steve Lu, Kimberlee Model, Tarik Riviere, Muthuramakrishnan Venkitasubramaniam, Xiao Wang, Steven Eker, Karim Eldefrawy, Stéphane Graham-Lengrand, Vitor Pereira, Hadas Zeilberger, Michael Adjedj, Daniel Benarroch Guenun, Eran Tromer, Aurélien Nicolas, Constance Beguier, and Mayank Varia. SIEVE Intermediate Representation. <https://github.com/sieve-zk/ir>, 2022.
  12. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.
  13. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.
  14. Tanguy Catlin, Xueqi Chang, Doug McElhaney, and Dimitris Paterakis. Connected revolution: The future of us auto insurance, 2023. McKinsey & Company, <https://www.mckinsey.com/industries/financial-services/our-insights/connected-revolution-the-future-of-us-auto-insurance>.
  15. Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 253–270. IEEE Computer Society, 2015.
  16. Liisa Ecola, Paul Sorensen, Martin Wachs, Max Donath, Lee Munnich, and Betty Serian. Moving Toward Vehicle Miles of Travel Fees to Replace Fuel Taxes Assessing the Path Forward, 2011. Rand Corporation, [https://www.rand.org/pubs/research\\_briefs/RB9576/index1.html](https://www.rand.org/pubs/research_briefs/RB9576/index1.html).
  17. Jens Ernstberger, Stefanos Chaliasos, Liyi Zhou, Philipp Jovanovic, and Arthur Gervais. Do you need a zero knowledge proof? *Cryptology ePrint Archive*, 2024.
  18. Jens Ernstberger, Chengru Zhang, Luca Ciprian, Philipp Jovanovic, and Sebastian Steinhorst. Zero-knowledge location privacy via accurate floating-point snarks, 2024.
  19. Transport for London. Ultra Low Emission Zone., 2023.
  20. Galois, Inc. swanky: A suite of rust libraries for secure computation, 2024. <https://github.com/GaloisInc/swanky>.
  21. Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In Weizhi Meng, Christian Damsgaard Jensen, Cas

- Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 1880–1894. ACM, 2023.
22. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schafneggger. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535, 2021.
  23. Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.
  24. Iden3. The circom Language, 2021. <https://docs.circom.io/circom-language/signals/>.
  25. IEA. Global ev outlook 2021, paris, 2021.
  26. Amirhossein Adavoudi Jolfaei, Abdelwahab Boualouache, Andy Rupp, Stefan Schiffner, and Thomas Engel. A survey on privacy-preserving electronic toll collection schemes for intelligent transportation systems. *IEEE transactions on intelligent transportation systems*, 24(9):8945–8962, 2023.
  27. Jong Wook Kim, Kennedy Edemacu, Jong Seon Kim, Yon Dohn Chung, and Beakcheol Jang. A survey of differential privacy-based techniques and their applicability to location-based services. *Computers & Security*, 111:102464, 2021.
  28. Steve Klabnik and Carol Nichols. *The Rust Programming Language*, chapter 10.2 Traits: Defining Shared Behavior. No Starch Press, 2nd edition, 2022.
  29. Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. xJsnark: A Framework for Efficient Verifiable Computation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 944–961. IEEE Computer Society, 2018.
  30. Andrew C. Myers. Jflow: Practical mostly-static information flow control. In Andrew W. Appel and Alex Aiken, editors, *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, pages 228–241. ACM, 1999.
  31. Bulat Nasrulin, Muhammad Muzammal, and Qiang Qu. A robust spatio-temporal verification protocol for blockchain. In *Web Information Systems Engineering-WISE 2018: 19th International Conference, Dubai, United Arab Emirates, November 12-15, 2018, Proceedings, Part I 19*, pages 52–67. Springer International Publishing, 2018.
  32. Huthaifa Obeidat, Wafa Shuaieb, Omar Obeidat, and Raed Abd-Alhameed. A review of indoor localization techniques and wireless technologies. *Wireless Personal Communications*, 119:289–327, 2021.
  33. Republic of Estonia Land Board. L-est coordinate system, 2019.
  34. Estonian Ministry of the Environment. Conditions and procedure for electric vehicle purchase subsidies. Entered into force on December 16th, 2019. (in Estonian), 2019.
  35. Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001*, pages 184–200. IEEE Computer Society, 2001.
  36. Evangelos Pournaras. Proof of witness presence: Blockchain consensus for augmented democracy in smart cities. *Journal of Parallel and Distributed Computing*, 145:160–175, 2020.

37. General Data Protection Regulation. Art. 22 gdpr. automated individual decision-making, including profiling. *Intersoft Consulting*, <https://gdpr-info.eu/art-22-gdpr>, 2020.
38. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.
39. Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking Proof-Based Verified Computation a Few Steps Closer to Practicality. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 253–268. USENIX Association, 2012.
40. R. Steinbach and B.C. Tefft. American driving survey: 2022 (research brief). Technical report, AAA Foundation for Traffic Safety, Washington, D.C., 2023.
41. Gordon Stewart, Samuel Merten, and Logan Leland. Snårkl: Somewhat Practical, Pretty Much Declarative Verifiable Computing in Haskell. In Francesco Calimeri, Kevin W. Hamlen, and Nicola Leone, editors, *Practical Aspects of Declarative Languages - 20th International Symposium, PADL 2018, Los Angeles, CA, USA, January 8-9, 2018, Proceedings*, volume 10702 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2018.
42. Justin Thaler. Proofs, arguments, and zero-knowledge (draft manuscript). <https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html>, 2023.
43. Verizon. Verizon connect near-real time gps fleet tracking, 2020.
44. Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *J. Comput. Secur.*, 4(2/3):167–188, 1996.
45. Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.
46. Sandra Wappelhorst, Dale Hall, Mike Nicholas, and Nic Lutsey. Analyzing policies to grow the electric vehicle market in european cities. *International Council on Clean Transportation*, 2020.
47. Chenkai Weng and Xiao Wang. emp-zk — efficient and interactive zero-knowledge proofs, 2023. <https://github.com/emp-toolkit/emp-zk>.
48. Evan W Wu, Marius Jurt, Ben Holden, and Yichao Jin. Proof of location verification towards trustworthy collaborative multi-vendor robotic systems. In *2024 IEEE International Conference on Industrial Technology (ICIT)*, pages 1–8. IEEE, 2024.
49. Wei Wu, Erwu Liu, Xinglin Gong, and Rui Wang. Blockchain based zero-knowledge proof of location in iot. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2020.
50. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2986–3001. ACM, 2021.

## A Additional algorithms in the Highway Tax use-case

We collect here the specifications of local computations by the Prover. The first of them, given in Alg. 6 determines whether a point is inside a triangle. One possible way to check this is to compute the area of the given triangle, and also compute the areas of three triangles that are formed by the given point and two vertices of the given triangle. If the given point is inside the given triangle, then these three triangles exactly cover the given triangle. If the given point is outside, then they also cover the given triangle, but also cover some area outside the triangle.

```

Input: Coordinates of a point  $(x, y)$ 
Input: Vertices  $(X_{j,k}, Y_{j,k})$  of triangles  $(1 \leq j \leq n_{\text{tri}}, k \in \{1, 2, 3\})$ 
Output: Index  $t$  of the first triangle containing  $(x, y)$ , or 1 if none
1 for  $i \leftarrow 1$  to  $n_{\text{tri}}$  do
2    $a \leftarrow \Delta\text{area\_dbl}(X_{i,1}, Y_{i,1}, X_{i,2}, Y_{i,2}, X_{i,3}, Y_{i,3})$ 
3    $b \leftarrow \Delta\text{area\_dbl}(X_{i,1}, Y_{i,1}, X_{i,2}, Y_{i,2}, x, y)$ 
4    $c \leftarrow \Delta\text{area\_dbl}(X_{i,1}, Y_{i,1}, x, y, X_{i,3}, Y_{i,3})$ 
5    $d \leftarrow \Delta\text{area\_dbl}(x, y, X_{i,2}, Y_{i,2}, X_{i,3}, Y_{i,3})$ 
6   if  $a = b + c + d$  then
7      $t \leftarrow i$ 
8     return  $t$ 
9 return 1

```

**Algorithm 6:** Finding the triangle: `find_triangle`

The second local computation is that of `get_bcoords`, converting Cartesian coordinates to unnormalized barycentric coordinates (summing up to twice of the area of the triangle). The computation in Alg. 7 makes use of published formulas<sup>5</sup> for conversion. In order to make sure that we get positive coordinates for points inside the triangle, we need to know the sign of the determinant in (2). We thus define

$$\Delta\text{area\_dbl\_sgn}(a_1, b_1, a_2, b_2, a_3, b_3) = \det \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ 1 & 1 & 1 \end{pmatrix} .$$

## B Extended Security and Privacy Analysis

We can present the security proof of our system in the universal composability (UC) [12] framework (actually, our treatment is closer to the equivalent *reactive*

<sup>5</sup> [https://en.wikipedia.org/wiki/Barycentric\\_coordinate\\_system#Vertex\\_approach](https://en.wikipedia.org/wiki/Barycentric_coordinate_system#Vertex_approach)

**Input:** Coordinates of a point  $(x, y)$   
**Input:** Coordinates of the vertices of a triangle:  $(a_1, b_1), (a_2, b_2), (a_3, b_3)$   
**Output:** Unnormalized barycentric coordinates  $(s, t)$  of the given point with respect to the given triangle

```

1  $area \leftarrow \Delta area\_dbl\_sgn(a_1, b_1, a_2, b_2, a_3, b_3)$ 
2  $sgn \leftarrow \text{if } area \geq 0 \text{ then } 1 \text{ else } -1;$ 
3  $s \leftarrow sgn * (b_1 \cdot a_3 - a_1 \cdot b_3 + (b_3 - b_1) \cdot x + (a_1 - a_3) \cdot y)$ 
4  $t \leftarrow sgn * (a_1 \cdot b_2 - b_1 \cdot a_2 + (b_1 - b_2) \cdot x + (a_2 - a_1) \cdot y)$ 
5 return  $(s, t)$ 

```

**Algorithm 7:** Barycentric coordinates of a point in a triangle:  
get\_bcoords

*simulatability* framework [35]). We can abstract the system model presented in Sec. 4 as an ideal functionality, with clear interfaces for both the environment (a.k.a. “the rest of the system”) and for the adversary. We can then show that the real system, consisting of the Witness device, the Prover, and the Verifier, all running the cryptographic technologies discussed in Sec. 5, is a secure implementation of this ideal functionality.

## B.1 Interface for the environment

Our system conceptually consists of three components — the Witness device, the Prover, and the Verifier —, which react to the stimuli from the environment. Both the real system and the ideal functionality offer the same interface towards the environment. Logically, the interface is split into three parts, corresponding to the components. The environment can give the following “stimuli” (i.e. commands) to the real system or the ideal functionality, and receive the following replies:

- At the beginning of the execution, the Witness device may receive the command  $(\text{init}, sid)$  from the environment. Here  $sid$  is the “session identifier” that is used to tie together any other commands / messages that the system or functionality may exchange in response to this command.
- During the execution, the Witness device may receive commands of the form  $(\text{move}, sid, (x, y))$ , where  $(x, y)$  are the current coordinates into which the environment has presumably moved the Witness device.
- At some point during the execution, the Prover may receive  $(\text{prove}, sid, AD)$ , while the Verifier also receives  $(\text{verify}, sid, AD)$ . These commands indicate that the system is now supposed to check whether the coordinate trace satisfies some policy, a part of which is given by the *authority’s data*  $AD$ . The input  $AD$  (which must be the same for the Prover and the Verifier) corresponds to Verifier’s inputs to Alg. 1 and Alg. 3. Note the use of  $sid$  to indicate which request to prove and request to verify correspond to each other. In response to these commands, the environment receives  $(\text{ok}, sid)$  or  $(\text{not\_ok}, sid)$  from both the Prover and the Verifier.

## B.2 Ideal functionality

Our ideal functionality  $\mathcal{I}$  is parametrized with a relation  $R$  corresponding to the *policy* that the trajectory of the system has to satisfy. The arguments of  $R$  are the list of coordinates  $\mathbf{S}$  and the authority's data  $AD$ . Our ideal functionality is also parametrized with the hash function  $H$  used in Alg. 1 and Alg. 3.

The ideal functionality offers the described interface to the environment and another one to the adversary. It keeps the list of coordinates  $\mathbf{S}$  as its internal state. The ideal functionality works as follows:

- At the beginning of the execution, it may receive (`corrupt_prover`) or (`corrupt_verifier`) (but not both) from the adversary. If it receives them,  $\mathcal{I}$  records that either Prover or Verifier has been corrupted.
- On input (`init, sid`) from the environment over Witness device's interface,  $\mathcal{I}$  initializes the sequence of coordinates  $\mathbf{S}$  to the empty sequence.
- On input (`move, sid, (x, y)`) from the environment over Witness device's interface,  $\mathcal{I}$  appends the coordinates  $(x, y)$  to the sequence of coordinates  $\mathbf{S}$ .

These steps are straightforward. The handling of the proof request is more complicated, because the output depends not only on the policy  $R$  and the list of coordinates  $\mathbf{S}$ , but also on authority's data  $AD$ , whether the same  $AD$  was input to the Prover and the Verifier, and, in case the Prover or the Verifier is corrupted, also on the choices of the adversary. Thus, on input (`prove, sid, ADP`) from the environment to the Prover, and (`verify, sid, ADV`) to the Verifier,  $\mathcal{I}$  works as follows:

- Initialize  $res_P \leftarrow 1$  and  $res_V \leftarrow 1$
- Send  $H(\mathbf{S})$ ,  $AD_P$ , and  $AD_V$  to the adversary.
- If Prover is corrupted, then also send  $\mathbf{S}$  to the adversary.
- If  $R(\mathbf{S}, AD_P) = 0$ , then put  $res_P := 0$ .
- If  $AD_P \neq AD_V$ , or if  $R(\mathbf{S}, AD_V) = 0$ , then put  $res_V := 0$ .
- If Prover is corrupted, then send (`output, sid, resP`) to the adversary, and receive back (`update, sid, b`). Update  $res_P := b$ .
- If Verifier is corrupted, then send (`output, sid, resV`) to the adversary, and receive back (`update, sid, b`). Update  $res_V := b$ .
- If  $res_P = 0$ , then send (`not_ok, sid`) to the environment over Prover's connection to the environment, otherwise send (`ok, sid`) over Prover's connection.
- If Prover is corrupted, then wait for (`proceed, sid`) from the adversary.
- If  $res_V = 1$ , then send (`ok, sid`) to the environment over Verifier's connection to the environment, otherwise send (`not_ok, sid`) over Prover's connection.

We see that there are some values that  $\mathcal{I}$  does not attempt to keep private. We have discussed previously that we consider the leak of  $H(\mathbf{S})$  acceptable. We also see that the adversary determines, what a corrupt Prover or Verifier returns to the environment; this design choice is a standard one. Finally, we see that if  $\mathbf{S}$  does not satisfy the policy, then there is no way for an uncorrupted Verifier to return `ok`.

### B.3 Real system

The real system consists of (the Turing machines realizing the steps of) the Witness device, the Prover and the Verifier. All machines are parametrized with the hash function  $H$ . The Prover and the Verifier are also parametrized with the relation  $R$ .

The Prover and the Verifier implement the ZKP protocol for the relation  $R$ . In order to compartmentalize it, we make use of the composability properties of the UC framework. Namely, we introduce a fourth machine  $\mathcal{F}_{\text{ZK}}^R$  to the real system. This machine is an ideal functionality modelling a zero-knowledge protocol for the relation  $R$ . The functionality is defined in [4, Fig. 11], specialized from the functionality given in [13]. It accepts inputs  $(\text{prove!}, \text{sid}, x, w)$  from the Prover machine and  $(\text{prove?}, \text{sid}, x')$  from the verifier machine, and sends back  $(\text{proven}, \text{sid})$  to the Verifier machine if  $R(x, w)$  holds,  $x = x'$ , and the adversary allows it to proceed. Given the description of  $R$  as an arithmetic circuit, there are a number of protocols that provide a secure implementation of  $\mathcal{F}_{\text{ZK}}^R$ . These secure implementations consist of a prover machine for ZKP, and a verifier machine for ZKP; in the actual deployment, these machines would be a part of our Prover and Verifier.

**Witness device** The Witness device maintains the list of coordinates  $\mathbf{S}$ . It works as follows:

- On input  $(\text{init}, \text{sid})$  from the environment, generates a key pair  $(pk, sk)$  for signing. Sends  $(\text{witpk}, \text{sid}, pk)$  to the Prover and the Verifier. Initializes the sequence of coordinates  $\mathbf{S}$  to the empty sequence.
- On input  $(\text{move}, \text{sid}, (x, y))$  from the environment, appends the coordinates  $(x, y)$  to the sequence of coordinates  $\mathbf{S}$ .
- On input  $(\text{getcoords}, \text{sid})$  from the Prover machine, sends  $(\text{coords}, \text{sid}, \mathbf{S}, \sigma)$  back to the Prover machine, where  $\sigma = \text{sig}_{sk}(H(\mathbf{S}))$ .

We see that the description of the Witness device very much matches our system model in Sec. 4.

**Prover machine** The Prover machine gets the list of coordinates from the Witness device, and uses them to present a ZKP of policy satisfaction to the Verifier (making use of  $\mathcal{F}_{\text{ZK}}^R$ ). It works as follows:

- On input  $(\text{witpk}, \text{sid}, pk)$  from the Witness device, store  $pk$ .
- On input  $(\text{prove}, \text{sid}, AD_P)$  from the environment, send  $(\text{getcoords}, \text{sid})$  to the witness device, and expect back  $(\text{coords}, \text{sid}, \mathbf{S}, \sigma)$ . Verify the signature  $\sigma$  on  $H(\mathbf{S})$  using the public key  $pk$ , and check whether  $R(\mathbf{S}, AD_P)$  holds. If not, send  $(\text{not\_ok}, \text{sid})$  to the environment. If yes, send  $(\text{sig}, \text{sid}, H(\mathbf{S}), \sigma)$  to the verifier machine, and  $(\text{prove!}, \text{sid}, (AD_P, H(\mathbf{S})), \mathbf{S})$  to  $\mathcal{F}_{\text{ZK}}^R$ .

The adversary may corrupt the Prover machine by sending it the  $(\text{corrupt})$ -command at the beginning of the execution. In this case, the Prover machine sends  $AD_P$ ,  $\mathbf{S}$  and  $\sigma$  also to the adversary. Also, the adversary controls, what the Prover machine sends to the Verifier, to  $\mathcal{F}_{\text{ZK}}^R$ , and to the environment.

**Verifier machine** The Verifier machine accepts the proof from the Prover machine (via  $\mathcal{F}_{\text{ZK}}^R$ ) It works as follows:

- On input  $(\text{witpk}, \text{sid}, \text{pk})$  from the witness device, store  $\text{pk}$ .
- On input  $(\text{verify}, \text{sid}, \text{AD}_V)$  from the environment, expect  $(\text{sig}, \text{sid}, h, \sigma)$  also from the Prover machine. Verify  $\sigma$  on  $h$ , using the public key  $\text{pk}$ . If it did not verify, send  $(\text{not\_ok}, \text{sid})$  to the environment. If it verified, then send  $(\text{prove?}, \text{sid}, (\text{AD}_V, h))$  to  $\mathcal{F}_{\text{ZK}}^R$ . On input  $(\text{proven}, \text{sid})$  from  $\mathcal{F}_{\text{ZK}}^R$ , send  $(\text{ok}, \text{sid})$  to the environment.

Verifier machine may be corrupted. In this case, the adversary controls what the verifier machine sends to  $\mathcal{F}_{\text{ZK}}^R$  and to the environment. Also in this case, the Verifier machine sends  $\text{AD}_V$  and  $h$  to the adversary.

## B.4 Simulators

The goal of this section is to show that the real system is *at least as secure as* the ideal functionality. We have to show that for any environment  $\mathcal{Z}$  connecting to the interface described in Sec. B.1 of the real system, and for any adversary  $\mathcal{A}$  connecting to the adversarial interface of both the real system and the environment  $\mathcal{Z}$ , there exists an adversary  $\mathcal{S}$  connecting to the adversarial interface of both the ideal functionality  $\mathcal{I}$  and the environment  $\mathcal{Z}$ , such that the environment  $\mathcal{Z}$  cannot distinguish whether it is executing together with the real system and  $\mathcal{A}$ , or with the ideal functionality and  $\mathcal{S}$ .

Given  $\mathcal{A}$ , we construct  $\mathcal{S}$  as the composition  $\text{Sim} \parallel \mathcal{A}$ , where  $\text{Sim}$  is an interactive Turing machine. In the rest of this section, we give the description of  $\text{Sim}$  and argue that it indeed makes  $\text{Sim} \parallel \mathcal{A}$  a suitable adversary  $\mathcal{S}$ . Note that the machine  $\text{Sim}$  basically has two interfaces:

- At one side, it is able to connect to the interface that  $\mathcal{I}$  offers to the adversary.
- At the other side, it must provide the same interface to  $\mathcal{A}$  that is provided by the real system.

In order to show that the simulator  $\text{Sim}$  turns any  $\mathcal{A}$  to a suitable  $\mathcal{S}$ , we have to show that the real system is indistinguishable to the composition of the ideal functionality and the simulator. This indistinguishability must be for a distinguisher that connects to both the interface for the environment (Sec. B.1), as well as to the interface for  $\mathcal{A}$  provided by the real system.

The simulator “simulates” the executions of uncorrupted parties, perhaps with less inputs than the actual parties would have. If neither the Prover nor the Verifier have been corrupted, then the construction of the simulator is trivial. In this case, the simulator receives  $H(\mathbf{S})$ ,  $\text{AD}_P$  and  $\text{AD}_V$  from  $\mathcal{I}$ . It does not have to send any messages itself, except for notifying  $\mathcal{A}$  that a ZK proof is currently running by  $\mathcal{F}_{\text{ZK}}^R$ .

If either the Prover or the Verifier has been corrupted, then the simulator has to construct the messages that the corrupted party would send to the adversary  $\mathcal{A}$ . Also, the simulator has to be able to handle the messages that the corrupted

party sends out; these messages have been constructed by  $\mathcal{A}$ . When  $\mathcal{A}$  sends out the corruption request, the simulator forwards it to  $\mathcal{I}$ . The following simulation depends on which party was corrupted. As the request comes out at the beginning of the execution, we can split the construction of the simulator to two parts.

**Simulator for corrupted Verifier** During the initialization, the simulator generates a public-private key pair  $(pk, sk)$ . Sends  $(witpk, sid, pk)$  to the adversary (as Witness device sending a message to the Verifier).

During proving, simulator gets  $AD_P, AD_V$  and the hash  $h$  (and  $sid$ ) from the ideal functionality. Signs  $h$  using  $sk$ , obtaining  $\sigma$ . Also gets the value  $res_V$  from the ideal functionality. Sends  $(sig, sid, h, \sigma)$  to the adversary (as Prover sending a message to the Verifier). Gets back  $(prove?, sid, (AD_V, h))$  as a message from the Verifier to  $\mathcal{F}_{ZK}^R$ . If the the value  $h$  differs from its previous value, or if  $AD_V$  differs from  $AD_P$  received previously, then  $Sim$  sets  $res_V := 0$ . Sends  $(update, sid, res_V)$  to the ideal functionality.

We see that if the corrupted Verifier (controlled by the adversary) follows the protocol, submitting  $h$  and  $AD_V$  to the ZK proof system, then the output that the environment receives from the Verifier is the same with both the real system and with the ideal functionality, depending on the values  $S$  and  $AD_P$  that the environment submits through Prover's interface. If the corrupt Verifier changes the values it gives to  $\mathcal{F}_{ZK}^R$ , then the output to the environment will be `not_ok`.

**Simulator for corrupted Prover** During the initialization, the simulator generates a public-private key pair  $(pk, sk)$ . Sends  $(witpk, sid, pk)$  to the adversary (as Witness device sending a message to the Prover).

During proving,  $Sim$  gets  $S, AD_P$ , and  $AD_V$  from the ideal functionality  $\mathcal{I}$ . Computes  $\sigma$  on  $S$ , using  $sk$ . Receives  $(getcoords, sid)$  from the adversary (as Prover sending a message to the Witness device). Sends  $(coords, sid, S, \sigma)$  back. Will again get these values from the adversary (as Prover submitting its inputs to  $\mathcal{F}_{ZK}^R$ ). Will also get the value  $b$  that the adversary  $\mathcal{A}$  wants the Prover machine to return to the environment. Sends it to the ideal functionality as  $(update, sid, b)$ . If the adversary has sent a different  $AD_P$  to  $\mathcal{F}_{ZK}^R$ , or it has changed  $S$ , then do not send  $(proceed, sid)$  to the ideal functionality.

The difference of the real system, and the composition of the ideal functionality and simulator is the following. In the real system,  $\mathcal{F}_{ZK}^R$  is used to evaluate  $S$  and *authority's data*. This evaluation is done by executing  $R$  on these data, and the hash function and the signature are used to protect the integrity of the witness that goes into  $\mathcal{F}_{ZK}^R$ . In the ideal system, the ideal functionality will evaluate the policy on  $S$  and  $AD$ ; the integrity of  $S$  holds by default because it is not moved around before policy evaluation. Hence we have to argue that  $R$  is a correct implementation of the policy. We also have to argue that the probability of using different  $S$  in the real system and in the composition of the ideal system and the simulator simulator, is negligible.

We have argued that  $R$  is a correct implementation of the policy in Sec. 5.1 and in Sec. 5.2. Also, changing  $S$  in the real system requires the adversary to either forge the signature or find a different pre-image of the hash function.