Privacy-Preserving Trajectory Matching on Autonomous Unmanned Aerial Vehicles

Savio Sciancalepore
s.sciancalepore@tue.nl
Eindhoven University of Technology (TU/e)
Eindhoven, Netherlands

Dominik Roy George
d.r.george@tue.nl
Eindhoven University of Technology (TU/e)
Eindhoven, Netherlands

ABSTRACT

Autonomous Unmanned Aerial Vehicles (UAVs) are increasingly deployed nowadays, thanks to the additional features and enhanced flexibility they provide, e.g., for transportation and goods delivery. On the one hand, discovering in advance collisions occurring with other UAVs in the future could enhance the efficiency of the path planning, reducing further the delivery time and UAVs' energy consumption. On the other hand, location and timestamps–key to detecting and avoiding collisions in advance–are sensitive and cannot be shared indiscriminately with untrusted entities.

This paper solves the aforementioned challenging problem by proposing PPTM, a new protocol for efficient and effective privacy-preserving trajectory matching on autonomous UAVs. PPTM allows two UAVs, possibly not connected to the Internet, to discover any spatial and temporal collisions in their future paths, without revealing to the other party anything else than the colliding time and coordinates. To this aim, PPTM grounds on a dedicated tree-based algorithm, namely, *Incremental Capsule Matching*, tailored to the unique features of spatio-temporal data, and it also integrates a lightweight privacy-preserving proximity testing solution for performing private comparisons. We tested our solution on real devices with heterogeneous processing capabilities (a regular laptop, a tiny processing unit, and a mini-drone), showing that PPTM can perform privacy-preserving trajectory matching even in a few milliseconds, up to 98.27% quicker compared to the most efficient competing solution.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; **Security protocols**; **Privacy-preserving protocols**.

ACM Reference Format:
Savio Sciancalepore and Dominik Roy George. 2022. Privacy-Preserving Trajectory Matching on Autonomous Unmanned Aerial Vehicles. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3564625.3564626

ACSAC '22, December 5–9, 2022, Austin, TX, USA

ACM ISBN 978-1-4503-9759-9/22/12…$15.00
https://doi.org/10.1145/3564625.3564626

1 INTRODUCTION

Autonomous Unmanned Aerial Vehicles (UAVs) are increasingly deployed in a plethora of application domains [17, 21]. For instance, ships are increasingly adopting autonomous UAVs for underwater explorations [14], and goods delivery companies such as Amazon already obtained the necessary licenses to deliver goods to customers using autonomous drones [30]. According to the most recent analyses, the autonomous vehicles market size is expected to reach 888.40 Billion USD by 2028, with a rapid annual growth [13].

With such an enormous increase of deployed autonomous UAVs, detecting in advance possible collisions and avoiding them becomes crucial for enhanced service provisioning and people' safety [12, 33]. Besides, discovering in advance collisions occurring in the vehicles' paths (namely, *finding matches in their trajectories*) might help redesign the whole path of the UAV, reducing delivery times and energy consumption. In this context, the most straightforward and diffused solution is to let the vehicles exchange path-related data, including the scheduled locations and timestamps, allowing them to estimate mutual distances. However, the indiscriminate disclosure of spatio-temporal UAVs' data poses severe privacy issues, as an attacker can use such information for several malicious purposes, e.g., obtaining the location of a goods storage area, identifying areas where vehicle capture is more feasible, and discovering users' additional private information.

To the best of our knowledge, privacy-preserving trajectory matching on autonomous UAVs is a novel research problem. Although a few contributions in the context of Online Social Networks (OSNs) considered similar problems (see Sec. 2 for an overview), such schemes only look at side information (e.g., the number of matching points), and they are not intended to work on standalone devices, in real-time. Thus, they require multiple complex and time-consuming operations, hardly affordable on regular computing devices, forcing the vehicles to integrate a high-speed Internet connection. In addition, they involve dedicated servers to perform the necessary computations, and only consider location matching, but never time matching between trajectories. Conversely, drones often operate in remote locations, where Internet connectivity is scarce, unreliable, or not available at all. Thus, any protocol intended to perform privacy-preserving trajectory matching with selected UAVs should run onboard, without relying on the cooperation of any external entity.

Contribution. In this paper, we present *PPTM*, an efficient and effective solution to achieve *P*rivacy-*P*reserving *T*rajectory *M*atching on autonomous UAVs. In brief, *PPTM* allows standalone UAVs, possibly not equipped with outsourcing capabilities, to discover spatio-temporal proximity in their complete trajectories, without revealing to the other party anything else than the colliding time

and coordinates. To achieve such an ambitious objective, *PPTM* efficiently combines a new dedicated tree-based algorithm for the analysis of spatio-temporal data, namely, *Incremental Capsule Matching*, with existing privacy-preserving proximity testing solutions, traditionally adopted in the context of OSNs. The result is a lightweight solution, which can run efficiently even on constrained devices. We present two versions of *PPTM*, namely, the *Truncated Mode* and the *Full Mode*, the former being the most lightweight, at the expense of a few false positives. We also deployed *PPTM* on three different platforms, i.e., a regular laptop, a tiny processing unit (Raspberry Pi), and a mini-drone (ESPCopter), showing that it can perform privacy-preserving trajectory matching with latency gains up to 98.27% compared to the most efficient competing solution currently available, with a remarkable impact on the path planning capabilities of modern UAVs.

Roadmap. This paper is organized as follows: Sec. 2 reviews related work, Sec. 3 presents the scenario and adversary model, Sec. 4 provides the details of *PPTM*, Sec. 5 analyzes the security of *PPTM*, Sec. 6 provides the experimental assessment and comparison, and finally, Sec. 7 tightens the conclusions.

2 RELATED WORK

The *trajectory matching* problem tackled in this manuscript is different from *collision detection and avoidance*. Indeed, collision detection implies discovering approaching collisions based on the current location of the UAVs [35]. Such a problem is addressed in modern UAVs through the integration of dedicated sensors and technologies on board, e.g., cameras, microphones, LIDARs, and RADARs. However, being based on the acquisition of physical values from the surroundings of the vehicles, the aforementioned technologies cannot predict collisions that will occur in the future. Thus, they cannot be used to optimize the path of the vehicle and to increase its efficiency. Conversely, *trajectory matching* refers to the problem of finding intersections in the overall path of multiple autonomous vehicles. Performing such a task without disclosing trajectory data has been investigated by only a few contributions. In this area, the authors in [11] proposed to identify *proximity* between a point of one trajectory and the segment identifying the trajectory of the other vehicle, then to run the same logic on all pairs of points and segments, and then to combine the results using a circuit, such as Homomorphic Encryption (HE). The repeated usage of HE techniques makes the whole solution impractical on constrained devices. The authors in [34] proposed to evaluate the collisions between two trajectories by evaluating intersections between moving circles. They used four two-party protocols, including Secure Scalar Dot-Product, Yao's millionaire protocol, and a $(Z+V)$-disguise scheme, requiring significant computational and bandwidth overhead. Recently, the authors in [27] proposed a protocol for trajectory matching among UAVs, where a trusted third party could organize the traffic in a crowded area. However, privacy issues are not considered. The contribution closest to our problem is the one in [25], identifying *ex-post* locations visited in the past by users in OSNs. The authors proposed two protocols, based on Private-Set Intersection (PSI) and Bloom Filters, respectively. Such protocols are used in an iterative cell disclosure mechanism, where the entities apply *static* position quantization techniques to identify the cells traversed by their paths. We first notice that both the methods they proposed do not guarantee collision-free routes. Indeed, if two users are located at the boundaries of neighbouring cells, within the desired distance threshold, they would be considered as not colliding, although colliding in practice. Therefore, such a method is not fully reliable. Their first proposed method adopts the PSI-Cardinality method introduced in [7], and it is the most efficient at the time of this writing when involving traditional encryption techniques. In Sec. 6 we will show that it still requires significant computations, being hardly applicable on autonomous UAVs. Their second method is based on bloom filters, and allows the entities to verify trajectory intersection by testing membership in the exchanged bloom filters. Such an approach is also framed in the context of Device-to-Device Communications (D2D) in a later extension of the work in [26]. While this latter approach might be very efficient, it does not consider the information about the *cell* as sensitive. Therefore, an attacker might easily test the membership of all the cells into a specific region and obtain information about all the cells traversed (or not), creating severe privacy issues. Conversely, our work considers any location information as *private*, making our adversary model stronger than the one considered in the latter cited paper. In addition, both protocols do not consider *time* as a dimension, as they evaluate matches on already-traversed locations. The authors in [16] improved the method in [25] by using partially overlapping grids and removing duplicate cells from the set, but the resulting computational overhead is still too high for real-time collision detection on board (see Sec. 6).

The authors in [8] recently proposed a scheme for securely detecting collisions in the path of two moving objects. Their strategy combined the concept of *Gröbner basis*—transforming a path into a reference line— and the *Sweep Line* algorithm, for finding intersections between lines, and it uses a customized version of an Oblivious Transfer (OT) protocol for secure points generation. However, their scheme incurs high computational and bandwidth overhead— in the order of n^2, with n number of points in the set. The authors in [19] proposed a secure and privacy-preserving collision avoidance system for 5G fog-enabled Internet of Vehicles (IoV). Their scheme considers over-speeding drivers, and leverages continuous interactions with powerful *Fog Nodes*. Thus, the considered scenario is different from ours, where we consider autonomous vehicles.Differently, the authors in [36] focused on Location-Based Services (LBSs), and anonymize a trajectory reported by a user through a cloaking region making it indistinguishable from at least $l-1$ other trajectories. Note that trajectory privacy in OSNs has been considered also by the authors in [24] and [31]. However, their proposals focused on the contemporary trade-off between location privacy and utility of the data, which is a completely different scenario than the ones addressed in this manuscript.

Finally, note that *PPTM* extends the scheme in [15]. On the one hand, to the best of our knowledge, such a scheme is the most efficient among the proximity detection techniques currently available. On the other hand, we remark that *PPTM* does not simply straightforwardly integrate the cited scheme, as it was thought for *static* location data. Thus, when applied to multiple locations, such a scheme would require an overhead multiple of a factor of $N \times M$, being N and M the number of trajectory points. Conversely,

PPTM extends the scheme in [15] to work with dynamic trajectories, through several technical non-straightforward improvements, including: (i) integrating it into a tree-based comparison approach; (ii) combining its logic with the division of the space into *capsule* shapes; and (iii) roto-translating randomly the points before feeding them into the algorithm (see Sec. 4). The cited modifications lead to a brand new solution for privacy-preserving trajectory matching.

3 SYSTEM AND ADVERSARY MODEL

In this section, we introduce the scenario (Sec. 3.1) and adversarial model (Sec. 3.2) considered in our work.

3.1 Scenario

We assume two generic UAVs, namely, $\mathcal{A}$ and $\mathcal{B}$, operated by different pilots and produced by different manufacturers. $\mathcal{A}$ and $\mathcal{B}$ are autonomous, i.e., they have pre-installed trajectories $\mathbf{T_A}$ and $\mathbf{T_B}$ from source points $\mathbf{T_{A,0}}$, $\mathbf{T_{B,0}}$ to destination points $\mathbf{T_{A,M-1}}$, $\mathbf{T_{B,N-1}}$, respectively, with $M \neq N$. We assume that the time instant $t_{j,m}$ when the UAV j is scheduled to *occupy* a certain location $\mathbf{T_{j,m}}$ is also known (with a given accuracy). At the same time, the time lapse between two consecutive locations in the path is variable. In particular, we assume that the motion between two consecutive path points can be approximated with small error as a straight linear motion from $\mathbf{T_{j,m}}$ to $\mathbf{T_{j,m+1}}$ with velocity $\mathbf{V_{j,m}}$.

The UAVs feature a radio communication module, allowing them to communicate wirelessly. The specific communication technology depends on the type of UAV and can have different ranges. For instance, the UAVs can adopt the WiFi-direct communication technology, allowing them to communicate directly within the ISM band $[2.4 - 2.5]$ GHz without the aid of any infrastructure element [9]. In line with currently-deployed capabilities, we also assume that an UAV can initiate and maintain a secure peer-to-peer connection to another UAV, e.g., using the well-known Transport Layer Security (TLS) protocol [1]. We recall that TLS provides peers authentication and confidentiality, while it cannot prevent privacy issues, which depend only on the specific information disclosed to the other party. Note that the mutual radio visibility among the UAVs does not imply that they are at risk of immediate collision. Indeed, UAVs using WiFi-based communication systems are characterized by relatively large ranges, up to 25 km [18].

In this context, we assume that $\mathcal{A}$ and $\mathcal{B}$ would like to detect before-hand any collisions in their path, to modify the planned route accordingly. Moreover, they would like not to share the path and time schedule, to maintain complete path privacy. Our solution, namely, *PPTM*, allows to carry out this task directly on the UAVs, being feasible also when they are not connected to Internet.

3.2 Adversary Model

The adversary assumed in this work, namely, ϵ, is an *honest-but-curious* adversary featuring both passive and active capabilities. On the one hand, ϵ is a global, spatially-unbounded, and frequency-unlimited eavesdropper, able to detect any radio activity initiated by the target UAVs. Using such capabilities, ϵ can know the MAC addresses of the UAVs and try to spoof them. On the other hand, ϵ is also equipped with a radio that can transmit packets on the same communication frequencies of the target UAV. Thus, ϵ can create its own packets, either genuine or forged, and initiate an instance of *PPTM* with a target UAVs. ϵ can also deploy its own UAV and run an instance of *PPTM* with a target one.

Overall, the aim of ϵ is to know points in the path of a target UAVs. To this aim, ϵ can run the described protocols as intended, but work offline to obtain additional private information. Such *private* details include the locations in the path and time-related aspects, i.e., the start and end time of the trip and its duration. Note that path- and time-related information must be private for the UAVs, as ϵ can use them for several malicious purposes, e.g., to launch jamming attacks, capture the UAV and any packages it delivers, disrupt its operations, or profile the UAVs.

For the readers' convenience, we report in Tab. 3 (Annex A) the adopted notation. Boldface notations (e.g., $\mathbf{a}$) refer to vectors.

4 THE *PPTM* SCHEME

In this section, we first describe in Sec. 4.1 the rationale of *PPTM* on plaintext (without trajectory privacy), while Sec. 4.2 and Sec. 4.3 discuss the *Setup* and *Trajectory Match* phases of *PPTM*, respectively.

4.1 Trajectory Match Detection Logic

This section explains the logic of *PPTM* on plaintext, without considering trajectory privacy (addressed later). For ease of discussion, we consider 2D Cartesian trajectories (x, y), as methods for switching to 3-D and geodetic coordinates are well-known [5].

PPTM relies on a dedicated tree-based algorithm, namely, *Incremental Capsule Matching*, comparing an increasing number of UAVs' sub-trajectories. Let us assume the trajectories of two UAVs, namely, $\mathbf{T_A}$ and $\mathbf{T_B}$, containing the planned location of the UAVs at a given time. The algorithm consists of multiple steps, also referred to as rounds. First, the UAVs determine who has the highest number of remaining trajectory points. Assume that the trajectories $\mathbf{T_A}$ and $\mathbf{T_B}$ consist of M and N points, respectively, with $M \geq N$. At step $i = 1$, the UAV with the highest number of points (A) computes a *reference shape*, i.e., a grid containing its overall trajectory, with the requirement that *any point* located at a distance less than the minimum threshold δ from any of the points in the trajectory $\mathbf{T_A}$ is considered as a potential *colliding point*. Thus, the cited points are all mapped into the same grid, namely, the *reference shape*. Trajectory collisions are then discovered via the identification of a matching between grid identifiers of A and B. An important element is the

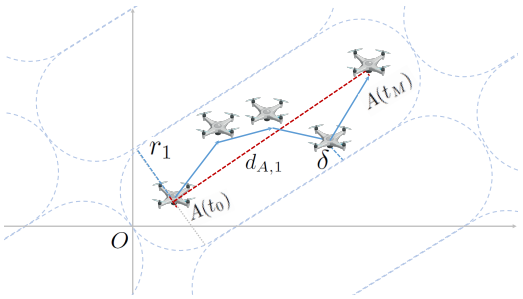

Figure 1: Tessellation logic in the *Incremental Capsule Matching* algorithm (2D space and $\mathbf{s_1} = 0$).

selection of the best shape to model such a concept. Without loss

of generality, consider a generic 2D polygon, and denote r as the radius of the circle where the polygon can be inscribed. Regular 2-D polygons other than circles (e.g., squares, pentagons, hexagons, etc.) cannot guarantee that all points whose distance from the geometrical center is less than r lies within the polygon, as for any shape there are points at a distance less than r not included in the polygon. Circles of radius r, instead, fulfil the above-mentioned property only with reference to a single location and time, while we require that the property is satisfied for a set of locations occupied between two time instants. Thus, we need a 3D shape consisting of multiple circles, whose center is a specific location occupied by the UAV. As depicted in Fig. 1, such a geometrical shape is the *capsule*, as points within a capsule have maximum distance r from its line of symmetry, being r the capsule radius [20].

Defining the *reference shape* as the *reference capsule*, we recall that a capsule is characterized by two dimensions, i.e., the radius and the length. To compute the radius of the *reference capsule* at the step $i = 1$, namely, r_1, A first computes the equation of the line merging the first and the last points of its trajectory, as in Eq. 1, where $u_{A,1}$ and $v_{A,1}$ can be found through well-known methods [32]. Note that we bound such a line from the first to the last point of the trajectory.

$$\tilde{y}_{A,1} = u_{A,1} \cdot x_{A,1} + v_{A,1}. \tag{1}$$

If needed, other elements can be included in Eq. 1, such as the effect of the wind. Then, A computes the distance $d_{m,\tilde{y}_{A,1}}$ of each point in its trajectory $\mathbf{T_A}$ from the above-cited line, and it takes the maximum distance as in Eq. 2.

$$dm_{A,1} = \max_{m} d_{m,\tilde{y}_{A,1}}. \tag{2}$$

We define δ as the collision threshold, i.e., the minimum allowed distance between points in T_A and other trajectories. A computes the *reference radius* r_1 at the step $i = 1$ as in Eq. 3, summing up the cited maximum distance, the collision threshold δ, and the term σ, modelling the maximum inaccuracy of the GPS position.

$$r_1 = dm_{A,1} + \delta + \sigma. \tag{3}$$

The *reference length* at the step $i = 1$, namely, h_1, is the sum of two components, i.e., twice the *reference radius* (considering the first and the last points of $\mathbf{T_A}$), and the Euclidean distance $d_{A,1}$ between the first and the last point of T_A, as in Eq. 4.

$$h_1 = d_{A,1} + 2r_1 = ||\mathbf{T_{A,M-1}} - \mathbf{T_{A,0}}||^2 + 2r_1. \tag{4}$$

To allow the remote UAV to map its trajectory using the capsule shape, A also needs to provide the orientation of the capsule. Thus, A computes the angle between the first and last point in its trajectory $\mathbf{T_A}$, namely, $\Theta_i = \Theta_1$, as in Eq. 5.

$$\Theta_1 = \arctan\left(\frac{T_{A,y,M-1} - T_{A,y,0}}{T_{A,x,M-1} - T_{A,x,0}}\right), \tag{5}$$

where $atan(\cdot)$ refers to the tangent arc operator. Then, A extracts two nonces $\mathbf{s_i} = \mathbf{s_1} = \left[s_{1,x}, s_{1,y}\right]$, used to translate its trajectory points by a random shift, generating a roto-traslation of the points in $\mathbf{T_A}$ (Eq. 6 and Eq. 7).

$$\begin{aligned} O_{1,x} &= x_{A,1} - r_1 - s_{1,x} \cdot h_1, \\ O_{1,y} &= y_{A,1} - r_1 - s_{1,y} \cdot 2 \cdot r_1. \end{aligned} \tag{6}$$

$$\begin{aligned} \mathbf{x'_A} &= [\mathbf{x_A} cos\,(\Theta) - \mathbf{y_A} sen\,(\Theta)] - O_{1,x}, \\ \mathbf{y'_A} &= [\mathbf{x_A} sin\,(\Theta) + \mathbf{y_A} cos\,(\Theta)] - O_{1,y}, \end{aligned} \tag{7}$$

Then, all the points in $\mathbf{T_A}$ are mapped to trajectory identifiers $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$, by dividing them for the combination of capsule dimensions, as shown in Eq. 8.

$$\begin{aligned} \tilde{\mathbf{x_A}} &= \left\lfloor \frac{\mathbf{x'_A}}{h_1} \right\rfloor, \\ \tilde{\mathbf{y_A}} &= \left\lfloor \frac{\mathbf{y'_A}}{2 \cdot r_1} \right\rfloor. \end{aligned} \tag{8}$$

Note that nonces allows to randomize the trajectory identifier of A, that would be always $[0, 0, 0]$, otherwise. A delivers to B: (i) the origin $\mathbf{O_1} = \left[O_{1,x}, O_{1,y}\right]$; (ii) the capsule length h_1; (iii) the capsule radius r_1; and, finally, (iv) the reference angle Θ_1.

B uses the received values to map the trajectory points $\mathbf{T_B}$ into the space tessellation provided by A, roto-translating its points as in Eq. 7 and taking the related position identifiers as per Eq. 8. The resulting position identifiers $[\tilde{\mathbf{x_B}}, \tilde{\mathbf{y_B}}]$ are then delivered to A.

A concludes the step $i = 1$ of the algorithm evaluating if any trajectory identifiers $[\tilde{\mathbf{x_A}}, \tilde{\mathbf{y_A}}]$ match with the remote trajectory identifiers $[\tilde{\mathbf{x_B}}, \tilde{\mathbf{y_B}}]$. If there are no matches, $\mathbf{T_A}$ and $\mathbf{T_B}$ do not collide; thus, they can stop the algorithm and output a *No Collision* decision. If there is a match, it means that there are chances of collision. However, given that the *reference capsule* was created starting from the extreme points in the trajectory $\mathbf{T_A}$, leading to a *coarse* comparison, the trajectories could also not collide. Therefore, A and B proceed further with the new step $i = 2$ of the algorithm, by dividing their *colliding trajectory parts* into two sets. Such sets share only one point, i.e., the last of the first set and the first of the second set.

In the following rounds, each portion of $\mathbf{T_A}$ and $\mathbf{T_B}$ are mutually compared. If a portion does not collide with any of the remote ones, it is discarded. Otherwise, it is still considered, and, in the following steps, its colliding points are further divided. The *Incremental Capsule Matching* algorithm stops when any of the following two conditions occur: (i) there are no further matches between any couple of capsules, indicating *No Collisions*; or (ii) there are no set of points containing more than one (1) trajectory points, indicating a *Collision* event.

As for the second condition in the above statement, we define two *versions* of the *Incremental Capsule Matching* algorithm.

- *Truncated Mode.* In case of collisions among trajectory portions, the algorithm stops if at least one of the two parties has no set of points containing more than one (1) trajectory point.
- *Full Mode.* In the case of collisions among trajectory portions, the algorithm stops only when both parties have no set of points containing more than one (1) trajectory point.

We notice that both modes guarantee zero false negatives, i.e., when the algorithm provides a *No Collision* output, there are no collisions. Indeed, the *Incremental Capsule Matching* algorithm compares *enlarged* trajectory pieces, including always all the trajectory points. Thus, the absence of collision in the enlarged trajectories implies also no collision among actual trajectory points.

At the same time, the *Full Mode* additionally guarantees zero false positives, i.e., when the algorithm provides a *Collision* output, there is always a collision. Indeed, in the final round, the capsule identifiers are created using actual location points. Thus, no additional locations outside of the *colliding region* are included in the computations. However, when trajectories collide, the latter version of the algorithm requires more comparisons and interactions between the two UAVs than the *Truncated Mode*. Indeed, both UAVs have to narrow down their initial set of points progressively to the size of 1 point before declaring the collision. Worst-case, the *Full Mode* requires a total number of $\lceil log_2 N \rceil$ rounds to identify the colliding points ($M \geq N$).

Conversely, when the trajectories collide, the *Truncated Mode* is more lightweight, as it requires only the UAV having the smallest amount of points to narrow down its set to 1 point before declaring a collision. Thus, worst-case, the *Truncated Mode* requires only $\lceil log_2 M \rceil$ rounds. Although providing zero false negatives as the *Full Mode*, this mode of the algorithm can provide false positives. Thus, when the algorithm provides a *Collision* output, there is a small chance that the UAVs are *not* going to collide. False positives occur because, at decision time, the UAV having the higher number of points still evaluates collisions based on *enlarged* capsules made up of multiple points, thus including points outside of the *colliding region*. Therefore, the *Truncated Mode* trades off computational and bandwidth overhead with false positives and the chance of small privacy leakage. In Sec. 6, we will evaluate the computational gain, false-positives rate, and privacy leakage of the *Truncated Mode* compared to the *Full Mode*.

Recall that matching between trajectory locations does not immediately imply vehicles' collision. Indeed, the UAVs collide only if they plan to be located at (approximately) the same location, *at the same time*. Therefore, if locations match, A and B also check if the time when they plan to at matching location(s) is within a given time τ. If there is a match also on time, A and B declare *collision*.

The above discussion did not intentionally consider privacy. Indeed, sharing the identifiers of any trajectory and time piece would lead each UAV to share private data. To overcome this limitation, we couple the algorithm discussed above with a privacy-preserving solution for matching elements in a private set. We report the details of such mechanisms in Secs. 4.2 and 4.3.

4.2 Setup Phase

In the setup phase, executed before starting the journey, the UAV j creates the cryptography materials necessary to run *PPTM*. They include the following parameters.

- Two functions $f, g : G \leftarrow \mathcal{G}$, in charge of mapping a set of discrete, finite, and low-entropy data G (locations and timestamps) into equal-size sets of unique values $\mathcal{G}$, representing unique identifiers.
- A generic hashing function $H(\cdot)$.
- Two prime numbers p'_j and q'_j, used to generate *safe primes* p_j and q_j, i.e., $p_j = 2p'_j + 1$ and $q_j = 2q'_j + 1$. The values of p_j and q_j also define $n_j = p_j \cdot q_j$ and $\phi(n_j) = 4p'_j q'_j$.

All the values are stored in the local vehicle memory and used during the following phases. While the hashing function $H(\cdot)$ and the size of the modular field n_j are public and shared during the initial connection setup, p_j and q_j are kept secret.

4.3 Trajectory Match Phase

In the *Trajectory Match* phase, A and B interact to establish if their trajectories *match* in any point, in space and time.

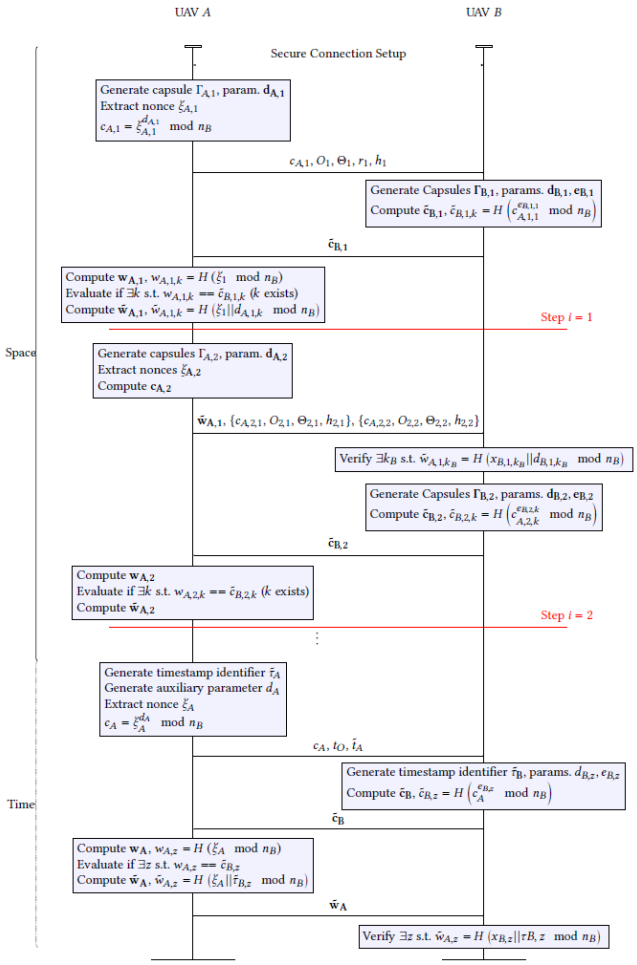

Figure 2: Sequence Diagram of *PPTM*.

This phase includes two sub-phases, i.e., *Space Trajectory Match* and *Time Trajectory Match*, dedicated to finding matches in the locations and in the time-related information, respectively. Overall, the *Time Trajectory Match* sub-phase is executed only if at least one match is found in the *Space Trajectory Match* sub-phase; otherwise, it is not executed. Note that the sequence of the above-described sub-phases is not casual. Indeed, as per Sec. 3.1, two UAVs coming in mutual visibility are both *active* at the same time. Thus, certainly, there will be collisions in the time domain (i.e., at least the current time). Conversely, collisions in space might not occur. Therefore, the (possible) absence of collisions in the space domain could prevent the execution of the *Time Trajectory Match* sub-phase, further reducing the overhead of the protocol.

Fig. 2 reports the (simplified) sequence diagram of the operations executed during the whole *Trajectory Match* phase by A and B, while we report detailed operations below.

(1) A and B first establish a secure connection, e.g., using TLS.
(2) A and B exchange the information about the number of remaining points in the trajectory. With $M \geq N$, and thus, A is the initiator and B the responder.
(3) At the step $i = 1$, A takes the first location $\mathbf{T_{A,0}}$ and last location $\mathbf{T_{A,M-1}}$ in its trajectory, and it computes the reference capsule $\Pi_{A,1}$, following the logic described in Sec. 4.1. We denote r_1 and h_1 the radius and the length of the reference capsule of A at the step $i = 1$, respectively, Θ_1 as the angle of the capsule, and O_1 as the random origin point.
(4) Then, A uses f, the origin point O_1, and the angle Θ_1 to map the capsule $\Pi_{A,1}$ to a unique identifier $\Gamma_{A,1}$. Per construction, the initiator A always has a single capsule identifying the reference trajectory $\mathbf{T_A}$. However, thanks to the nonces, the identifier value is random.
(5) A sets $\mathbf{d_{A,1}} = \left[\Gamma_{A,1}\right]$.
(6) Then, A extracts a nonce ξ_1, and computes the encrypted challenge $c_{A,1}$ as per Eq. 9.

$$c_{A,1} = \xi_{A,1}^{d_{A,1}} \mod n_B. \tag{9}$$

(7) Then, A delivers a wireless message on the secure connection, containing: (i) the encrypted challenge $c_{A,1}$, (ii) the origin point O_1, (iii) the angle Θ_1, (iv) the radius of the capsule r_1, and (v) the length of the capsule h_1.
(8) At message reception, B first uses O_1, Θ_1, r_1, and h_1 to map its trajectory into the space tessellation instructed by A. Note that, given that the trajectories of A and B are different, the trajectory of B could result in many identifiers. Let us define $\gamma_{\mathbf{B,1}} = \left[\gamma_{B,1,1}, \ldots, \gamma_{B,1,K}\right]$ the set of K capsule identifiers whose set of points contains at least a single point of the trajectory of B.
(9) For each unique $k-th$ capsule identifier, B sets $d_{B,1,k} = \Gamma_{B,1,k}$, and it computes the parameter $e_{B,1,k} = \frac{1}{d_{B,1,k}} \mod \phi(n_B)$.
(10) Then, for each $k - th$ capsule identifier, B computes the encrypted response $x_{B,1,k}$, as per Eq. 10.

$$x_{B,1,k} = c_{A,1}^{e_{B,1,k}} \mod n_B. \tag{10}$$

Afterwards, B computes the digest of each encrypted response $x_{B,1,k}$, namely $\tilde{c}_{B,1,k}$, through the hash function H, as in Eq. 11.

$$\tilde{c}_{B,1,k} = H\left(x_{B,1,k}\right). \tag{11}$$

(11) B sends back to A the vector of encrypted responses $\tilde{\mathbf{c}}_{\mathbf{B,1}} = \left[\tilde{c}_{B,1,1}, \ldots, \tilde{c}_{B,1,K}\right]$.
(12) At message reception, A compares each $\tilde{c}_{B,1,k}$ in $\tilde{\mathbf{c}}_{\mathbf{B,1}}$ with the *check values* $w_{A,1,k}$, computed as in Eq. 12.

$$w_{A,1,k} = H\left(\xi_1 \mod n_B\right). \tag{12}$$

Note that, if $\exists d_{B,1,k}$ s.t. $d_{A,1} = d_{B,1,k}$, then:

$$H\left(c_{A,1}^{e_{B,1,k}} \mod n_B\right) = H\left(\left(\xi_1^{d_{A,1}}\right)^{e_{B,1,k}} \mod n_B\right) = \\ = H\left(\xi_1 \mod n_B\right). \tag{13}$$

Thus, $w_{A,1,k} = \tilde{c}_{B,1,k}$, meaning that the capsule identifiers of A and B are the same. Otherwise, if the the capsule identifiers are all different, then $w_{A,1,k} \neq \tilde{c}_{B,1,k}, \forall k \in K$.
(13) A also computes the *encrypted proof of proximity*, namely, $\tilde{w}_{A,1}$, as per Eq. 14.

$$\tilde{w}_{A,1} = H\left(\xi_1 || d_{A,1} \mod n_B\right), \tag{14}$$

where the notation $||$ refers to the concatenation.
(14) If $w_{A,1,k} \neq \tilde{c}_{B,1,k}, \forall k \in K$, there are no intersections between the trajectories. Therefore, A delivers to B only $\tilde{w}_{A,1}$, to let B verify the absence of intersections.
(15) Conversely, if $\exists d_{B,1,k}$ s.t. $d_{A,1} = d_{B,1,k}$, A starts the step $i = 2$, dividing the set of trajectory points into two sets of size $\frac{M}{2}$, each generating one capsule, namely, $\mathbf{\Gamma_{A,2}} = \left[\Gamma_{A,2,1}, \Gamma_{A,2,2}\right]$.
(16) Then, in the generic step i, for each capsule $\Gamma_{A,i,k}$, A executes the operations at the steps (5), (6), and (7). Thus, A computes $c_{A,i,k}$ using fresh nonces $\xi_{A,i,k}$, as in Eq. 9. In addition to $\tilde{\mathbf{w}}_{\mathbf{A,i-1}} = \tilde{w}_{A,1}$, A delivers to B also a set of tuples $\{c_{A,i,k}, O_{i,k}, \Theta_{i,k}, r_{i,k}, h_{i,k}\}$.
(17) At message reception, B first checks which of its previously-generated capsule identifiers matched with the trajectory of A. To this aim, $\forall k_B$, it executes Eq. 15.

$$\tilde{w}_{A,i-1,k_B} \stackrel{?}{=} H\left(x_{B,i-1,k_B} || d_{B,i-1,k_B} \mod n_B\right), \tag{15}$$

where the values of $x_{B,i-1,k_B}$ can be immediately plug into the equation as being previously computed. If $\nexists k$ satisfying Eq. 15, *PPTM* stops, and the two UAVs can continue their travel without trajectory modifications.
(18) $\forall k$ s.t. Eq. 15 holds, B takes the correspondent set of colliding coordinates and, for each of them, it executes the operations described at the steps (8) to (11).
(19) *PPTM* continues iteratively up to the occurrence of one of the two conditions listed in Sec. 4.1 for the *Incremental Capsule Matching* algorithm.
- At the step i, $\nexists k_{A,i}, k_{B,i}$ satisfying Eq. 15.
- No capsules can be created further by the responder (*Truncated Mode*) or by both the initiator and the responder (*Full Mode*), i.e., all the trajectory points subsets have size 1.

If the output of the *Space Trajectory Match* is negative (i.e., no match), *PPTM* ends. Otherwise, for any couple of points colliding, A and B run the *Time Trajectory Match* sub-phase, described below. This sub-phase uses the same logic of the previous one, but on *time* components instead of *space*-related ones. In the following, for ease of presentation, we assume that all the colliding points are consecutive, i.e., they follow one to the other. However, the scheme holds also when the paths collide in multiple portions.

- Define $\mathbf{t_A}$ the timestamps of the colliding points of A from the previous sub-phase, with $\tilde{t_A} = t_{A,end} - t_{A,1}$. A selects a random *origin timestamp* t_O, and obtains the timestamp identifier $\tau_A = \left\lfloor \tilde{t_A} - t_O \right\rfloor$. Then, using the function g (Sec. 4.2), it obtains a unique identifier of the time frame, namely, $\tilde{\tau_A}$.
- A sets $d_A = \tilde{\tau_A}$.
- Then, A extracts a random nonce ξ_A and computes the *encrypted challenge* c_A, as in Eq. 16.

$$c_A = \xi_A^{d_A} \mod n_B. \tag{16}$$

- Then, A delivers to B the encrypted challenge c_A, the *origin timestamp* t_O, and the *time division* $\tilde{t_A}$.
- At message reception, B takes the timestamps of the first and second point of any colliding capsule k_B from the previous sub-phase, namely, $t_{B,k_B,0}$ and $t_{B,k_B,1}$, re-scales them through the *origin timestamp* t_O, and divides them by the *time division* $\tilde{t_A}$, to obtain the list of identifiers $\tilde{\tau_{\mathbf{B}}} = \left[\tilde{\tau_{B,1}}, \ldots, \tilde{\tau_{B,Z}}\right]$. Then, for each of them, it generates the parameter $e_{B,z} = \frac{1}{\tilde{\tau_{B,z}}}$, and the *encrypted response* $\tilde{x}_{B,z}$, as in Eq. 17

$$\tilde{x}_{B,z} = c_A^{e_{B,z}} \mod n_B. \tag{17}$$

  Then, B computes the digest of each encrypted response $\tilde{x}_{B,z}$, namely, $\tilde{c}_{B,z}$, as in Eq. 11.

$$\tilde{c}_{B,z} = H\left(\tilde{x}_{B,z}\right). \tag{18}$$

  B delivers to A $\tilde{\mathbf{c}}_{\mathbf{B,z}} = \left[\tilde{c}_{B,1}, \ldots, \tilde{c}_{B,Z}\right]$.
- At the reception of the message from B, A generates the *check value*, namely, w_A, as per Eq. 14.

$$w_A = H\left(\xi_A \mod n_B\right). \tag{19}$$

- Then, A checks if $\exists z$ s.t. $w_A \stackrel{?}{=} \tilde{c}_{B,z}$. If the check is verified for any $\tilde{c}_{B,z}$, then the two UAVs are going to be *close* (less than the guard space δ) at the same time, thus colliding. Therefore, they can trigger collision resolution actions (see below). Otherwise, A and B are going to be at the same place, but at different time instants, not risking collision. Therefore, *PPTM* can safely end.
- A allows B to verify matches, by sending back the *encrypted proof of proximity*, namely, $\tilde{w}_A$, as per Eq. 20.

$$\tilde{w}_A = H\left(\xi_A || d_A \mod n_B\right), \tag{20}$$

- To check for time overlap, $\forall z$, B executes Eq. 21.

$$\tilde{w}_{A,z} \stackrel{?}{=} H\left(\tilde{x}_{B,z} || \tilde{\tau_{B,z}} \mod n_B\right), \tag{21}$$

  If $\exists z$ s.t. Eq. 21 is satisfied, A and B will be in close proximity, at the same time. Therefore, they can trigger collision resolution actions. Otherwise, there is no collision.

Collisions Resolution. *PPTM* identifies spatio-temporal matching in the path of two communicating UAVs. Therefore, resolving collisions is not the aim of *PPTM*, and can be done through many strategies. E.g., A and B can adjust the coordinates of the colliding points and rerun *PPTM* to verify the absence of collisions. Otherwise, they can run other solutions based on mutual coordinates agreement, using the previously-established secure connection.

5 SECURITY CONSIDERATIONS

In this section, we first use the automated tool ProVerif to prove that the single step of *PPTM* achieves the desired security objective, i.e., location privacy (Sec. 5.1). Then, through sound probability theory, we quantify the probability that the adversary guesses a point in the trajectory of the target with a given error margin, based on the output of the *Incremental Capsule Matching algorithm* (Sec. 5.2).

5.1 Formal Security Analysis via Proverif

The most important security feature offered by *PPTM* is *Trajectory Privacy*. Indeed, *PPTM* integrates a customization of the privacy-preserving proximity testing solution proposed by the authors in [15], providing either partial (*Truncated Mode*) or complete (*Full Mode*) trajectory privacy, while still allowing to identify colliding locations. In the following, in line with other works in the literature [23, 28, 29], we apply ProVerif to formally verify the security of the single step of our scheme, as well as to confirm that our construction using such a protocol does not break its security.

ProVerif assumes the underlying cryptographic primitives at the roots of the protocols are robust, and that the adversary is aware of the cryptographic procedures and the public parameters of the protocol [4]. Based on these assumptions, ProVerif formally tests the security of the protocol against the Dolev-Yao attacker model, i.e., an adversary able to access, modify, delete, and forge new messages on the public communication channel. If ProVerif identifies an attack, it also lists the steps to break the protocol.

As regarding the properties of our interest, recall that ProVerif provides the output *not attacker(elem[]) is true* when the attacker cannot retrieve the value of *elem*. Conversely, if the output is *not attacker(elem[]) is false*, the attacker can retrieve the value of *elem*. Similarly, ProVerif provides the output *weak secret(elem[]) is true* when the attacker cannot execute guessing attacks on the value *elem* or brute-force it, while it provides the outcome *weak secret(elem[]) is false* when the attacker is able to guess or brute-force the value *elem*. Finally, the output *non-interference (elem[]) is true* indicates that the attacker cannot distinguish if the input of the protocol is changing; otherwise, ProVerif returns the output *non-interference (elem[]) is false*. Fig. 3 shows the output provided by Proverif. Interested readers can use the source code we released at [6] to reproduce our results and further extend them in customized use-cases. The

```
Verification summary:
Weak secret dA_i is true.
Weak secret dB_i_k is true.
Query not attacker(dA_i[]) is true.
Query not attacker(dB_i_k[]) is true.
Non-interference dB_i_k is true.
Non-interference dA_i is true.
```

Figure 3: Verification results of the ProVerif tool.

outcome of ProVerif shows that: (i) the private information (locations or timestamps) $d_{A,i}$ and $d_{B,i,k}$ at the step i of *PPTM* are not exposed to the attacker; (ii) the attacker cannot guess/brute-force such locations when used in the protocol, despite their inherent low entropy, and (iii) the location identifiers keep changing, providing strong secrecy. Thus, ProVerif confirms that the single step of *PPTM* achieves the desired properties, while ensuring collisions discovery.

5.2 Additional Security Considerations

In the previous section, we formally proved the security of the single step of *PPTM*. However, *PPTM* includes multiple steps and, at each step, the communicating entities use as private input trajectory identifiers (i.e., capsules) representative of a different group of

trajectory points. In the following, using sound probability theory, we formally define the guessing probability of the remote party (i.e., the *honest-but-curios* adversary) at the step i of *PPTM*, namely, $p_c(i)$, i.e., the probability that the adversary guesses correctly a location traversed by the remote party, using the information available at the step i of *PPTM*. Specifically, note that *PPTM* ends in two cases: (i) no colliding trajectory identifiers are found at the step i, or (ii) a number κ_i of colliding trajectory identifiers are found at the step i, but energy constraints force the protocol to stop. We discuss the two cases in Props. 5.1 and 5.2, respectively.

Proposition 5.1. *Assume that* PPTM *stops at the step i, because of no colliding trajectory identifiers in their private set. Then, the probability for an honest-but-curious adversary to guess the $\tilde{k}$ trajectory identifiers of radius δ traversed by the remote party and used in the protocol at the step i is $p_c(i) = \frac{\tilde{\kappa}}{2^{(\lceil log_2(M) \rceil - i)} \cdot (U_i - \xi_i)}$, being ξ_i the number of challenges sent by the adversary, U_i the overall number of element identifiers at the step i, and M the overall trajectory points.*

Proof. Assume that *PPTM* stops at the step i because A and B do not have any colliding trajectory identifiers. We can define the probability for an honest-but-curious adversary to guess the correct element identifiers possessed and used by the remote party at the step i (namely, $p_B(i)$) as in Eq. 22.

$$p_B(i) = \frac{\kappa_i}{U_i - \xi_i}, \tag{22}$$

where ξ_i is the number of encrypted challenges sent by A to B at the step i, κ_i is the number of encrypted responses sent back by B to A at the step i, and U_i is the overall number of available trajectory identifiers at the step i. Note that, per construction, $p_B(i) \in [0, 1]$. We now link $p_B(i)$ to $p_C(i)$, i.e., the probability of guessing a location travelled by the target entity. Recall that, at the step i, we consider trajectory identifiers representative of 2^{MAX-i} of distinct trajectory points, where $MAX = \lceil log_2(M) \rceil$. Also, at the step MAX, the trajectory identifiers (capsules) have a radius equal to the safety radius δ. Considering that the κ_i colliding trajectory identifiers at the step i are representative of $\tilde{\kappa}$ location identifiers in the trajectory of the remote party, we can derive $p_c(i)$ as in Eq. 23.

$$p_c(i) = \frac{\tilde{\kappa}}{2^{(\lceil log_2(M) \rceil - i)} \cdot (U_i - \xi_i)}. \tag{23}$$

□

Proposition 5.2. *Assume that at the step i,* PPTM *finds κ_i colliding element identifiers in the private set of A and B, but the peers stop* PPTM. *Then, the probability for an honest-but-curious adversary to guess one of the $\tilde{k}$ location identifiers possessed by the remote party and used at the step i of the protocol is $p_c(i) = \sum_{j=1}^{\kappa_i} \frac{\tilde{k_j}}{2^{(MAX-i)}}$, with $MAX = \lceil log_2(M) \rceil$ and $\sum_{j=1}^{\kappa_i} \tilde{k_j} = \tilde{k}$.*

Proof. Assume that, at the step i, *PPTM* finds κ_i colliding trajectory identifiers in the private set of A and B, but the peers stop the protocol. Recall that the trajectory identifier at the step i of *PPTM* is representative of a group of 2^{MAX-i} identifiers of radius δ, being δ the safety radius and $MAX = \lceil log_2(M) \rceil$. Also, consider that each of the κ_i colliding identifiers is representative of a number $\tilde{k_j}$ of *occupied trajectory identifiers* of radius δ, and that, per definition, $\sum_{j=1}^{\kappa_i} \tilde{k_j} = \tilde{k}$. Thus, the probability $p_c(i)$ of guessing an occupied trajectory identifier (i.e, capsule) of radius δ of the remote party at the step i of *PPTM* is defined by Eq. 24.

$$p_c(i) = \sum_{j=1}^{\kappa_i} \frac{\tilde{k_j}}{2^{(MAX-i)}}. \tag{24}$$

□

6 PERFORMANCE ASSESSMENT

In this section, we evaluate the performance of *PPTM* both via simulations (Sec. 6.1) and experiments on actual hardware (Sec. 6.2).

6.1 Simulations

We first implemented *PPTM* using Matlab *R2021b*, and we evaluated the performance of its two different *versions*. Specifically, we set up a grid of size 10,000 m ×10,000 m ×100 m, and we generated 1,000 couples (i.e., 2,000) of *random walk* trajectories, each with a random number of path points, length, origin, and destination. Although more complex mobility models could be considered, we selected the *random walk*. Indeed, it is immediately applicable to drones movements, typically not affected by obstacles. Second, *random walk* trajectories are characterized by unpredictably large blending angles, loops, and self-overlapping paths, representing worst-case conditions where to test our approach. We first evaluated the *accuracy* of the *Truncated Mode* of *PPTM*, i.e., the capability to correctly identify collisions. Tab. 1 reports the statistics of our tests. The acronym *NC* refers to *No Collision*, while *C* refers to *Collision*.

Table 1: Performance of the Truncated Mode of *PPTM*.

		PPTM Output	
		NC	*C*
Ground Truth	*NC*	835	17
	C	0	165

Note that, in line with the discussion in Sec. 4.1, the *Truncated Mode* of *PPTM* never generates *false negatives*. Out of 1,000 tests, we obtained 17 false positives, reporting an overall false-positive ratio of 10.3%. False positives (and a small privacy leakage) are the price to pay for the enhanced computational gain. To quantify such a gain, we evaluated the overall *number of comparisons* required by *PPTM* in the *Full Mode* and *Truncated Mode*, being such a metric directly related to the *computational overhead* of the scheme. Indeed, for each comparison, an entity running *PPTM* should carry out one modular exponentiation and one modular inversion. Fig. 4 reports the results of our experimentation, through the Empirical Cumulative Distribution Function (ECDF). To provide a benchmark and highlight the novelty and efficiency of our solution, we also included the number of comparisons required by the algorithm in [15], i.e., the solution based on privacy-preserving comparison of each couple of points in the two trajectories, one by one. In approx. 83% of the tests, the *Truncated Mode* and the *Full Mode* require exactly the same comparisons, leading to zero false positives. In the remaining 17% of the tests, the *Truncated Mode* requires fewer comparisons (approx. a half) at the expense of false positives and small privacy leakage. Specifically, the percentile 90 of the *Truncated*

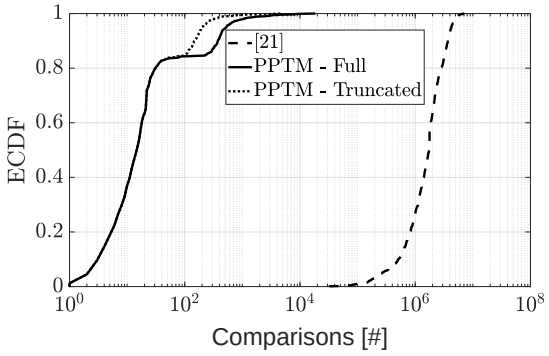

Figure 4: Comparisons required by *PPTM* (*Full* and *Truncated* mode) and the solution in [15].

Mode and the *Full Mode* amount to 145 and 378 comparisons, i.e., 4 orders of magnitude less than [15] (3, 708, 180 comparisons). Overall, our results confirm the effectiveness of our approach and the significant advantages provided by *PPTM* when compared to [15].

We also evaluated the *privacy leakage* caused by the *Truncated Mode*, i.e., the percentage of points of the *smaller* trajectory leaked to the other party. As summarized in Tab. 2, in collision cases, the *Truncated Mode* leaks on average 0.079% of the smaller trajectory, with a 95% confidence interval of 0.014%. Thus, when run on constrained platforms (see Sec. 6.2), the *Truncated Mode* trades off performance with a small privacy leakage, enabling the application of *PPTM* on the device, without requiring outsourcing.

Table 2: Non-colliding points leaked by the Truncated Mode.

	Avg.	Max	Min	95% conf. int.
PPTM Trunc. Mode	0.08%	6.7%	0	0.07%-0.09%

6.2 Performance on Real Devices and Comparison

To provide further insights into the performances of *PPTM* when run on real UAVs, we implemented and tested it on three devices characterized by heterogeneous processing capabilities.

- *Laptop Dell XPS 9560.* It is a laptop equipped with 32GB of RAM and four Intel i5-7300HQ processors running at 2.80 GHz. It models UAVs with powerful processing units.
- *Raspberry PI 3 Model B+.* It is a tiny processing unit, equipped with four 64-bit core processors running at 1.4GHz [22]. It models UAVs characterized by intermediate processing units.
- *Espcopter Mini-Drone.* It is a wirelessly networkable small-size programmable mini-drone, powered by the microcontroller ESP8266 produced by EspressIf [10]. It models the worst case of a very constrained autonomous UAV.

On the laptop and the Raspberry we adopted the python library *SAGE*, adapting the code provided in [15] to work with *PPTM*. On the *Espcopter*, we implemented *PPTM* through the *mbedTLS* library, allowing us to leverage the cryptographic hardware accelerators integrated in the *ESP8266* micro-controller of the mini-drone [2].

To provide a benchmark, we also implemented the scheme in [25], being the one directly comparable to *PPTM*. Although the authors in [25] also provided a scheme using Bloom Filters, it leaks the identifier of the cell where the UAV is, not being acceptable for our scenario. Also, recall that the protocol in [25] requires $2(N+1)$ exponentiations, with N smallest number of trajectory points.

We report the results of our analysis in Fig. 5. Each row of the figure includes two sub-figures, reporting the percentiles 80 and 95 of the completion time of *PPTM* on the specific board (recall that the percentile 80 of the two modes of *PPTM* are the same). For each platform, we also reported the performances of the protocols with three modular fields, i.e., 512, 1, 024, and 2, 048 bits, providing a security level of 160, 192, and 224 bits, respectively [3]. Note that the adoption of elliptic curves could not be beneficial in our case, as the security of *PPTM* is connected to the hardness of the factorization problem [15]. All the figures include the mean value over 1, 000 tests and the 95% confidence interval, also considering the delay of the interactions between the UAVs. Our results show that *PPTM* outperforms the competing solution on all the analyzed platforms and configurations, providing significantly reduced completion times. On the laptop, considering the modular group of 512 bits, the percentile 95 of the completion time of the *Full Mode* of *PPTM* is 0.381 s, approx. 73% less than the protocol in [25], taking 1.423 s. The gain of *PPTM* increases when considering constrained platforms, such as the Espcopter. Indeed, with the modular group of 512 bits, the *Truncated Mode* completes in 2.599 s and 16.771 s, in the 80% and 95% of cases, respectively. Compared to the time required by [25], such values are 98.27% and 92.82% less, respectively. In the same configuration, the percentiles 80 and 95 of the completion time of the *Full Mode* on the Espcopter are 98.27% and 81% less than the approach in [25], respectively. Such outstanding performances definitely enable privacy-preserving trajectory matching even on constrained devices, not equipped with an Internet connection.

7 CONCLUSIONS AND FUTURE WORK

This paper proposed *PPTM*, an effective and efficient solution for privacy-preserving trajectory matching on autonomous UAVs. *PPTM* efficiently combines a new dedicated algorithm, namely, *Incremental Capsule Matching*, with privacy-preserving proximity testing, creating a new solution working efficiently on spatio-temporal data sequences. We presented two versions of *PPTM*, namely, the *Truncated Mode* and the *Full Mode*, with the first guaranteeing a significant decrease of the computational requirements at the expense of little false positives and small privacy leakage. We first tested the performance of *PPTM* using simulations, evaluating its effectiveness in detecting matches in UAVs' trajectories, as well as the enormous computational advantages compared to traditional proximity testing. We also implemented *PPTM* on multiple heterogeneous devices (a full-fledged laptop, a mini-pc, and a constrained mini-drone) and compared its performance against available solutions. Our experiments demonstrated that *PPTM* could run even on very constrained platforms, such as the ESPcopter mini-drone, with a computational gain up to 98.27% to competing solutions, paving the way for affordable enhanced privacy in the operation of

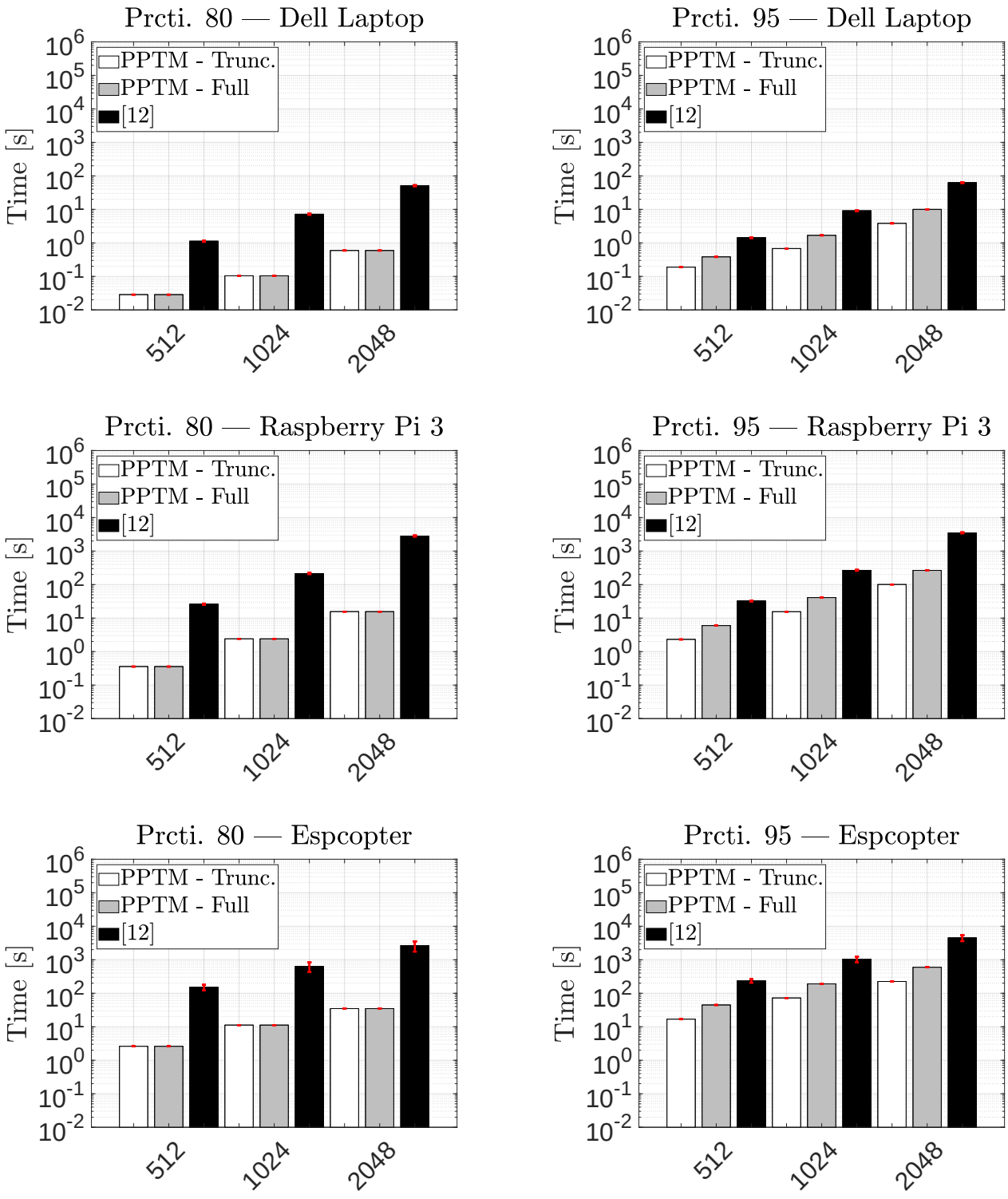

Figure 5: Percentiles 80 (left) and 95 (right) of the time required by *PPTM* (*Truncated Mode* and *Full Mode*) and the solution in [25] on the Dell (up), Raspberry Pi (mid) and Espcopter (bottom).

future autonomous UAVs. In future, we will test our solution with additional mobility models and real mobility traces.

ACKNOWLEDGMENTS

This work has been supported by the INTERSECT project, Grant No. NWA.1162.18.301, funded by Netherlands Organisation for Scientific Research (NWO). The findings reported herein are solely responsibility of the authors.

REFERENCES

[1] AllAboutSSL. 2016. Drones is now getting digital TLS / SSL certificates. https://www.whatissslcertificate.com/drones-is-now-getting-digital-tls-ssl-certificates/. (Accessed: 2022-Sep-28).

[2] ARM. 2021. ARM MbedTLS library. https://github.com/ARMmbed/mbedtls. (Accessed: 2022-Sep-28).

[3] Elaine Barker. 2020. *Recommendation for key management: Part 1 - General.* Technical Report. NIST.

[4] Bruno Blanchet. 2016. Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif. *Foundations and Trends in Privacy and Security* 1, 1–2 (Oct. 2016), 1–135.

[5] Pinar Civicioglu. 2012. Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm. *Computers & Geosciences* 46 (2012), 229–247.

[6] D. George, S. Sciancalepore. 2022. Open Source Code of the Implementation of PPTM into ProVerif. https://github.com/DominikRoy/PPTM. (Accessed: 2022-Sep-28).

[7] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2012. Fast and private computation of cardinality of set intersection and union. In *International Conference on Cryptology and Network Security*. Springer, 218–231.

[8] Motahareh Dehghan and Babak Sadeghiyan. 2019. Privacy-preserving collision detection of moving objects. *Transactions on Emerging Telecommunications Technologies* 30, 3 (2019), e3484. https://doi.org/10.1002/ett.3484

[9] DJI. 2019. DJI Demonstrates Direct Drone-To-Phone Remote Identification. https://tinyurl.com/y39pft7x. (Accessed: 2022-Sep-28).

[10] ESPcopter. 2021. ESPcopter: Programmable MiniDrone. https://espcopter.com/. (Accessed: 2022-Sep-28).

[11] Keith B Frikken and Mikhail J Atallah. 2004. Privacy Preserving Route Planning. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. 8–15.

[12] Joseph Funke, Matthew Brown, Stephen M. Erlien, and J. Christian Gerdes. 2017. Collision Avoidance and Stabilization for Autonomous Vehicles in Emergency Scenarios. *IEEE Transactions on Control Systems Technology* 25, 4 (2017), 1204–1216. https://doi.org/10.1109/TCST.2016.2599783

[13] Globenswire. 2021. Autonomous Vehicle Market Size to Reach USD 888.40 Billion by 2028. https://www.globenewswire.com/news-release/2021/02/01/2167017/0/en/Autonomous-Vehicle-Market-Size-to-Reach-USD-888-40-Billion-by-2028-Emergence-of-Advanced-Driver-Assistance-System-ADAS-will-be-the-Key-Factor-Driving-the-Industry-Growth-States-Eme.html. (Accessed: 2022-Sep-28).

[14] Inside Unmanned Systems. 2021. Amsterdam's Canals Float Fleet of Autonomous Boats for Transport, Waste Collection. https://insideunmannedsystems.com/amsterdams-canals-float-fleet-of-autonomous-boats-for-passenger-transport-waste-collection-environmental-sensing/. (Accessed: 2022-Sep-28).

[15] Panayiotis Kotzanikolaou, Constantinos Patsakis, Emmanouil Magkos, and Michalis Korakakis. 2016. Lightweight private proximity testing for geospatial social networks. *Computer Communications* 73 (2016), 263–270. Online Social Networks.

[16] Xiuguang Li, Yuanyuan He, Ben Niu, Kai Yang, and Hui Li. 2016. An Exact and Efficient PrivacyPreserving Spatiotemporal Matching in Mobile Social Networks. *International Journal of Technology and Human Interaction (IJTHI)* 12, 2 (2016), 36–47.

[17] Stefano Mariani, Giacomo Cabri, and Franco Zambonelli. 2021. Coordination of Autonomous Vehicles: Taxonomy and Survey. *ACM Comput. Surv.* 54, 1, Article 19 (Feb. 2021), 33 pages. https://doi.org/10.1145/3431231

[18] Ben Nassi, Asaf Shabtai, Ryusuke Masuoka, and Yuval Elovici. 2019. SoK - Security and Privacy in the Age of Drones: Threats, Challenges, Solution Mechanisms, and Scientific Gaps. *arXiv preprint arXiv:1903.05155* (2019).

[19] Lewis Nkenyereye, Chi Harold Liu, and JaeSeung Song. 2019. Towards secure and privacy preserving collision avoidance system in 5G fog based Internet of Vehicles. *Future Generation Computer Systems* 95 (2019), 488–499. https://doi.org/10.1016/j.future.2018.12.031

[20] Lionel Pournin and Thomas M Liebling. 2009. From Spheres to Spheropolyhedra: Generalized Distinct Element Methodology and Algorithm Analysis. In *Research trends in combinatorial optimization*. Springer, 347–363.

[21] Amir Rasouli and John K. Tsotsos. 2020. Autonomous Vehicles That Interact With Pedestrians: A Survey of Theory and Practice. *IEEE Transactions on Intelligent Transportation Systems* 21, 3 (2020), 900–918. https://doi.org/10.1109/TITS.2019.2901817

[22] Raspberry PI. 2021. Raspberry Pi 3 Model B+. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. (Accessed: 2022-Sep-28).

[23] Savio Sciancalepore. 2022. PARFAIT: Privacy-preserving, secure, and low-delay service access in fog-enabled IoT ecosystems. *Computer Networks* 206 (2022), 108799.

[24] Reza Shokri, George Theodorakopoulos, and Carmela Troncoso. 2016. Privacy games along location traces: A game-theoretic framework for optimizing location privacy. *ACM Transactions on Privacy and Security (TOPS)* 19, 4 (2016), 1–31.

[25] Jingchao Sun, Rui Zhang, and Yanchao Zhang. 2013. Privacy-preserving spatiotemporal matching. In *Proceedings IEEE INFOCOM*. 800–808. https://doi.org/10.1109/INFCOM.2013.6566867

[26] Jingchao Sun, Rui Zhang, and Yanchao Zhang. 2016. Privacy-Preserving Spatiotemporal Matching for Secure Device-to-Device Communications. *IEEE Internet of Things Journal* 3, 6 (2016), 1048–1060. https://doi.org/10.1109/JIOT.2016.2549046

[27] Alisson R. Svaigen, Azzedine Boukerche, Linnyer B. Ruiz, and Antonio A. F. Loureiro. 2021. MixDrones: A Mix Zones-Based Location Privacy Protection Mechanism for the Internet of Drones. In *Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (Alicante, Spain) *(MSWiM '21)*. 181–188.

[28] Pietro Tedeschi, Savio Sciancalepore, and Roberto Di Pietro. 2021. ARID: Anonymous Remote IDentification of Unmanned Aerial Vehicles. In *Annual Computer Security Applications Conference (ACSAC)*. Association for Computing Machinery, 207–218.

[29] Pietro Tedeschi, Savio Sciancalepore, and Roberto Di Pietro. 2022. PPCA - Privacy-Preserving Collision Avoidance for Autonomous Unmanned Aerial Vehicles. *IEEE Transactions on Dependable and Secure Computing* (2022), 1–1. https://doi.org/10.1109/TDSC.2022.3159837

[30] The Economist. 2021. Business is booming as regulators relax drone laws. https://www.economist.com/science-and-technology/2021/06/17/business-is-booming-as-regulators-relax-drone-laws. (Accessed: 2022-Sep-28).

[31] George Theodorakopoulos, Reza Shokri, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. 2014. Prolonging the hide-and-seek game: Optimal trajectory privacy for location-based services. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. 73–82.

[32] Marc Van Kreveld, Otfried Schwarzkopf, Mark de Berg, and Mark Overmars. 2000. *Computational Geometry Algorithms and Applications*. Springer.

[33] Jiadai Wang, Jiajia Liu, and Nei Kato. 2019. Networking and Communications in Autonomous Driving: A Survey. *IEEE Communications Surveys Tutorials* 21, 2 (2019), 1243–1274. https://doi.org/10.1109/COMST.2018.2888904

[34] Xu Wei-jiang, Jing Wei-wei, Huang Liu-sheng, and Yao Yi-fei. 2007. Privacy-Preserving Collision Detection of Two Circles. In *Proceedings of the 2nd International Conference on Scalable Information Systems* (Suzhou, China) *(InfoScale '07)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Article 73, 7 pages.

[35] Jawad N. Yasin, Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. 2020. Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches. *IEEE Access* 8 (2020), 105139–105155. https://doi.org/10.1109/ACCESS.2020.3000064

[36] Ayong Ye, Yacheng Li, Li Xu, Qing Li, and Hui Lin. 2017. A Trajectory Privacy-Preserving Algorithm Based on Road Networks in Continuous Location-Based Services. In *2017 IEEE Trustcom/BigDataSE/ICESS*. 510–516. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.278

APPENDIX A: NOTATION

Table 3: Notation used throughout the paper.

Notation	Description
ϵ	Adversary.
M, N	Number of entries in the path of UAVs $\mathcal{A}$ and $\mathcal{B}$.
$\mathbf{V_{j,m}}$	Speed of the UAV j in the time-frame $[t_m t_{m+1}]$.
$t_{j,m}$	Timestamp of the location m occupied by the UAV j.
$\mathbf{T_j}$	Trajectory of the j-th UAV.
r_i	Radius of the ref. capsule at step i of *PPTM*.
$u_{j,1}, v_{j,1}$	Coefficients of the equation of the line merging the first and last points of T_j.
$d_{m,\tilde{y}_{j,1}}$	Distance of the m-th point in T_j from the ref. line equation $\tilde{y}_j$.
$dm_{j,1}$	Maximum value between all $d_{m,\tilde{y}_{j,1}}$.
δ	Collision threshold.
σ	Maximum GPS inaccuracy.
h_i	Length of the ref. capsule at step i of *PPTM*.
$d_{j,1}$	Euclidean distance between first and last points in T_j.
Θ_i	Angle of the ref. capsule at step i of *PPTM*.
$\mathbf{s_i}$	Nonces extracted at the step i of *PPTM*.
$\mathbf{O_i}$	Origin points used at the step i of *PPTM*.
$\mathbf{x'_j}, \mathbf{y'_j}$	x-y coordinates of roto-translated points of T_j.
$\tilde{\mathbf{x}}_\mathbf{j}, \tilde{\mathbf{y}}_\mathbf{j}$	Position identifiers of T_j.
f, g	Functions mapping positions and times to unique ids.
$H(\cdot)$	Generic hashing function.
p_j, q_j	Safe primes of the UAV j.
n_j	Size of the modular field of the UAV j.
$\xi_{j,i}$	Nonce of UAV j at round i of *PPTM*.
$\mathbf{c_{j,i}}$	Vector of the encrypted challenges generated by j at the round i of *PPTM*.
$\mathbf{\Gamma_{j,i}}$	Vector of position identifiers on j at the round i of *PPTM*, containing K elements.
$\tilde{\mathbf{c}}_\mathbf{j,i}$	Vector of the encrypted responses generated by j at the round i of *PPTM*.
$\mathbf{w_{j,i}}$	Vector of *check values* on j at the step i of *PPTM*, containing K elements.
$\tilde{\mathbf{w}}_\mathbf{j,i}$	Encrypted proof of proximity computed by j at the step i of *PPTM*.
$\mathbf{t_j}$	Vector of timestamps of colliding points on j.
$\tilde{t_j}$	Time-frame between the first and last timestamp in $\mathbf{t_j}$.
t_O	Origin Timestamp of the *Time Trajectory Match*.
τ_j	Timestamp id for the *Time Trajectory Match* sub-phase.
$\tilde{\tau_j}$	Timestamp id obtained by applying g to τ_j.