Trajectory-based Authenticated Key Establishment for Dynamic Internet of Things

DOI:
10.1109/ACCESS.2022.3215688

Document Version
Final published version

Link to publication record in Manchester Research Explorer

Citation for published version (APA):

Published in:
IEEE Access

Citing this paper
Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights
Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy
If you believe that this document breaches copyright please refer to the University of Manchester’s Takedown Procedures [http://man.ac.uk/04Y6Bo] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.
Trajectory-based Authenticated Key Establishment for Dynamic Internet of Things

MENGYA LI1, and NING ZHANG1

1Department of Computer Science, The University of Manchester, Manchester, M13 9PL, UK
Corresponding author: Mengya Li (e-mail: mengya.li@manchester.ac.uk).

This work is supported by the University of Manchester under President’s Doctoral Scholar Award.

ABSTRACT In Internet of Things (IoT), cryptography-based security services are widely used to mitigate security threats. However, establishing cryptographic keys between entities in dynamic IoTs is a challenging issue, due to the high mobility of these entities. They may not have established secrets prior to the key establishment, and the key establishment has to be finished in a limited time. Existing solutions either rely on the entities having prior secrets, or have a latency that may not meet the time limit. This paper proposes a new key establishment method that does not use prior secret or trust and has high efficiency. The method, called Trajectory-based Authenticated Key Establishment (TAKE), establishes a shared secret and uses it to derive a pair of symmetric keys, using only real-time trajectory data of a moving entity. The shared secret can also be used for authentication or to establish asymmetric keys. Theoretical analysis and experimental evaluation demonstrate that TAKE can efficiently establish keys with multiple security levels and resist attacks on authentication, confidentiality and integrity. Experiment results show TAKE can distinguish authorised entities and establish keys for them with high reliability, with a true positive rate of 99% and a false positive rate of 0. One key establishment finishes within 200 ms, which is significantly faster than other solutions without using prior secrets.

INDEX TERMS Cryptography, error tolerance, Internet of Things, key establishment

I. INTRODUCTION

Internet of Things (IoT) is a type of network that connects smart devices and provides wide access to services from anywhere at any time [1] [2]. Dynamic IoTs refer to the IoTs where entities can move around and access services without disruption. An example is Internet of Vehicles (IoV) for smart transportation [3]. IoV creates a safer and more relaxed travel experience for vehicle users, by providing driving assistance and automation services and giving access to any third-party services on the Internet. The services are supported by communication between users and their peers, and between users and their remote service providers. As users travel, their peers change, and their local access points to the remote service providers also change. Users need to dynamically form wireless communication channels with their current peers and local access points to continuously access services. The communication channels are vulnerable to various threats, which may compromise the security of service data and cause the services to malfunction [1] [2] [3]. Measures should be taken to mitigate the threats and ensure data security.

One way to ensure data security is using cryptography-based security services [1] [2] [3] [4]. However, these services require the communicating entities to have established cryptographic keys. Such keys should only be known to the authorised entities. To this end, the keys should only be established after the communicating entities have authenticated each other. The keys also need to remain secret and authentic throughout their lifetimes. Such keys can be established with authenticated key establishment (AKE) methods.

Existing AKE methods use one or more secrets to establish secure keys between authorised entities. The secrets are necessary for authenticating the entities, and for protecting the new keys in distribution using public communication channels. Based on the secrets they use, the existing methods can largely be classified into two categories: AKE using prior secrets, and AKE using temporary secrets. Prior secrets are secrets established before AKE, such as pre-shared secrets.
between the authorised entities or private keys with associated verifiable public keys. Temporary secrets are secrets extracted at the time of key establishment from data sources that only the authorised entities can access. Existing AKE methods using these secrets face challenges in dynamic IoTs for the following three reasons. First, entities have limited time to establishing a key for a dynamically formed channel [3] [8]. Second, keys may need to be established for entities that meet for the first time and do not have any prior secrets. Third, an entity may connect to different access points and/or use different services at the same time. It may need to establish keys with different security levels to suit the context. Existing methods using public keys are computationally expensive, and may not be able to be used by resource-restrained entities or finish key establishment in the limited time [9] [10] [11].

The methods using temporary secrets also have a latency of several seconds [12] [13] [14], which may not meet the time requirement. Besides, they are mostly designed for entities in a close physical range and may not be applicable to remote key establishment. The methods using pre-shared secrets are more efficient, but are not feasible when entities have no prior secrets. The existing methods are also mostly not designed to support variable security levels.

In this paper, we propose a new AKE method, called Trajectory-based Authenticated Key Establishment (TAKE). For proof of concept, TAKE is designed in the context of IoV, to establish keys for vehicles and remote service providers. TAKE uses temporary secrets for authenticating the entities and protecting confidentiality and integrity of the keys. The temporary secrets are extracted from the real-time trajectory of vehicles. This is to improve efficiency, as the trajectory data are collected by each of the authorised entities with low costs and latency. However, sampling errors may happen during data collection, thus the data collected by different entities may not be identical. To establish identical secrets with the non-identical data, error correction and tolerance measures are designed to allow for sampling errors while minimising the threats of attacks on the secrets. Furthermore, TAKE lets the entities choose a security level for each key establishment. This achieves a trade-off between efficiency and security, as keys with higher security level need higher costs to establish and use. With variable security levels, for each service session, a key can be established to provide the desired level of security with minimum costs.

Compared to existing work, TAKE has the following merits. First, TAKE can establish a shared secret over a public channel for entities that do not have any prior secrets. TAKE only requires the entities to be able to collect real-time trajectory data with high quality, which is a common requirement in dynamic IoTs such as IoV and Internet of Drones (IoD) [15]. Second, TAKE achieves high efficiency, which makes it suitable for applications with strict timeliness requirements. TAKE has low communication costs, as only one message needs to be sent to establish one key. TAKE also has low computational costs, as it mostly uses computationally cheap operations, such as hash function and exclusive-

or. Third, TAKE provides a new way to use trajectory as a source of secrets. Trajectory has been used in another security service, entity authentication [16] [17] [18] [19]. It is used to verify that an identity claimed by an entity is true, but it has not been used to extract or establish secrets. We have conducted a quantitative experimental evaluation on the security of secrets extracted from real-time trajectory data using TAKE. The results show that the extracted secrets have enough pseudorandom bits to meet the security requirements, i.e., trajectory can be used as a source of secrets.

The rest of the paper is organised as follows. Section II gives an overview of the related work on AKE. Section III describes a use case of key establishment in the context of IoV, and gives security threats, requirements and assumptions in this use case. Section IV introduces the high-level ideas used to design TAKE. Section V gives the design primitives. Section VI describes the design of TAKE and Section VII gives TAKE in detail. Section VIII and Section IX respectively give the theoretical and experimental evaluation of TAKE. Section X gives the conclusions and future work.

II. RELATED WORK

A. AKE USING PRIOR SECRETS

Existing AKE methods using prior secrets are mostly based on two cryptographic techniques: Diffie-Hellman key exchange (DHKE), and key derivation.

AKE methods adopt the DHKE scheme as it brings three advantages. First, DHKE scheme can establish fresh keys that are only known to key establishing entities using only public communication [20]. Second, it achieves forward/backward secrecy as the new keys are independent from old keys. Third, it prevents the values of established keys from being controlled by one entity, thus achieving fairness. However, the basic DHKE scheme does not ensure the integrity of the established keys or the authenticity of the key establishing entities. It also has high computational costs due to the asymmetric cryptography it uses.

To provide integrity protection and entity authentication, researchers have proposed to use additional prior secrets in DHKE-based schemes. For example, in [9], entities register their public keys in advance. During an AKE instance, key materials exchanged between the entities are signed using the private keys corresponding to the registered public keys, so that the entities can verify the origin and integrity of the key materials. In [10], to establish keys between a user device and drones, user password and biometrics are registered in advance at a central authority. In registration, the user device stores credentials that bind the password and biometrics with the private secret of the authority. During an AKE instance, the credentials are used to generate a checksum on the key materials, and the checksum and key materials are sent together to the authority to verify the origin and integrity of the key materials. The use of additional secrets, e.g., generating signatures and checksums, adds more costs on top of the costs of using asymmetric cryptography to exchange key materials with DHKE. To improve efficiency, AKE methods...
have used more efficient asymmetric cryptography, such as elliptic curve cryptography \cite{9} \cite{10} \cite{21} and chaotic maps \cite{22} \cite{23}.

AKE solutions based on key derivation improve efficiency further by avoiding asymmetric cryptography \cite{24} \cite{25} \cite{26}. In these methods, each entity uses a key derivation function (KDF) to generate a key from the same inputs. As KDFs are typically based on cryptographic hash functions, they are computationally cheaper than DHKE. To establish keys, prior secrets and optionally some temporary shared values are input to the KDFs. The prior secrets are used to authenticate the entities, as only the entities which knows the prior secrets can derive correct keys. The temporary shared values are values that are known to the key establishing entities and are only valid for one AKE instance. They are used to provide freshness. These values can be sequence numbers, timestamps, and/or random numbers freshly generated in AKE instances \cite{24} \cite{25} \cite{26}. However, if temporary shared values are related to previous instances, e.g. sequence numbers \cite{25}, the methods may be vulnerable to desynchronisation attacks. Attackers may cause the temporary shared value to fall out of synchronisation so that authentic entities cannot derive the same keys. In addition, it may be more difficult for such methods to achieve forward secrecy, as new key values are not independent from old key values.

The need of prior secrets limits the usability of the AKE solutions above. First, these solutions require entities to be able to store private keys or shared secrets and keep them confidential through their lifetimes. To achieve this, they have used temper proof storage to store keys and secrets \cite{25} and dedicated hardware, such as smart cards \cite{22} or terminal devices \cite{10} \cite{24}, to support the use of certain types of prior secrets, such as biometrics and passwords. Using temper proof storage or dedicated hardware increases the costs of these methods, so they may not be suitable for resource-constrained application scenarios. Also, using biometrics and passwords requires assistance from human users. This may not be feasible in scenarios without human presence, such as machine-to-machine communication. Second, these solutions may not be feasible when the entities to establish keys do not have prior secrets. When mobile entities have never interacted before, they might not have a pre-shared secret and cannot use the key derivation based AKE methods. In this case, public-key-based methods may have to be used, which causes higher computational costs and higher latency.

B. AKE USING TEMPORARY SECRETS

AKE solutions can establish temporary secrets using data sampled by the entities from data source(s).

Temporary secrets can be used as additional factors to prior secrets to enhance security of AKE schemes. For example, to establish keys between vehicles and their local access points \cite{27}, location is used in addition to public keys. The entities send their own location to each other in AKE messages protected by public key encryption and signature. Then they calculate key values by solving equations using the shared location data as the equation parameters. In \cite{28}, to establish keys between entities using the same communication channel, phase and magnitude parameters of the wireless signal of the channel are used as temporary secrets in addition to pre-shared passwords. The signal parameters can enhance the security of the scheme because only the two entities using the communication channel can sample signal parameters accurately. The parameter values remain secret to any other entities.

Temporary secrets can also be used as the sole secret to provide authentication, confidentiality and integrity protection in an AKE scheme. In this case, as no prior secret is used, entities cannot share data samples in secret by using encryption, and have to use their independently sampled data of the same data source(s). However, the entities may not have identical samples due to sampling errors. The samples must be corrected and become identical, so that the entities can agree on identical keys. A widely used method to correct samples and establish keys is a cryptographic technique called fuzzy extractor \cite{29}. Existing solutions have used samples of wireless signal strength \cite{12}, sound \cite{30} \cite{13} and movement data \cite{31} collected by devices in close range, to establish keys for these devices using fuzzy extractors. However, these solutions may have a high latency. This is because for fuzzy extractor to generate secure keys, the input samples must contain a certain amount of randomness so that it is hard to predict. To provide such randomness, entities need to collect samples for a certain period of time after an AKE instance starts. Methods have been proposed to reduce the sample collection time, such as using more data sources and improving sample processing methods to extract randomness more effectively \cite{13} \cite{31}. But these solutions can still have a latency of several seconds, which might not meet the timeliness requirements in dynamic IoTs. In addition, existing solutions may be challenged when symmetric keys are to be established exclusively for two entities. This is because all devices in a close range to the authorised entities may be able to sample the same data source(s) and have close data samples. These devices may all be able to establish the same keys using their own samples. Furthermore, existing solutions can be improved to support key establishment for remote entities. The entities which are not physically in a close range may not be able to collect samples of a shared data source located in the physical environment.

III. A USE CASE

We present a use case called V2IKE, as an example of AKE in dynamic IoTs. We first build a use case model, to describe entities and interactions in the use case. Next, we analyse the security threats and specify the design requirements. Then we give the assumptions used in the design.

A. USE CASE MODEL

V2IKE use case model is shown in Figure \[1\]. It consists of four types of entities.
1) Vehicle: Vehicles carry human users to their destinations and act as an interface for users to use IoV services. In the use case model, vehicles are equivalent to IoV users. Vehicles are equipped with an on-board unit (OBU) for communication and computation, and on-board sensors to collect real-time data. OBU can use sensor data and information obtained through communication to support IoV services that can be performed on board, such as motion control and collision avoidance [3] [32].

2) Traffic control centre (TCC): TCC is a centralised authority and service provider in the IoV. It provides driving assistance and automation services which regulate traffic in the global range, such as route following and route planning [3] [33]. It is also responsible for providing Internet access for on-board users. TCC provides services to vehicles through vehicle-to-infrastructure (V2I) communication.

3) Road side unit (RSU): RSUs directly communicate with vehicles (V2R communication). They are connected to TCC through infrastructure communication channels. They are part of V2I communication as they relay messages from vehicles to remote recipients. RSUs may also provide local range IoV services, such as traffic information sharing and local cluster communication [3] [33].

4) Road side sensors (RSS): RSSs are sensors that are installed at the road side to collect information needed for IoV services. They may collect weather data, traffic data, etc. RSU, RSS and TCC are considered as IoV Infrastructure entities.

![Figure 1: V2IKE use case model.](image)

In V2IKE, keys are to be established between a vehicle and TCC. It is chosen as the example to create a proof of concept, because it captures the typical characteristics of AKE in dynamic IoTs. First, keys are established frequently. To access services from TCC, a vehicle needs to be connected to an RSU. An RSU had a limited coverage range. When the vehicle leaves the range of its current RSU and moves into the range of the next one, it starts a new service session with the new RSU. To secure the new session, a new key needs to be established. As the vehicle travels, it frequently starts new service sessions and establishes new keys. Second, the keys has to be established as quickly as possible. This is to minimise disruption to service, as services provided by TCC are critical to road safety and should be available at all times. Third, the key establishment is between two entities in remote locations, and uses wireless public communication. Fourth, the desired security levels of the established keys can be dynamically changing. A vehicle can use different services, and moves in different local networks which may face different levels of security risks. The vehicle can choose to use a key with a higher security level for a more sensitive service or in a more risky local network. The V2IKE use case may not have a compelling reason for not using prior secrets, as TCC and vehicles typically would have established long-term secrets when the vehicles were registered. However, it can still benefit from using an AKE method that does not require prior secrets. For example, it can help preserve privacy. If vehicles do not need to use prior secrets such as long-term private/public keys, there are lower risks of unauthorised vehicle tracking by monitoring public keys in public communication and linking identities [34], [35].

B. THREAT ANALYSIS

The following attacks may threat the security in the V2IKE use case.

1) Modification attack

Modification attacks target the integrity of the keys. The attacks modify the data exchanged between TCC and vehicles. This may cause the AKE to fail and disrupt the services as the entities may not be able to agree on the same keys with the modified data. Moreover, the modified data may lead the entities to establish keys the values of which may be controlled to some extent by the attackers. This may make it easier for the attackers to reveal the keys.

2) Impersonation attack

Impersonation attacks are a threat to entity authentication, where attackers impersonate an authentic entity to other entities. In V2IKE, attackers can impersonate TCC to the vehicle, impersonate the vehicle to TCC, or to impersonate both entities to each other. By impersonating the authentic entities, attackers may gain authorised access to services, obtain confidential or sensitive data, and inject false data into communication and cause malfunction.

3) Guessing attacks

Guessing attacks aim to discover the value of a secret. It is a threat to the confidentiality of the temporary secrets and the keys. The attacks can be done in two ways: brute-force guessing and cryptoanalysis. In brute-force guessing attacks, attackers may try all possible values in guesses and test if a guess is correct, until they find the correct value. In cryptoanalysis, they attempt to discover a secret value by analysing known information related to the secret. The attackers may use...
such information and their cryptography knowledge to learn information about the secret, and then use this information to direct their guesses and improve the chances of finding the secret.

4) Denial-of-service (DoS) attacks
DoS attacks are those that try to stop authentic entities from successfully establishing keys, by interrupting a key establishment process. DoS attacks are a threat to availability. Attackers may try to interfere with the communication channel to prevent the entities from receiving messages needed for AKE, or exhaust the resources such as computational power and storage of the entities so that they fail to finish the AKE process in time.

C. ADVERSARY MODEL
The design of TAKE focuses on external threats, i.e. threats from outsiders. An outsider refers to any entity which is not one of the pair of key establishing entities. It is assumed that outsiders may try to compromise the integrity and confidentiality of the keys to be established and they may try to impersonate authorised entities. Threats on availability, such as DoS attacks, are outside the scope of this paper. In detail, our adversary (i.e. threat) model is defined as follows.
1) Vehicles are dishonest. They may eavesdrop on communications between other entities. They may mount modification attacks, intercepting and then modifying or replaying a message destined to its legitimate recipient. They may mount impersonation attacks, impersonating another vehicle or an IoV infrastructure entity by stealing or guessing their credentials, or other useful data that may allow them to obtain the credentials.
2) RSUs are trustworthy. They will faithfully perform their functions. Especially, they will not modify, replay or drop messages they should relay between vehicles and TCC.
3) TCC, as the centralised authority, is trustworthy. It will faithfully perform its functions.

D. REQUIREMENT SPECIFICATIONS
Based on the use case model and the threat analysis above, we specify a set of requirements for an effective and efficient AKE solution.

1) Functional requirements
(PR1) Dynamic key establishment: TCC and a vehicle, regarded as the authorised entities, should be able to establish a pair of symmetric keys. The key establishment should not use any prior secrets, and should use only public communication.

2) Security requirements
(SR1) Mutual authentication: keys should only be established between the authorised entities which have been mutually authenticated.
(SR2) Key Confidentiality: the keys should remain confidential under guessing attacks, as required by the authorised entities. Key confidentiality is considered as the security level of a key.
(SR3) Data integrity: the AKE design should resist modification attacks. Modified data should be detected by the authorised entities.
(SR4) Forward/backward secrecy: if keys generated in some AKE instances are revealed, the confidentiality of the keys in other instances should not be affected.

3) Performance requirements
(PR1) High efficiency: the costs caused by using the AKE solution should be minimised on both authorised entities.

E. ASSUMPTIONS
The following assumptions are used in the design of TAKE.
(A1) Assume authorised entities have the same trajectory data collection system parameter values. The parameters are resolution, sampling frequency, and maximum localisation error in Euclidean metric. The parameter values meet the data quality requirements of IoV services.
(A2) Assume authorised entities have synchronised clocks and have agreed on at which timestamps they should sample real-time trajectory data and movement data.
(A3) Assume authorised entities record location data in the decimal degree format of the WGS 84 reference system [36].

IV. HIGH-LEVEL IDEAS
The design of TAKE uses three high-level ideas: choosing a suitable source of secrets, temporary secret extraction, and key establishment with the extracted secret.

A. DATA SOURCE FOR EXTRACTING SECRETS
To meet the functional and the security requirements, we need a way to achieve mutual authentication and secure data exchange using public communication without using prior secrets. This can be done by using temporary secrets, if we can to find a suitable data source for extracting temporary secrets in the V2IKE use case. Such a data source should have the following features. First, it should be a reliable source of secrets (SR1, SR2, SR3). The data sampled from the source should remain secret to any unauthorised entities. That is, in V2IKE, only the vehicle and TCC should be able to capture data samples from the source with high quality, i.e., with high accuracy and precision. In addition, the samples from different sources should be unique in value, and each vehicle should have a unique source, so that the vehicles can be reliably distinguished from each other using the samples. Second, sampling the data source and storing the data samples should add minimum costs to the vehicle and TCC (PR1). This means, preferably, data samples from this source should have already been captured by vehicles and TCC to support IoV services, so that using these samples for AKE would not add much cost.

We analyse the data collection in IoV and find that real-time trajectory has these features, so TAKE chooses it as the source of secrets. The features of real-time trajectory and the...
reasons for choosing it are described in more detail in Section VI-A.

B. TEMPORARY SECRET EXTRACTION

To extract a temporary secret, the vehicle and TCC needs to each generate a copy of the same secret from the trajectory data samples collected by themselves. This can be done by using a challenge-response mechanism. In each TAKE instance, a challenge is generated, to instruct which data to choose from all the data samples and how to compose the secret. Each entity then follows the instruction and generates a response as its temporary secret. If they have high-quality data samples, the sampling errors in them will be limited. They will have similar temporary secrets, which are regarded as close copies of the same secret.

Using the challenge-response mechanism brings the following advantages. First, a fresh challenge can be used for each key establishment. This helps resist impersonation attacks using replayed messages (SR1). The fresh challenge also results in a fresh temporary secret. With a randomised challenge, the temporary secrets can be different in each instance, which helps achieve forward/backward secrecy (SR4). Second, the challenge can be adjusted to extract secrets of different sizes, which can be used to adjust the security level of the established keys as required (SR2).

C. KEY ESTABLISHMENT

Keys are established by exchanging a fresh secret random value, i.e., a base secret, between the entities. The base secret is then used to derive a pair of symmetric keys using key derivation. By using a fresh random value, keys of different instances (SR4) are independent, and discovering the key of one instance does not affect the confidentiality of keys in other instances (SR4).

The base secret is sent using public communication, which is under threat from confidentiality and integrity attacks. TAKE uses the temporary secrets to protect the base secret in transit (SR2, SR3). However, the temporary secrets extracted by the vehicle and TCC may be similar but different. For them to share the base secret, the temporary secrets must be made identical. Thus, TAKE should be able to tolerate sampling errors in temporary secrets and make the secrets identical. At the same time, the error tolerance capability may make it easier for attackers to succeed in guessing attacks. This is because, instead of finding the exact value of the temporary secret, they can now succeed by finding one of all the values within the error tolerance range to the exact secret. TAKE uses both error correction and error tolerance techniques, to make the close temporary secrets identical for the authorised entities, while rejecting guesses from unauthorised entities. We describe how keys are established with error tolerance in Section VI-C.

V. DESIGN PRIMITIVES

A. UNIVERSAL HASH FUNCTIONS

Universal hash functions are a family of hash functions that can transform non-uniform inputs into uniformly distributed hash values [37]. The definition is given as follows.

Definition 1 (Universal Hash Functions). Let $H$ be a set of hash functions $h: \{0, 1\}^m \rightarrow \{0, 1\}^l$. $H$ is called a universal family of hash functions if for any $x_1 \neq x_2 \in \{0, 1\}^m$, there exists

$$Pr_{h \in H}[h(x_1) = h(x_2)] \leq 2^{-l}$$  \hspace{1cm} (1)

The universal hash functions can be constructed as

$$h_{a,b}(x) = ((ax + b) \mod p) \mod n$$  \hspace{1cm} (2)

where $n = 2^l$, $p \geq 2^m$ is a randomly chosen prime, $a, b$ are randomly chosen integers modulo $p$ with $a \neq 0$.

B. FUZZY EXTRACTOR

1) Definitions

A fuzzy extractor establishes a fresh identical secret between two entities which have nonidentical but close copies of a shared secret. The formal definition of fuzzy extractor is given as follows [29], [35], [39].

Definition 2 (Fuzzy Extractor). Let $\mathcal{M}$ be a metric space with a distance function, $\text{dis}(-)$. A $(\mathcal{M}, m, l, t, \epsilon)$-fuzzy extractor is a pair of randomised functions, generation (Gen) and reproduction (Rep), where,

- $\text{Gen}$ takes an element $w \in \mathcal{M}$ as input, and outputs an extracted string $S \in \{0, 1\}^l$ and a helper string $P \in \{0, 1\}^*$, and
- $\text{Rep}$ takes an element $w' \in \mathcal{M}$ and a bit string $P \in \{0, 1\}^*$ as inputs, and outputs string $S \in \{0, 1\}^l$.

A fuzzy extractor uses $\text{Min-entropy}$ to measure the randomness of its input, $w$. $\text{Min-entropy}$ of a random variable, $A$, is defined as follows.

$$H_\infty(A) = -\log(\max_{a \in A} \Pr(A = a)).$$  \hspace{1cm} (3)

A fuzzy extractor has two properties: the correctness property and the security property. The correctness property means the fuzzy extractor should be able to correct the errors between copies of the shared secret that are close enough, so that they become identical. The security property means the fuzzy extractor should output keys that can resist guessing attacks, even if the helper string $P$ is made public. The formal definitions of the two properties are given as follows.

Definition 3 (Correctness property). If $\text{dis}(w, w') \leq t$ and $S$ and $P$ were generated by $(S, P) \leftarrow \text{Gen}(w)$, then $\text{Rep}(w', P) = S$. If $\text{dis}(w, w') > t$, then no guarantee is provided about the output of Rep.

Definition 4 (Security property). For any distribution $W$ on $\mathcal{M}$ of min-entropy $m$, the string $S$ is nearly uniform even for those who observe $P$. That is, if $(S, P) \leftarrow \text{Gen}(w)$, then $SD((S, P), (U_1, P)) \leq \epsilon$, where $U_1$ is a uniformly distributed random variable over $\{0, 1\}^l$, and SD is a function.
that measures statistical distance of two distributions. The statistical distance of two distributions, $A$ and $B$, over a common discrete domain $V$, is defined as

$$SD(A, B) = \frac{1}{2} \sum_{v \in V} |Pr(A = v) - Pr(B = v)|. \quad (4)$$

The correctness property is achieved with a technique called secure sketch $[38]$.

**Definition 5 (Secure Sketch).** A secure sketch is defined as a pair of functions $(Fsk, Cor)$, where

- $Fsk$ is a typically randomised sketch function, which takes an input $w \in M$ and outputs a sketch $ss \in \{0, 1\}^*$, such that for all random variable $W$ over $M$ with min-entropy $H_\infty(W) \geq m$, the average min-entropy of $W$ given $Fsk(W)$ satisfies $H_\infty(W|Fsk(W)) \geq m'$;
- $Cor$ is a correction function, which, given a word $w' \in M$ and a sketch $ss$, outputs a word $w'' \in M$ such that for any $ss \leftarrow Fsk(w)$ and $\text{dis}(w, w') \leq t$, $w'' = w$ holds.

The secure sketch measure the randomness of the input $w$ after publishing the sketch $ss$ with average min-entropy. The average min-entropy of a random variable $A$ given $B$ is defined as

$$H_\infty(A|B) = -\log( E_{b \in B} E_{a \in A} \text{Pr}(A = a|B = b)) \quad (5)$$

2) Constructions

A number of constructions of fuzzy extractors and their secure sketches have been proposed. This section gives the constructions used in TAKE. These constructions are a code-offset secure sketch $[29]$, a key-binding fuzzy extractor and a robust secure sketch $[38]$.

The code-offset secure sketch is based on fuzzy commitment $[40]$. Its sketch-correction functions $(Fsk, Cor)$ are constructed as follows.

$$ss = Fsk(w; seed), \quad \text{where} \quad \begin{cases} c = C(seed) \in C \\ ss = c \oplus w, \end{cases} \quad (6)$$

$$w'' = Cor(w', ss) = C(D(ss \oplus w')) \oplus ss, \quad (7)$$

where $C$ and $D$ are the encoding and decoding functions of an error-correcting code, $c$ is a codeword of the code, and $C$ is the codeword space of the code.

The key-binding fuzzy extractor is constructed using the code-offset secure sketch. Its generation and reproduction functions, $(Gen, Rep)$, are constructed as follows.

$$\langle S, P \rangle = Gen(w), \quad \text{where} \quad \begin{cases} S = \{0, 1\}^l = RNG(l) \\ P = ss = Fsk(w), \end{cases} \quad (8)$$

$$S = Rep(w', P), \quad \text{where} \quad \begin{cases} w'' = Cor(w', P) \\ c = ss \oplus w'' \\ S = D(c). \end{cases} \quad (9)$$

where $w, w'$ are a pair of nonidentical but close samples of random variable $W$, $RNG(l)$ is a random number generator that generates a $l$-bit random number. If the error-correcting code in the $(6)$ and $(7)$ is a $(n, k, 2t + 1)$-binary code, the key-binding fuzzy extractor can generate key $S$ with an average min-entropy $H_\infty(S|P) = m + k - n \leq m - 2t$, where $m = H_\infty(W)$, and $k \leq n - (2t + 1) + 1$, following the Singleton bound of the $(n, k, 2t + 1)$-binary code $[38]$.

The robust secure sketch provides integrity protection for the secure sketch $ss$. This is achieved by using a checksum. The checksum is a hash value generated with $ss$ and $w$, using universal hash functions. As a result, the construction is secure without using the random oracle assumption $[39]$. The robust secure sketch is constructed as a family of functions $(Fsk_L, Cor_L)$, as follows.

$$Fsk_L(w) = ss, \text{where} \quad \begin{cases} ss = Fsk(w) \\ checksum = h_L(w, ss^*) \\ ss = \langle ss^*, checksum, L \rangle, \end{cases} \quad (10)$$

$$Cor_L(w', ss) = \begin{cases} w'' = Cor(w', ss^*), \text{if dis}(w', w'') \leq t \\ \text{and } h_L(w'', ss^*) = \text{checksum}, \text{otherwise}. \end{cases} \quad (11)$$

where $h_L \in H$ and $H$ is a family of universal hash functions.

**C. KEY DERIVATION FUNCTION**

A key derivation function (KDF) generates a cryptographic key from a base secret. It can stretch a short secret into longer keys or use a base secret to obtain keys in certain formats $[41]$. For example, KDF is used to derive longer cryptographic keys from a shorter password $[42]$. Some examples of KDF are PBKDF2 $[43]$ and scrypt $[44]$. We denote a KDF as

$$key = KDF(s, l), \quad (12)$$

where $key$ is the derived key, $s$ is the base secret, $l$ is the key length.

**VI. TAKE DESIGN**

**A. REAL-TIME TRAJECTORY AS A SOURCE OF SECRETS**

The trajectory of a vehicle is a series of locations the vehicle has been to at different moments in time during its journey. Trajectory data are the data samples captured of this trajectory. We analyse the collection of real-time trajectory data in the IoV to show it is feasible and cost-efficient to use real-time trajectory as a source for secret. Then we analyse the uniqueness and randomness of the trajectory data to show it is a reliable source of secret.

1) Trajectory Data Collection

Real-time trajectory data are collected to support a range of IoV services. For example, to carry out motion control and collision avoidance, a vehicle needs its current location...
data to compare with the map, to ensure it stays in the lane and does not crash into the side of the road. For the route following service, TCC collects locations of all vehicles on the road, to monitor their movements and give commands to them so that they follow the scheduled route. For the route planning service, TCC uses the real-time trajectory data of vehicles and information of their destinations, to predict their next moves and schedule their routes to avoid traffic jams. To use the trajectory data for AKE, the costs are mainly caused by retrieving these data from storage, which are minimal.

Real-time trajectory data can be captured by vehicles and TCC with high accuracy and precision. A vehicle can capture data of its own trajectory with on-board localisation techniques, such as IMU, GPS/GNSS, SLAM, vision-based localisation, and range-sensor-based localisation with LiDAR and/or radar [45] [46] [47] [48]. TCC can use RSS to collect raw data, and then use localisation and object identification and tracking techniques to obtain trajectory data of any vehicle on the road [49] [50] [51] [52] [53]. In practice, vehicle locations are captured at a high frequency, e.g., 10 Hz, and within a small error range, e.g., within 1 m to the ground truth.

2) Uniqueness and randomness of trajectory data
Uniqueness means a data source provides unique samples for each entity. Randomness means the samples from a data source is hard to predict without accessing the data source. We discuss the uniqueness and randomness of trajectory data, to show that 1) real-time trajectory data of each vehicle are unique, and 2) only the vehicle and TCC are able to have accurate data of the whole trajectory of this vehicle. Uniqueness and randomness depend on two factors: how a trajectory is generated and how trajectory data are collected.

A trajectory is generated by a vehicle while moving. We refer to this trajectory as the True Trajectory of the vehicle, as it is the ground truth. True Trajectory is modelled as a continuous curve which describes the location of the vehicle at each moment in time. We define that the location of a vehicle is the location point at its centre. The True Trajectory is determined by the driving process of the vehicle. We model the driving process with states and transitions between states. A vehicle can be in any one of the following three states:

1) Stable state, where a vehicle is driving at constant speed (the speed is greater than 0).
2) Acceleration state, where a vehicle accelerates from a previous speed to a new speed. Deceleration is considered as an acceleration state with a negative acceleration value.
3) Stop state, where a vehicle stays at a fixed location.

Transitions between states describe the changes between states caused by vehicle actions. For example, a vehicle starting is modelled as a transition from a stop state to an acceleration state. When the vehicle acceleration changes, it can be described as a series of acceleration states, each of which has a different acceleration value. When the vehicle stops accelerating and drives at a constant speed, it goes into a stable state with this speed. When the vehicle is stopping, it goes from the current state into an acceleration state with a negative acceleration value, until the speed decreases to 0 and it goes into a stop state.

To analyse the True Trajectory values, we assume all vehicles obey the following traffic rules at all times:

1) A vehicle drives at a speed between the lower and upper speed limits.
2) A vehicle maintains a stopping distance from the vehicle in front of it, and the stopping distance is decided according to its current speed.
3) A vehicle maintains enough distance between itself and the vehicles in the adjacent lanes.
4) A vehicle obeys traffic lights.

Using the driving process model and the traffic rules, we find that vehicles have unique True Trajectories, if they do not crash into each other. This is because at each moment in time, every vehicle should be at a different location, as they all maintain distances from each other. Unless the vehicles collide, they should each have a different location at the same moment. As a result, True Trajectories are unique in value.

The randomness of a True Trajectory depends on the transitions between states. When a vehicle is driving in the same state, if its movements (i.e., speed, acceleration and heading) are known to attackers, its location at any future moment can be calculated from the current location. This makes the True Trajectory predictable. However, transitions change the states the vehicle is in and the movements in the states. With more transitions, it is harder to know all the movements and calculate the accurate future locations, so the True Trajectory becomes less predictable, i.e., more random. Transitions are caused by driving actions. Driving actions are hard to predict, because they are taken on spot under the influence of many factors, such as road condition, weather, road traffic, and user needs. If a vehicle takes more driving actions, its True Trajectory will be more random, even if it uses the same route.

The trajectory data sampled by an entity are called Raw Samples. A Raw Sample can be represented as a sequence of timestamp-location tuples (hereafter called footprints), where the locations of the vehicle are sampled at certain sampling timestamps, as shown in Figure 2. The uniqueness and randomness of the Raw Samples directly impact the security and performance of TAKE, as they are where temporary secrets are extracted from.

We first prove that Raw Samples of True Trajectories are unique. That is, footprints in Raw Samples of different True Trajectories never have the same values. It is clear that footprints with different timestamps are different. We prove the footprints with the same timestamp (hereafter called Matched Footprints) in Raw Samples of different True Trajectories are unique, i.e., they have different location values. Due to the uniqueness of True Trajectory, the ground truth locations in Matched Footprints of different True Trajectories are different. The distance between the two ground truth
locations is the sum of the distance between the vehicles at the
timestamp and the car length. This distance is typically
at least several meters, as the distance between vehicles are
stopping distance or lane width, which is typically over a
meter, and car length is typically over a meter too. The
location values in Matched Footprints in the Raw Samples
may have localisation errors introduced during the sampling.
However, these errors are typically within a meter. Therefore,
the distance between ground truths are much larger than
localisation errors, and the distance between location values
of Matched Footprints in Raw Samples will be greater than
0. This means these location values are different, and Raw
Samples of different True Trajectories are unique.

![Figure 2: Sampling Raw Samples of two True Trajectories.](image)

The two True Trajectories are generated by two vehicles, \( V_1 \) and \( V_2 \). They drive on a similar route. Green dashed lines outline the road they are driving on and the \( x \) and \( y \) axes are longitude and latitude. The location data are sampled at three timestamps, \( t_1, t_2 \) and \( t_3 \). Each sampled location falls into a range, shown as a shaded circle in the figure. The circle is centred at the ground truth location of the vehicle at the
timestamp and its radius is localisation error upper bound.

Randomness of Raw Samples is related to the ability of
an entity to sample one True Trajectory. Entities which can
sample the True Trajectory better will have advantage in
the knowledge of the True Trajectory, and the Raw Samples
will appear less random to them. We analyse the random-
ness of Raw Samples perceived by three groups: authorised
entities, benevolent unauthorised entities, and attackers. Au-
thorised entities are TCC and the vehicle which generates
the True Trajectory (called Trajectory Owner), e.g., \( V_1 \) in
Figure 2. Benevolent unauthorised entities are unauthorised
entities which may be able to sample the True Trajectory but
do not intentionally try to compromise the security of the
keys. An example of a benevolent unauthorised entity is a
neighbouring vehicle to the Trajectory Owner (called Owner
Neighbour), e.g., \( V_2 \) in Figure 2. Attacker try to compromise
the security of the keys, and will launch attacks to learn
information of the True Trajectory.

Trajectory Owner and TCC both can collect the Raw
Samples with a high accuracy and at a high frequency, as
described above. As sampling errors have an upper bound,
the location values of Matched Footprints in their respective
Raw Samples are within a given range. Because each pair of
Matched Footprints have close values, it is possible for one
of them to predict the Raw Sample of the other based on its
own Raw Sample. Therefore, Raw Samples do not appear to
be random for the authorised entities.

An Owner Neighbour can sample the True Trajectory of
Trajectory Owner, using on-board sensing and object track-
ing techniques [49]. However, this is only feasible when
Owner Neighbour is in a close physical range to the Tra-
jectory Owner. When the distance between them becomes
larger, the samples will have larger errors or the sampling
will fail. Therefore, an Owner Neighbour can have partial
information of Raw Samples of Trajectory Owner’s True
Trajectory. The amount of information is limited by the time
during which they are physically close.

To discuss how random Raw Samples are to attackers,
we need to consider what methods they can use to sample
the True Trajectory of a target Trajectory Owner. Attackers
cannot use the on-board trajectory collection techniques, but
they can use the techniques used by TCC or the neighbour ing
vehicles. That is, they can install their own RSSs and set up
a trajectory data collection system similar to the one used
by TCC. Alternatively, they can follow the Trajectory Owner
and collect Raw Samples. However, both methods are risky
and expensive. The first method needs a large amount of
investment on hardware and software, which may not be
affordable to resource-limited attackers. Also, the installed
RSSs may be discovered by inspection. The second method
is also risky, as the attackers need to follow the Trajectory
Owner closely throughout its journey if they want to sample
the whole True Trajectory. This is a suspicious behaviour
and may be discovered. Also, it is inefficient to only be able
to attack one target at one time, which may discourage the
attackers to do so. To summarise, it is difficult for attackers
to have accurate Raw Samples of a whole True Trajectory.

One feasible attack is learning trajectory information by
eavesdropping. This can be done efficiently by monitoring
the identifiers used by a target Trajectory Owner in public
communication and recording where the identifiers appear,
e.g., with which RSUs the messages with these identifiers are
exchanged. Then the attackers can find out when the target is
in the coverage range of these RSUs. However, if no more
detailed trajectory data are sent in public communication, the
attackers are unlikely to know more accurate information of
the True Trajectory. Other measures can reduce the amount
of trajectory information the attackers can learn with this
eavesdropping attack. For example, if the Trajectory Owner
uses unlinkable identifiers with different RSUs, the attackers
may only be able to know the time when the target is in
one RSU coverage range. If the target takes other privacy-
preserving measures when it is within the range of one RSU,
the attackers may not be able to learn the time it takes to
travel through this area.

To mitigate the eavesdropping threat, we design the secret
extraction method so that even the attackers know when the
target Trajectory Owner is in the coverage range of an RSU,
it will still be statistically hard for them to guess the secrets.

**B. CHALLENGE-RESPONSE SECRET EXTRACTION**

The challenge is designed as a sequence of timestamps.
One of the authorised entities (the verifier) generates the
challenge and sends it to the other entity (the claimant). They both use this challenge to find footprints in their own Raw Samples, and use the location data in the footprints to generate a response. The response is a subset of the Raw Samples, referred to as a Trajectory Sample (TrajSmp). TrajSmp is the extracted temporary secret.

The temporary secret should achieve the expected level of assurance in mutual authentication (SR1), and should be secure enough to protect key confidentiality. This means the temporary secret should be as confidential as the key, because the temporary secret keeps the key confidential when it is distributed using public communication. If the TrajSmp is discovered, the key is no longer a secret. Attackers may use guessing attacks to discover a TrajSmp. To resist these attacks, the extraction should generate TrajSmps with enough randomness. At the same time, to minimise costs (PR1), TrajSmps should provide such randomness with the smallest size. This is because generating and using larger TrajSmps has higher computational and communication costs. We design the secret extraction using the following three ideas.

(TSE1) We expand the sample space of TrajSmps, to increase randomness. With a larger sample space, we increase the number of possible values of a TrajSmp. This makes it statistically harder for attackers to guess the correct TrajSmps. The expansion is done by using both time and spatial information contained in the trajectory data to generate TrajSmp. More specifically, we use a subset of timestamps in the Raw Samples to choose footprints, and using the location data of the chosen footprints to form TrajSmps. Compared to using only the location data, i.e., only the spatial information, this method uses information from two dimensions (i.e., time and space) and the mapping relationship between them. This increases the uniqueness and randomness of the extracted TrajSmp without increasing the size.

(TSE2) To further improve TrajSmp randomness, we reduce the correlation between the footprints in a TrajSmp with three randomisation measures. First, the verifier chooses timestamps from its Raw Sample randomly. Second, the timestamps are chosen without repetition, i.e. one timestamp cannot be chosen twice in one instance. Third, the chosen timestamps are shuffled, i.e., the order of the timestamps in the challenge are randomised so that the timestamp values are not monotonically increasing or decreasing.

(TSE3) The size of TrajSmp is decided according to the securitv level required per instance. A TrajSmp with larger size, i.e., consisting of more footprints, has more randomness. This is because each footprint contributes an amount of randomness and more footprints bring more randomness to the TrajSmp. So, by varying the size of TrajSmp used in a TAKE instance, we can change the security level of the key. To decide the TrajSmp size in an instance, we first estimate how much randomness can be provided in one footprint considering the quality of Raw Samples collected by the authorised entities. Then we calculate how many footprints are needed to generate a TrajSmp with enough randomness to satisfy the security level.

C. KEY ESTABLISHMENT WITH ERROR TOLERANCE

To establish keys with TrajSmps, we need to solve the following problems: error tolerance (FR1, SR1), integrity protection (SR3), key confidentiality (SR2) and high efficiency (PR1).

1) Error tolerance

Due to the independent sampling of True Trajectory and sampling errors, the vehicle and TCC may have nonidentical Raw Samples. As a result, the TrajSmps they extracted using the method above may be nonidentical, but should be accepted as authentic. For clarity, we refer to the TrajSmps generated by authorised entities as Authentic TrajSmps, and the TrajSmps that are within a given error tolerance range to Authentic TrajSmps as Valid TrajSmps. As described in Section IV, TAKE should be able to accept and correct Authentic TrajSmps, while minimising the probability of attackers finding Valid TrajSmps. We refer to this property as reliability. To achieve reliability, TAKE uses three error tolerance measures. ET1 and ET2 are used for error correction, and ET3 is used for setting the smallest error tolerance range.

(ET1) We use a fuzzy extractor to correct errors in TrajSmps. Due to the correctness property, the fuzzy extractor can eliminate the errors between two valid samples. Due to the security property, it can provide confidentiality for both the sample and the base secret while sending the secret using public communication. In other words, the fuzzy extractor make Valid TrajSmps identical and distribute a base secret with confidentiality protection in one step.

Because a fuzzy extractor can only correct errors in discrete samples, we need to quantise each TrajSmp, to transform it from a real-valued location data sequence to a binary string. We call the string BinSmp. Due to the correctness property, only when the verifier’s BinSmp and the claimant’s BinSmp are within a given error threshold (we call these BinSmps Valid BinSmps) can the base secret be established between them. Thus, BinSmp functions as a part of identity proof to mutually authenticate the verifier and the claimant (SR1). However, for higher reliability, the entities are also authenticated with TrajSmp, as described below in ET3.

(ET2) We reduce a certain type of error between Authentic TrajSmps, by designing an error reduction method called Mismatch Correction. Mismatch Correction reduces mismatch errors during the generation of TrajSmps. Mismatch error happens when an authentic entity cannot find an exact match in its Raw Sample to a timestamp in the challenge (a mismatch timestamp). It can be caused by the clocks of the entities falling out of synchronisation or data loss in the data collection process.

Mismatch Correction estimates the location coordinates at the mismatch timestamp, using the available footprints and movement data. Movement data are the sampled data of the movements of an entity, i.e., speed, acceleration and heading. In the IoV, movement data can be collected at the same time with location data, using the same trajectory data collection systems [48] [52]. Movement data have been used for estimating missing data points and for error correction
in vehicle localisation and tracking solutions [48]. Mismatch Correction uses an approach similar to those solutions to estimate the vehicle location at the missing data points.

(ET3) We use a pair of error thresholds, called Dual Error Threshold, to define error tolerance range and verify samples. Because TrajSmp and BinSmp are the same Trajectory Sample in different formats, they both should be valid if the entity is authentic (SR1). It is not accurate to only verify BinSmp, because the quantisation from TrajSmp to BinSmp may introduce additional errors. Because of the different formats and the quantisation error, it is not accurate to use the same way or threshold to verify both BinSmp and TrajSmp. Errors in BinSmps are most accurately measured by the number of different bits between two BinSmps, i.e., the Hamming distance between the two binary strings. Errors in TrajSmps are most accurately measured by the Euclidean distance between each pair of Matched Footprints, i.e., the Euclidean distance between two location points with the same timestamp in two TrajSmps.

The Dual Error Threshold consists of a threshold for the Euclidean distance, called Euclidean Error Threshold (EET), and a threshold for the Hamming distance, called Hamming Error Threshold (HET). EET is used for verifying TrajSmp, while HET is used for the fuzzy extractor and for verifying BinSmp. To verify a pair of TrajSmps, the Euclidean distance between each pair of Matched Footprints is calculated and compared with EET. If all Euclidean distances between Matched Footprints are within the EET, the TrajSmps are accepted as Valid TrajSmps. HET is firstly used in the fuzzy extractor to correct errors in BinSmps. Then, the Hamming distance between the corrected BinSmp and the input BinSmp is calculated and compared with HET. If it is below HET, the correction is proved to be successful and the BinSmp is accepted as valid. Both TrajSmp and BinSmp must be accepted as valid for the verifier and the claimant to be accepted as authentic.

The values of EET and HET should also be chosen carefully to achieve reliability. Because the most accurate measurement of distance between location points is the Euclidean distance, TAKE uses EET as the tighter error threshold. HET value is decided according to EET value, and can be relaxed to tolerate possible additional errors caused by quantisation and ensure all Authentic TrajSmps can be corrected by the fuzzy extractor. We discuss the effects of different EET and HET values on reliability in Section IX.

2) Integrity protection
The data exchanged between entities in TAKE, e.g., the base secret, must be authentic so that the established keys are correct. As the data in transit are under threat of modification attacks, TAKE needs to protect the integrity of the data.

Integrity protection is achieved by generating a checksum on all data sent between the entities, similar to that used in the robust fuzzy extractor construction in [10] and [11]. The checksum is generated using a Universal Hash Function with BinSmp as the key. When the recipient reproduces the identical BinSmp, it uses the same Universal Hash Function and the BinSmp to generate a hash value on all received data, and compare the hash value with the checksum. If they are the same, the received data are proved to be authentic. Otherwise, it is believed that the data are modified and the key establishment is aborted.

3) Key confidentiality
The following specific guessing attacks may be threats to key confidentiality.

(T1) Brute-force-Key-Guess: Attackers may reveal the key with brute-force guessing attacks.

(T2) Brute-Force-Sample-Guess: Attackers may reveal the temporary secret, i.e. the TrajSmp/BinSmp, with brute-force guessing attacks and then use the revealed temporary secret to reveal the key.

(T3) Cryptoanalysis: Attackers may eavesdrop on the data exchange in a TAKE instance, to gain information about the TrajSmps/BinSmp and direct their guess at the TrajSmp/BinSmp and the key.

(T4) Eavesdropping-and-guess: Attackers may gain information about the trajectory data by eavesdropping on IoV communications, and then use this information to direct their guess at TrajSmps.

We mitigate the threats with the following measures.

(KC1) For T1, we want to maximise the randomness of the keys so that it is statistically harder for brute-force attacks to succeed. To do so, we choose to use the key-binding fuzzy extractor construction and a KDF. Compared with the other fuzzy extractor construction, the key-extracting construction [38], the key-binding construction can establish a shared secret with higher randomness using inputs with the same randomness [39]. This means base secrets established with the key-binding construction can be more random and thus more resistant to brute-force attacks. Then, to generate a key with the required security level, i.e., key length, we use a KDF to securely stretch the base secret.

(KC2) For T2, we need to generate a TrajSmp/BinSmp pair with at least the same level of resistance to brute-force attacks as the key. This is done by determining the TrajSmp size in TSE3.

(KC3) For T3, we need to minimise the information about the temporary secret or the key that can be learned from the data exchanged with public communication. These data are the challenge, key parameters for the instance (key length, HET, EET), the secure sketch of the fuzzy extractor, and the checksum for integrity protection. The main source of information leak is the secure sketch and the checksum.

A secure sketch may gradually leak information about a sample if the same sample is repeatedly used [55]. Each secure sketch leaks a certain amount of information about the sample it is generated on, as described in Section V. Every time the sample is used, a different secure sketch is generated on it. Because the secure sketches are sent in public communication, attackers may collect all the secure sketches generated on the same sample. The more times the sample
is reused, the more information about it is leaked, and it becomes more predictable to the attackers. We reduce this information leak by generating a fresh TrajSmp for each instance and randomising the generated TrajSmp. This reduces the chance of reusing the same TrajSmp, so that secure sketch is rarely generated repeatedly on the same sample, and attackers are unlikely to obtain enough information about any one TrajSmp through secure sketches to reveal it. As a result, TAKE will be more resistant to cryptanalysis.

The checksum is a hash value, which may leak information about the input. The information leak is reduced by using Universal Hash Function. Because Universal Hash Function generates nearly uniformly distributed outputs, the outputs are the least correlated with the inputs, and reveal minimum information about the inputs.

Timestamps in the challenge are part of trajectory data and are related to TrajSmps. However, they are not used as a part of TrajSmps. Without knowing the timestamp-location combinations, i.e., the whole footprints, revealing timestamps does not reveal TrajSmp values. Key Parameters do not reveal secret information about TrajSmp or key values.

(KC4) For T4, We calculate the randomness of each footprint in TSE3 considering attacker knowledge of trajectory. Recall the remark we have made regarding Raw Sample randomness, which is attackers can at best know the time period when a certain vehicle is in a certain geographical area. When guessing a TrajSmp, attackers need to find a close enough guess for each footprint in it. That is, they need to find footprints the locations in which are within the EET to the locations of the Matched Footprints in Authentic TrajSmps. With their knowledge, the attackers can guess the locations uniformly in the known geographical area. We calculate the randomness of each footprint based on the probability that the guessed location falls in the EET to the location of the Matched Footprint in an Authentic TrajSmp.

4) Efficiency
For PR1, costs in communication and computation should be minimised, without lowering security level. In the design above, we have chosen the most efficient techniques that can satisfy the functional and security requirements. These choices are using hash function for integrity protection, using KDF to derive keys from base secrets (KC1), and choosing the smallest TrajSmp size according to the required security level (KC2). In addition to these, we further reduce the costs in communication and computation, by using truncation in the transformation from TrajSmp to BinSmp.

Truncating the sample during the transformation can reduce costs in both communication and computation, because a shorter BinSmp results in a secure sketch with a smaller size. A secure sketch with a smaller size requires less computation for error correction and sample verification. Also, as the secure sketch needs to be sent between the entities, less data need to be sent when the secure sketch has a smaller size. However, the truncation should not reduce the randomness of BinSmps. The truncation is possible in this transformation, because in the binarised location values of the footprints, there are invariable bits which do not contribute to the randomness. These bits are considered as redundant and can be discarded without reducing randomness. The truncation is done by keeping the variable bits in the BinSmp and discarding the invariable bits.

VII. TAKE IN DETAIL
A. NOTATIONS

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Footprint Sample Area</td>
</tr>
<tr>
<td>lonrange</td>
<td>Range of the longitudes in the trajectory samples</td>
</tr>
<tr>
<td>latrange</td>
<td>Range of the latitudes in the trajectory samples</td>
</tr>
<tr>
<td>(x, y)</td>
<td>Longitude and latitude coordinates of a location point</td>
</tr>
<tr>
<td>Res</td>
<td>Resolution of a trajectory data collection system</td>
</tr>
<tr>
<td>LEP</td>
<td>Maximum localisation error parameter of a trajectory data collection system</td>
</tr>
<tr>
<td>{v, acc, θ}</td>
<td>An entry of movement data, consisting of speed, acceleration and heading</td>
</tr>
<tr>
<td>MovData</td>
<td>Movement Data, consisting of a sequence of movement data entries</td>
</tr>
<tr>
<td>H∞,x</td>
<td>Min-entropy of a random variable, x</td>
</tr>
<tr>
<td>KeyLen</td>
<td>Key length</td>
</tr>
<tr>
<td>KP</td>
<td>Key Parameters</td>
</tr>
<tr>
<td>l</td>
<td>Security Parameter that specifies a security level of 2^l</td>
</tr>
<tr>
<td>KTény</td>
<td>A trajectory key held by an entity, Ety</td>
</tr>
<tr>
<td>nonce</td>
<td>A nonce</td>
</tr>
<tr>
<td>cEty</td>
<td>A codeword from an error-correcting code, generated by entity Ety</td>
</tr>
<tr>
<td>fp</td>
<td>A footprint</td>
</tr>
<tr>
<td>nfpx</td>
<td>Number of footprints</td>
</tr>
<tr>
<td>t fp</td>
<td>Euclidean error threshold per footprint</td>
</tr>
<tr>
<td>eb</td>
<td>Hamming error threshold per footprint</td>
</tr>
<tr>
<td>tb</td>
<td>Total number of bits of a binary footprint coordinate</td>
</tr>
<tr>
<td>tbs</td>
<td>Hamming error threshold for a BinSmp</td>
</tr>
<tr>
<td>ts</td>
<td>A timestamp</td>
</tr>
<tr>
<td>TS</td>
<td>Timestamp sequence</td>
</tr>
<tr>
<td>TrajDataény</td>
<td>Raw Sample of a True Trajectory collected by an entity, Ety</td>
</tr>
<tr>
<td>TrajSmpény</td>
<td>Real-valued trajectory sample generated by an entity, Ety</td>
</tr>
<tr>
<td>BinSmpény</td>
<td>Binary trajectory sample of an entity, Ety</td>
</tr>
<tr>
<td>ss</td>
<td>A Secure Sketch</td>
</tr>
<tr>
<td>exam</td>
<td>A Checksum</td>
</tr>
</tbody>
</table>

Notations used in TAKE are given in Table I.

B. ARCHITECTURE

The architecture of TAKE is shown in Figure 3. TAKE consists of two modules, a Gen Module and a Rep Module. The Gen Module generates a key and a message for key reproduction (KeyRepMsg). The Rep Module reproduces the key using KeyRepMsg. For two entities, a verifier and a claimant, to use TAKE, the verifier should have the Gen Module, and the claimant should have the Rep Module. In our use case, as shown in Figure 1, TCC acts as the verifier and uses the Gen Module, and a vehicle acts as the claimant and uses the Rep Module.
The Gen Module consists of three algorithms: a Sample Generation Algorithm (SmpGen), a Sample Process Algorithm (SmpProc), and a Trajectory Key Generation algorithm (TrajKeyGen). SmpGen takes as input the verifier’s Raw Sample and a key requirement. It outputs a TrajSmp, \( TrajSmp_v \), the corresponding timestamp sequence to the sample, \( TS \), and Key Parameters, \( KP \). SmpProc takes a TrajSmp, \( TrajSmp_v \), and transforms it into a BinSmp, \( BinSmp_v \). TrajKeyGen uses a BinSmp, \( BinSmp_v \), and Key Parameters, \( KP \), as input. It outputs a key, \( KT_v \), a Secure Sketch, \( ss \), and a Checksum, \( csum \). Finally, the Gen Module collects outputs from the algorithms, keeps \( KT_v \) secret, composes KeyRepMsg with \( TS \), \( KP \), \( ss \) and \( csum \), and sends it to the claimant.

The Rep Module consists of three algorithms: a Sample Reproduction Algorithm (SmpRep), the same SmpProc as in Gen Module, and a Trajectory Key Reproduction algorithm (TrajKeyRep). The Rep Module first parses KeyRepMsg into \( TS \), \( KP \), \( ss \) and \( csum \). SmpRep uses \( TS \), the claimant’s Raw Sample and Movement Data as inputs. It outputs a TrajSmp, \( TrajSmp_c \). SmpProc transforms a TrajSmp into a BinSmp, \( BinSmp_c \). TrajKeyRep uses \( BinSmp_c \), Key Parameters \( KP \), Secure Sketch \( ss \) and Checksum \( csum \) as inputs and tries to reproduce the key. If the reproduction is successful, TrajKeyRep outputs a key, \( KT_c \). Otherwise, it outputs a null value and the key establishment fails.

TAKE is put into operation as follows. First, a security service used by the verifier and the claimant requests to use TAKE to establish a pair of keys. It gives a key requirement in the request. Next, the verifier calls the Gen Module, and provides its Raw Sample and the key requirement as input. Then, the Gen Module runs the algorithms with these input data. After this, it gives the key \( KT_v \) to the security service and sends KeyRepMsg to the claimant through a public communication channel. When receiving KeyRepMsg, the claimant uses the Rep Module to reproduce the key. The Rep Module takes the claimant’s Raw Sample, its Movement Data, and KeyRepMsg as input. Then, it runs the algorithms. If the verifier and the claimant are authentic, the Rep Module will give a reproduced key to the security service. Otherwise, it will notify the security service that the key establishment has failed.

C. ALGORITHMS
This section describes the detailed steps in TAKE algorithms. Pseudocode of the algorithms is given in Appendix A.

1) SmpGen
SmpGen consists of three steps: (1) determining Key Parameters and TrajSmp size; (2) Selecting footprints; (3) Composing TrajSmp, \( TrajSmp_v \), and the challenge timestamp sequence, \( TS \).

In step (1), the Key Parameters to be determined are key length and the Dual Error Threshold. The values are determined according to the given security level and the parameters of the trajectory data collection systems. If the
security level is set as $2^{-l}$, the key length should be $l$ bits. That is, $KeyLen = l$. For the Dual Error Threshold, we first determine the value of EET. EET value is decided according to maximum sampling errors of the trajectory data collection systems. The EET value should be no less than twice the maximum localization error parameter in Euclidean metric, $LEP$. To determine the value of HET, we first need to determine $TrajSmp$ size, as HET is the error threshold for the whole BinSmp. The $TrajSmp$ size is determined according to key length and EET. As described in TSE3, $TrajSmp$ size depends on how many footprints are needed for generating a $TrajSmp$ with enough randomness for the required key length. The randomness that one footprint can provide is measured with its min-entropy, as

$$H_{\infty,fp} = -\log_2((\pi t_{fp}^2)/A),$$  \hspace{1cm} (13)

where $A$ is the Footprint Sample Area, the geographic area from which footprints are selected, and $t_{fp}$ is the EET value. $H_{\infty,fp}$ is an estimation of the maximum likelihood for attackers to guess the location value in a footprint correctly in one attempt with the knowledge of the geological area the footprint is chosen from. For a $TrajSmp$ to establish a key of length $KeyLen$, the number of footprints needed, $n_{fp}$, is determined with

$$n_{fp} = \lceil KeyLen/H_{\infty,fp} \rceil,$$  \hspace{1cm} (14)

where $\lceil \cdot \rceil$ is the ceiling function. After determining $n_{fp}$, the value of HET, $t_{bs}$, can be determined with

$$t_{bs} \geq eb \cdot n_{fp},$$  \hspace{1cm} (15)

where

$$eb = \lceil \log_2(t_{fp}/Res + 1) \rceil.$$  \hspace{1cm} (16)

$Res$ is the resolution of the trajectory data collection systems. It is the smallest distance between two location points that the trajectory data collection systems can distinguish.

In step (2), footprints are selected from the verifier’s Raw Sample randomly and without repetition. In step (3), the location coordinates in the selected footprints are extracted and put into a sequence, $TrajSmp_v$, and the timestamps of them are put into another sequence, $TS$. That is,

$$TrajSmp_v = \{(x_i, y_i)\}, \hspace{1cm} 1 \leq i \leq n_{fp},$$  \hspace{1cm} (17)

and

$$TS = \{t_{si}\}, \hspace{1cm} 1 \leq i \leq n_{fp},$$  \hspace{1cm} (18)

where the $i$-th selected footprint is $\{t_{si}, x_i, y_i\}$.

2) $SmpRep$

$SmpRep$ has three steps: (1) extracting the exact match footprints, (2) Mismatch Correction, and (3) composing the reproduced sample. Step (1) and (2) are both used to extract footprints, while step (3) uses the extracted footprints to compose the sample.

In Step (1), $SmpRep$ uses the timestamp sequence $TS$ from $SmpGen$ outputs to extract footprints from the claimant’s Raw Sample, $TrajData_c$. For each timestamp in $TS, t_{si}$, it searches $TrajData_c$ to find a footprint that has the smallest absolute difference from $t_{si}$. If it finds an exact match, that is, there is a footprint in $TrajData_c$ which has the same timestamp $t_{si}$, it extracts this footprint (an exact Matched Footprint). If not, the timestamp is regarded as a mismatch timestamp, and $SmpGen$ goes to step (2).

Step (2) performs Mismatch Correction (ET2). Suppose the mismatch timestamp is $t_{si}$. First, the algorithm finds the closest footprints, which are the footprints with the timestamps closest to the mismatch timestamp in the claimant’s Raw Sample. This is done by searching for the two timestamps in $TrajData_c$ that have the smallest absolute difference from $t_{si}$. One of them is smaller than $t_{si}$ and the other is larger than $t_{si}$. We denote them as $t_{si}$ and $t_{sr}$, and the footprints they belong to are $(t_{si}, x_i, y_i)$ and $(t_{sr}, x_r, y_r)$.

Second, it finds all movement data between the closest timestamps. This is done by searching the claimant’s Movement Data, $MovData$, to find speed, acceleration and heading data with timestamps between $t_{si}$ and $t_{sr}$. Then, it uses the movement data to calculate the displacement of the claimant between the mismatch timestamp and the closest timestamps as follows.

$$ds_a = v_a \cdot (t_{sa} - t_{sa-1}) + 1/2 \cdot acc_a \cdot (t_{sa} - t_{sa-1})^2,$$  \hspace{1cm} (19)

where $t_{sa}$ and $t_{sa-1}$ are two consecutive timestamps in $MovData$, $t_{si} < t_{sa-1} < t_{sa} < t_{sr}$, $v_a$ and $acc_a$ are the speed and acceleration at timestamp $t_{sa}$, and $ds_a$ is the displacement between $t_{sa-1}$ and $t_{sa}$.

Finally, the approximate location at the mismatch timestamp is calculated by adding the displacements to the locations of the closest footprints. That is, for $t_{si} < t_{sa} < t_{ji},$

$$x_{i,t} = x_i + \sum{ds_a \cdot \cos(\theta_a)},$$
$$y_{i,t} = y_i + \sum{ds_a \cdot \sin(\theta_a)}.$$  \hspace{1cm} (20)

And for $t_{si} < t_{sa} < t_{sr},$

$$x_{i,r} = x_r - \sum{ds_a \cdot \cos(\theta_a)},$$
$$y_{i,r} = y_r - \sum{ds_a \cdot \sin(\theta_a)}.$$  \hspace{1cm} (21)

$\theta_a$ is the heading at timestamp $t_{sa}$.

The final estimation is the average of the two calculated locations. That is,

$$x_i = 1/2(x_{i,t} + x_{i,r}),$$
$$y_i = 1/2(y_{i,t} + y_{i,r}).$$  \hspace{1cm} (22)

The footprint at the mismatch timestamp is then extracted as $(t_{si}, x_i, y_i)$. Step (1) and (2) are repeated until all the timestamps in $TS$ have the corresponding footprints extracted.

Step (3) composes a $TrajSmp$ in a similar way to step (3) in $SmpGen$. With all the extracted footprints, the claimant’s $TrajSmp$, $TrajSmp_v$, is generated as

$$TrajSmp_v = \{(x_i, y_i)\}, \hspace{1cm} 1 \leq i \leq n_{fp},$$  \hspace{1cm} (23)

where the $i$-th extracted footprint is $\{t_{si}, x_i, y_i\}$.\hfill\rlap{\footnotesize This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/}
3) SmpProc
SmpProc has three steps: (1) truncation, (2) binarization, and (3) concatenation. Step (1) and step (2) process each footprint in a TrajSmp individually and sequentially. Step (3) concatenates all processed footprints into one string.

In step (1), a footprint is truncated according to the number of distinguishable locations in the Footprint Sample Area. This number is calculated using the size of the Footprint Sample Area, and the resolution of the trajectory data collection systems. The Footprint Sample Area, \( A \), is considered as a rectangular area between two longitudes and two latitudes. That is, \( A = \text{longrange} \times \text{latrange} \), where \( \text{longrange} \) and \( \text{latrange} \) are the distance between two longitudes and the distance between two latitudes, and they form the length and width of \( A \). The number of distinguishable values the longitude and latitude coordinates of a footprint can take are \( \text{lonrange}/\text{res} \) and \( \text{latrange}/\text{res} \). The total bits that one coordinate can be represented with in binary format are calculated as

\[
\text{tb} = \log_2(\max(\text{lonrange}/\text{res}, \text{latrange}/\text{res})),
\]

(24)
The longitude and latitude coordinates are truncated as \( (\text{lon}, \text{lat}) \), where

\[
\text{lon} = x \cdot 10^5/\text{res} \mod 2^{\text{tb}}, \\
\text{lat} = y \cdot 10^5/\text{res} \mod 2^{\text{tb}},
\]

(25)
where \((x, y)\) are the location coordinates of a footprint.

In step (2), the longitude and the latitude coordinates of a footprint are binarized separately. Each coordinate is converted into a binary string of length \( \text{tb} \). Then the two strings are concatenated.

Step (1) and step (2) are repeated until all location coordinates in the TrajSmp are transformed into binary strings. Then, in step (3), they are concatenated in the same order as in the TrajSmp, and form one binary string, \( \text{BinSmp} \). The length of BinSmp, \( n \), is

\[
n = 2 \cdot n_{fp} \cdot \text{tb}.
\]

(26)

4) TrajKeyGen
TrajKeyGen uses the primitives given in Section [V]. It consists of three steps: (1) generating a base secret, (2) generating a Secure Sketch and a Checksum, (3) key derivation from the base secret. Step (1) generates a nonce as the base secret using a pseudorandom number generator. The length of the nonce is decided by the length of BinSmp, \( \text{BinSmp}_v \), and the value of HET, \( t_{bs} \), as

\[
k = n - d + 1
\]

(27)
Where \( k \) is the length of the nonce, \( n \) is the length of \( \text{BinSmp}_v \), and \( d = 2t_{bs} + 1 \).

Step (2) uses the nonce to generate a Secure Sketch of \( \text{BinSmp}_v \), as in [6]. The nonce is encoded using the encoding function of an error-correcting code, and becomes a codeword, \( c_v \). \( c_v \) is then combined with \( \text{BinSmp}_v \) using XOR operation, and the result is the Secure Sketch, \( ss \).

Then, a Checksum is generated using the Universal Hash function, as in [2]. The Checksum is generated on all the data that will be sent to the claimant and the BinSmp. That is,

\[
csum = h(\text{KeyLen}|t_{fp}|t_{bs}|ss|TS|\text{BinSmp}_v).
\]

(28)
Step (3) uses a KDF, as in [12], to turn the nonce into a key with the length as specified by the security level parameter. That is,

\[
KT = \text{KDF}(\text{nonce}, \text{KeyLen}),
\]

(29)
Where \( \text{nonce} \) is the nonce from step (1).

5) TrajKeyRep
TrajKeyRep is based on the reproduction function of the key-binding fuzzy extractor construction. It consists of four steps: (1) recovering the nonce; (2) verifying sample; (3) verifying data integrity; (4) key derivation from the nonce. Each step can only take place if the previous steps are all successful.

In step (1), first, the BinSmp generated from the reproduced TrajSmp, \( \text{BinSmp}_v \), is used to recover a codeword \( c_v \) from the Secure Sketch, i.e., \( c_v = ss \oplus \text{BinSmp}_v \). Second, we try to recover the nonce by decoding the recovered codeword with the decoding function of the error-correcting code. If the Hamming distance between the recovered codeword \( c_v \) and the original codeword \( c_v \) is within HET, the nonce will be correctly recovered.

Step (2) verifies the verifier and the claimant have Valid TrajSmps and BinSmps. It checks whether the Hamming distance between \( \text{BinSmp}_p \) and \( \text{BinSmp}_v \) is no more than HET, and the Euclidean distance between the locations of each pair of Matched Footprints in \( \text{TrajSmp}_p \) and \( \text{TrajSmp}_v \) is no more than EET. To calculate the Hamming distance, firstly, the recovered nonce is encoded again using the same encoding function as in TrajKeyGen. This is to recover the original codeword \( c_v \). Secondly, \( c_v \) is used to recover the verifier’s BinSmp, \( \text{BinSmp}_v \), i.e., \( \text{BinSmp}_v = ss \oplus c_v \). Thirdly, the Hamming distance between \( \text{BinSmp}_p \) and \( \text{BinSmp}_v \) is calculated. If the Hamming distance is no more than \( t_{bs} \), the algorithm continues to verify the Euclidean distance. To do so, TrajKeyRep transforms the segments in \( \text{BinSmp}_p \) and \( \text{BinSmp}_v \) that are transformed from two location coordinates of one footprint back into decimal format. That is, every \( \text{tb} \) bits of the BinSmps are converted back to a decimal number, and every \( 2\text{tb} \) bits are a segment of location coordinates of one footprint. If the location coordinates are different between the Matched Footprints, the Euclidean distance between them are calculated, and then compared with \( t_{fp} \). The calculation and comparison are repeated for all bits in the BinSmps. If all the Euclidean distances are less than \( t_{fp} \), the TrajSmps are accepted as valid, and the verifier and the claimant are mutually authenticated.

Step (3) verifies the data received from the verifier are authentic. This is done by using the Checksum \( csum \). The recovered \( \text{BinSmp}_p \) in step (2) is used to generate a hash value as in [28]. Then, the hash value is compared with the received \( csum \). If the two are equal, the data are verified as
authentic. This means that the nonce recovered in step (1) is the same as the nonce used by the verifier, and it can be used to derive the same key as the verifier’s.

Step (4) derives a key from the nonce with the same key length parameter and KDF as in \(\text{(29)}\). This is the key established between the verifier and the claimant in this TAKE instance. If any of the above steps fails, TrajKeyRep should output a Null value and notify the claimant that the key establishment has failed.

**VIII. THEORETICAL EVALUATION**

We give theoretical analysis on the security and efficiency of TAKE. The security analysis compares the security properties achieved by the design to the security requirements (SR1-SR4) specified in Section [III-D](#). Efficiency is analysed by estimating computational costs using time complexity analysis, and estimating communication costs.

**A. SECURITY ANALYSIS**

1) Mutual Authentication (SR1)

Mutual authentication is achieved by verifying the validity of TrajSmp and BinSmp. Attackers may compromise the authentication and impersonate authorised entities by finding a pair of Valid TrajSmp/BinSmp or revealing a pair of TrajSmp/BinSmp used in an instance. The level of assurance achieved by TAKE can be measured by the probability of the attackers successfully finding or revealing such a pair. We evaluate this probability considering the attack methods described in Section [VII](#). (Brute-force-Sample-Guess) The attackers try to find a Valid TrajSmp/BinSmp pair by randomly guessing the value of TrajSmp/BinSmp. Attackers use knowledge about the True Trajectory of the claimant.

(Eavesdropping-and-Guess) The attackers eavesdrop on the IoV communication, learn during which time period the claimant is in which geological area, and find a TrajSmp/BinSmp pair based on this knowledge.

(Guess-by-Sketch) This is cryptanalysis on secure sketches. The attackers obtain the Secure Sketch in a target TAKE instance and try to find the TrajSmp/BinSmp pair used to generate this Secure Sketch.

For Brute-force-Sample-Guess to succeed, the attackers can start with finding a Valid TrajSmp or a Valid BinSmp. Intuitively, the probability for guessing a Valid TrajSmp with brute-force attacks (Brute-Force-TrajSmp-Guess, BFTG) is negligible if no knowledge of True Trajectory is used, because it requires the attackers to guess one location from all locations on the Earth surface correctly for several times. Using the EET value determined in SmpGen, the probability for BFTG to succeed is

\[
Pr[BFTG = \text{succ}] = \frac{\pi \cdot t_p^2}{4\pi R_e^2} \cdot n_f, \quad (30)
\]

where \(R_e\) is the radius of Earth. This probability is approximately \(10^{-14}\) for one footprint, and decreases exponentially as \(n_f\) increases.

The probability of guessing a Valid BinSmp with brute-force attacks (Brute-Force-BinSmp-Guess, BFBG) is approximately \(2^{−\text{KeyLen}} \cdot 2^{−n_{fp} (eb + \log_2 \pi)}\) (proof in Appendix [B](#)). This is less than the security level parameter, \(2^{−\text{KeyLen}}\), which means the achieved security is higher than required.

To analyse the success probability of Eavesdropping-and-Guess (EAG), we use a worst case scenario. In this scenario, the Footprint Sample Area is exactly the same area the attackers know the claimant is in. The attackers will randomly choose their guesses within this area. However, because TrajSmp size is determined considering this level of attackers’ knowledge, the probability for EAG to succeed is still approximately \(2^{−\text{KeyLen}}\) (proof in Appendix [B](#)).

We analyse the threat of Guess-by-Sketch using the average min-entropy, \(H_{\infty}(BS|SS)\), as in [5]. The average min-entropy, \(H_{\infty}(BS|SS)\), measures the likelihood of attackers guessing the input BinSmp value correctly with the publicly known Secure Sketch, \(ss\). The value of \(H_{\infty}(BS|SS)\) depends on the min-entropy of the distribution BinSmp from. This means the success probabilities of BFBG and EAG have an impact on the success probability of Guess-by-Sketch. We give the detailed analysis in Appendix [B](#). The analysis shows that when \(eb = 1\), the probability for Guess-by-Sketch with BFBG to succeed is less than \(2^{−\text{KeyLen}}\). When \(eb \geq 2\), this probability is higher than \(2^{−\text{KeyLen}}\). Also, Guess-by-Sketch has a probability of success higher than \(2^{−\text{KeyLen}}\), in the worst case of EAG.

This leaves some vulnerability in terms of providing the required level of assurance, if we perform the mutual authentication by verifying only BinSmp. However, the verification is designed to use both BinSmp and TrajSmp, which may provide remedy in the worst case scenario. As TrajSmp also needs to be valid, there are more strict requirements on which bits in a BinSmp are allowed to be guessed wrong so that the corresponding TrajSmp can pass the verification. That is, the min-entropy of BinSmp distribution, \(BS\), is higher than the estimation in our proof, if both TrajSmp and BinSmp are to be accepted as valid.

Another factor to consider is, when analyzing the min-entropy of \(BS\), we have assumed that \(BS\) is a uniform distribution. That is, every bit in BinSmp should have a min-entropy of nearly 1. This min-entropy depends on the True Trajectory value in addition to SmpGen and SmpProc algorithms, and it is difficult to analyse True Trajectory value theoretically. We measure the min-entropy of BinSmp distributions with experiments in Section [IX](#).

2) Key Confidentiality (SR2)

There are two threats to key confidentiality: Brute-Force-Key-Guess (T1), and Guess-by-TrajSmp.

(Guess-by-TrajSmp) The attackers try to reveal the temporary secret, i.e., the TrajSmp, and then use the TrajSmp and KeyRepMsg to reveal the key.

The security against Brute-Force-Key-Guess (BFKG) depends on the randomness of the base secret, i.e., the nonce, and the security of the KDF. We assume that the KDF
algorithm used in TAKE is secure, as the design of a secure KDF is out of the scope of this paper. The nonce is generated with a pseudorandom number generator (PRNG). We assume the PRNG generates uniformly distributed bits, as the design of a secure PRNG is also out of the scope of the paper. With the nonce length in (2), the probability for attackers to guess the nonce is no higher than $2^{-k}$, which is approximately $2^{-K_{\text{KeyLen}}}$ when HET is set without relaxation.

The security against Guess-by-TrajSmp depends on the confidentiality of TrajSmp. We have analysed the security of TrajSmp under potential attacks in the analysis above, which shows it is unlikely that the attackers could find the TrajSmp used in an instance. As a result, it is unlikely to for Guess-by-TrajSmp to succeed.

3) Data Integrity (SR3)

To compromise data integrity, attackers may use the following two attacks.

(Hash-collision) The attackers find a hash collision and modify the data so that the modified data can be verified with the authentic Checksum.

(Forge-all) The attackers modify both the data and the Checksum so that the modified data can be verified with the modified Checksum.

As given in (1), the probability of finding a collision when using Universal Hash Functions is $2^{-l}$, where $l$ is the length of the Checksum. Therefore, the probability for a Hash-collision attack to succeed is $2^{-l}$.

For a Forge-all attack to succeed, the attackers need to create a Checksum that can verify the modified data. To do so, they must find a Valid BinSmp, or expect a collision. Therefore, the probability for a Forge-all attack to succeed is the larger one between $2^{-l}$ and the probability of finding a Valid BinSmp, which is approximately $2^{-K_{\text{KeyLen}}}$.

4) Forward/backward Secrecy (SR4)

Attacks may use the following attacks to break forward/backward secrecy.

(Correlated-Key) The attackers try to use the revealed keys/nonce from other instances to reveal the key/nonce in a target instance.

(Correlated-Smp) The attackers try to use the revealed temporary secrets, i.e., TrajSmp/BinSmp pairs, to reveal the TrajSmp/BinSmp pair in a target instance, and then use the revealed temporary secrets to reveal the key.

Correlated-Key is mitigated by using a fresh nonce for each instance. The nonces in different instances are independent, provided that the PRNG is secure. Therefore, knowing keys/nonce from other instances does not increase the probability for them to reveal the key/nonce in the target instance.

Correlated-Smp is mitigated by generating a fresh TrajSmp for each instance and randomising the generation. However, different TrajSmyps generated for the same authorised entities are naturally correlated for two reasons. First, they are subsets of the same samples of the data source, i.e., the same TrajData. Second, the sampled data of the True Trajectory are correlated. This is because each footprint value is correlated with the value of the previous footprint, as the claimant moves from the previous location to the current location and the movements are continuous. The correlation between these TrajSmps is affected by the correlation of the sampled data, the size of TrajData owned by authorised entities, and the random selection of footprints. If the True Trajectory is more random, i.e., the claimant experiences more transitions between states during its travel, the sampled data will be less correlated, and the TrajSmps generated from these data will be less correlated. When TrajData has a larger size, the random selection has more footprints to choose from, and it is less likely that the TrajSmyps of two instances will use the same footprints. In addition, the selected footprints may be more sparsely distributed, i.e., the footprint timestamps have larger intervals or the footprint locations have a larger distance between them. The more sparsely distributed footprints are less correlated.

It is difficult to quantitatively analyse the correlation between TrajSmps in different instances theoretically, because it is affected by the value of a True Trajectory and randomisation. We analyse the correlation quantitatively by evaluating min-entropy with experiments in Section IX.

B. EFFICIENCY ANALYSIS

1) Time Complexity

The time complexity of TAKE algorithms is given in Table 2. In the table, $O(\text{Hash})$ and $O(\text{PRNG})$ respectively denote the complexity of a hash function and the PRNG. The complexity of the KDF is also represented with $O(\text{hash})$, as KDF is typically based on a hash function. $O(C)$ and $D$ denote the complexity of the encoding/decoding functions of the error-correcting code used by the Secure Sketch. $n_{fp}$ and $n_{rec}$ are respectively the number of footprints chosen, and the number of footprints in TrajData. In this analysis, the search function in SmpRep is assumed to be a linear search through $n_{rec}$ records.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SmpGen</td>
<td>$O(n_{fp})$</td>
</tr>
<tr>
<td>SmpRep</td>
<td>$O(n_{fp} \times n_{rec})$</td>
</tr>
<tr>
<td>TrajProc</td>
<td>$O(n_{fp})$</td>
</tr>
<tr>
<td>TrajKeyGen</td>
<td>$O(C) + O(\text{PRNG}) + 2 \times O(\text{hash})$</td>
</tr>
<tr>
<td>TrajKeyRep</td>
<td>$O(D) + 2 \times O(\text{hash}) + O(2 \times n_{fp} + n_{rec})$</td>
</tr>
</tbody>
</table>

Using the results in Table 2, the time complexity for the verifier and the claimant to use TAKE can be estimated. As shown in Figure 3, the verifier generates a key by running SmpGen, SmpProc and TrajKeyGen. The total complexity is $O(C) + O(\text{PRNG}) + 2 \times O(\text{hash}) + 2 \times O(n_{fp})$. The claimant reproduces the key by running SmpRep, SmpProc and TrajKeyRep. The total complexity is $O(D) + 2 \times O(\text{hash}) + O(2 \times n_{fp} + n_{rec}) + O(n_{fp})$. 

VOLUME 4, 2016

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/
2) Communication Costs

Communication costs are determined by the size of the message exchanged in a TAKE instance. Only one message is exchanged, i.e., KeyRepMsg. KeyRepMsg consists of the challenge timestamp sequence TS, the Secure Sketch ss, the Checksum csum, Key Parameters KP, parameters for the Universal Hash Functions, and depending on the KDF used, an optional salt value.

To describe the size of these items, we assume the size of a timestamp value is \( l_{ts} \) bits, the size of a real value is \( l_{real} \) bits, the size of an integer is \( l_{int} \) bits, the size of BinSmp is \( n \) bits and the size of the Checksum is \( l_{csum} \) bits. We assume a salt value is used in the KDF. The total size of KeyRepMsg is

\[
n_{fp} \cdot l_{ts} + l_{real} + 6 \cdot l_{int} + n + l_{csum}.
\]

3) Impacts of Communication Errors

Two types of communication errors may happen during a TAKE instance, namely, packet loss and bit error. We give a theoretical analysis on how these two types of errors may affect the efficiency of TAKE.

Packet loss refers to the case where a packet is lost during transmission. In TAKE, this means KeyRepMsg or part of this message is lost to the claimant. This would cause a timeout and this TAKE instance will fail. The claimant will know of the failure as it does not receive KeyRepMsg in time. The verifier would know it has failed due to the lack of acknowledgements when it tries to use the trajectory key of this instance. Then, they may try to establish another key. This means that a new TAKE instance will be initiated. The new instance is independent from the failed instance, i.e., all the steps of TAKE will be carried out again for the new instance. The consequence of this is that the time taken for a successful key establishment is longer. It is the number of TAKE instances used until the key is successfully established multiplied by the the timeout value.

Bit error refers to the case where the sender of a message sends a bit 1 and the receiver receives 0 as the bit, or the sender sends 0 and the receiver receives 1. When bit errors happen in KeyRepMsg, there may be two outcomes. One is the bit error can be corrected with the error tolerance capability of TAKE. In this case, the success or failure outcome of this TAKE instance will be the same as if there is no bit error. The computational costs are increased slightly, as more bits are corrected during the decoding and possibly more verification for EET is needed in TrajKeyRep. This increase is negligible compared to the costs of running a new TAKE instance. The other outcome is the bit error cannot be corrected with the error tolerance capability of TAKE. In this case, the key establishment will fail, and the Rep Module of the claimant will notify the security service of the failure result. Then, the entities may try to establish another key using another TAKE instance, until they succeed. This is similar to when a packet loss happens.

Therefore, communication errors may cause a TAKE instance to fail. If the entities choose to retry until they establish a key successfully, the time it takes is estimated as \( N_{TI} \cdot t_{TO} \), where \( N_{TI} \) is the number of TAKE instances they have used until they establish the key, and \( t_{TO} \) is the timeout value set for a TAKE instance.

IX. EXPERIMENTAL EVALUATION

This section evaluates the performance of TAKE in a realistic scenario, and investigates the impacts of multiple factors on the performance. Experimental results are analysed and discussions are made on how to improve TAKE performance and security by choosing suitable parameter values.

A. EVALUATION METHODOLOGY

We first give the metrics used for the evaluation. Second, we describe an experiment scenario where a travelling vehicle is using TAKE to establish a session key with TCC. Third, we describe how to generate the Raw Samples in the scenario. Fourth, we design the test cases to evaluate the impacts of different factors on the security and performance of TAKE. Finally, we describe the implementation of TAKE algorithms we use to conduct the experiments.

1) Evaluation Metrics

The evaluation quantitatively analyses the reliability, randomness and efficiency of TAKE. We evaluate reliability of key establishment results and randomness of temporary secrets to investigate how well TAKE can satisfy SR1 and SR2. For efficiency, we only consider the computational costs in the experiments, as the communication costs are low and do not vary much between instances according to the theoretical analysis. Computational costs have more influence on the overall efficiency. Higher reliability, higher randomness and higher efficiency means better performance and security.

Reliability is measured with True Positive Rate (TPR) and True Negative Rate (TNR). A true positive result means a key is successfully established between authorised entities. A true negative result means an authorised entity has refused to establish a key with an unauthorised entity, despite the attempt made by the unauthorised entity. TPR and TNR are calculated as

\[
TPR = \frac{N_{TP}}{N_{Test}}, \quad TNR = \frac{N_{TN}}{N_{Test}},
\]

where \( N_{Test} \) is the total number of tests, \( N_{TP} \) is the number of true positive results out of all tests, and \( N_{TN} \) is the number of true negative results out of all tests. Higher TPR and higher TNR indicate higher reliability.

Randomness is measured with the min-entropy of a BinSmp distribution, BSME (BinSmp Min-Entropy). The min-entropy is evaluated using a statistic tool, NIST Entropy Assessment Tool [56]. It takes as input a large number of samples output by a random source, and gives an estimation of min-entropy per bit, BME. We calculate BSME value as

\[
BSME = n \cdot BME,
\]

where \( n \) is the length of a BinSmp. Higher BSME indicates higher randomness.
Efficiency is measured with Key Generation Time (KGT) and Key Reproduction Time (KRT). KGT is the total time for the verifier to generate a key and a KeyRepMsg with TAKE Gen Module. KRT is the total time for the claimant to reproduce the key with TAKE Rep Module. Lower KGT and lower KRT indicates higher efficiency.

2) Experiment Scenario

We build the following scenario to conduct the experiments. A vehicle called TgtV (Target Vehicle) is moving from one RSU coverage range to another. It needs to disconnect from the previous RSU and connect to the next RSU to start a new session with TCC. At the same time, another vehicle, FlwV (Follower Vehicle), is travelling behind TgtV on the same route. FlwV is not intentionally trying to steal TgtV’s keys. TgtV and FlwV are the only vehicles on the road. There is no interruption to their journey apart from traffic lights. TgtV and FlwV take the sampled trajectory data of TgtV to establish a key. TgtV acts as a claimant and TCC acts as a verifier. This scenario is designed to investigate the performance and security of TAKE in a worst-case scenario. It is a worst case because there are few interruptions to TgtV’s journey, so there is less randomness and less uniqueness to its True Trajectory. This will lead to lower randomness of the TrajSmp/BinSmp generated from the Raw Samples of TgtV’s True Trajectory, and it is more likely that FlwV may accidentally reveal the established key using its own sampled data, the Raw Samples of FlwV’s True Trajectory. If TAKE is plausible in this disadvantageous scenario, it should be able to achieve the expected performance in other scenarios.

3) Trajectory Data Generation

As TAKE is used when TgtV is changing its local access points (i.e. RSUs), TgtV should have moved for some distance, e.g., the coverage range of one RSU, and its journey should last for a suitable time, e.g., the time it takes to travel through the coverage range at legal driving speeds. The coverage range of RSUs can vary significantly in real life, e.g., from 300 m (required in the United States Department of Transportation RSU specifications [57]) to 2500 m (estimated by RSU manufacturers in an ideal environment [58] [59]). In the experiments, it is assumed that RSUs are placed at an interval of 1 km, to provide more reliable continuous communication. In addition, as in Section III-E, TCC and vehicles should collect trajectory data with high frequency, precision and accuracy. However, to our best knowledge, there is no available real-world dataset of real-time trajectory that are collected for this scenario and meet the data quality requirements. Therefore, we generate the trajectory data with simulation.

The data generation takes two steps. First, we generate the True Trajectories of TgtV and FlwV with traffic simulation. Then, we generate the Raw Samples collected by TCC and the vehicle from the simulation results.

The traffic simulation is performed by implementing the SUMO [60]. SUMO is an open-source traffic simulation tool that has been applied to multiple vehicular network simulation studies [61] [62] [63]. The simulation scenario is built as shown in Figure 4. It is set in the urban area of Karlsruhe, Germany. This city is chosen because it is used for developing the KITTI benchmark test suite [64], which tests the performance of vehicle localisation and tracking techniques. The test results are later used in the generation of Raw Samples. The map of Karlsruhe is obtained from OpenStreetMap [65]. The map area includes driveways of single lanes and multiple lanes. We define the route of TgtV and FlwV, which consists of 23 roads and 4 crossings with traffic lights. The route is approximately 1.2 km. When travelling within the speed limit of urban traffic (50 km/h in Germany [66]), TgtV and FlwV take approximately 90 seconds to finish the route, starting from a stop state. The total simulation time is set to be 300 s, so that the vehicles can have enough time to finish the route even if they need extra time to wait for traffic lights. FlwV is set to depart 10 s after TgtV departs.

SUMO is set to collect the location data and movement data of TgtV and FlwV at sampling timestamps during the simulation. All data are collected at a sampling rate of 100 Hz. Location coordinates are represented in the WGS 84 reference system, at a resolution of 0.000001. We form True Trajectory data with the timestamps and the location coordinates at the timestamps.

Next, we generate Raw Samples as collected by the entities. This is done by adding sampling errors to the True Trajectory data, to imitate the data quality provided by a real-world trajectory data collection system. We consider two types of sampling errors: missing record error and localisation error. Missing record error refers to the cases where the
vehicle locations are not recorded at some sampling timestamps. Missing record error can be caused by an outage of GPS signal, or association errors in vision-based localisation and tracking systems. Localisation error refers to the difference between the sampled locations and the ground truth locations. We assume for a trajectory data collection system, Missing Record Rate (MRR) and Localisation Error Range (LER) are given in system parameters as the upper bounds of the errors, and all sampled data have errors within the upper bounds. We generate Raw Samples, i.e., \( \text{TrajData} \), for TCC, TgtV and FlwV respectively. We reduce the sampling rate to 10 Hz and add random errors within MMR and LER to each timestamp-location tuples in the True Trajectory data.

We use the movement data sampled during the simulation to form MovData, by only reducing the sampling rate to 10 Hz and adding no sampling errors. This is because, compared to the sampling errors in location data, the sampling errors in movement data are negligible. At the sampling rate of 10 Hz, small sampling errors in speed, acceleration or heading will not significantly change the location coordinates calculated by Mismatch Correction. The algorithm for generating \( \text{TrajData} \) and MovData from simulation results is given in Algorithm 6 in Appendix A.

4) Test Design
Tests are designed to investigate the impacts of three factors: security level, error tolerance capability, and trajectory data quality. The impacts are investigated by changing four variables in test cases: key length \( \text{KeyLen} \) for security level, EET/HET pair \((t_{fp}, t_{bs})\) for error tolerance capability, and MMR and LER values \( \text{MMR} \) and \( \text{LER} \) for trajectory data quality. We build a Baseline Case using typical parameter values, to obtain a baseline performance for comparison. Then we build a test case for each of the four variables.

Tests are performed as follows. Due to the randomisation in TAKE algorithms and the random nature of sampling errors, key establishment is repeatedly performed to obtain statistically significant results. The repetition is done at two levels. First, the Raw Sample data, \( \text{TrajData}_a \) and \( \text{TrajData}_c \), are generated for \( N_D \) times, for each set of variable values. This is to simulate random sampling errors in the raw data. Second, key establishment tests are run for \( N_{KE} \) times using the same \( \text{TrajData}_a / \text{TrajData}_c \) pair. In each of the key establishment tests, a key is established using TAKE Gen and Rep Modules. This repetition is to simulate the situation where the same Raw Samples are used for multiple TAKE instances. The total number of key establishment tests run for each test case are \( N_{Test} = N_D \cdot N_{KE} \).

In each key establishment test, TCC uses TAKE Gen Module with \( \text{TrajData}_a \) generated from TgtV’s True Trajectory as input. TgtV uses TAKE Rep Module with \( \text{TrajData}_a \) generated from its own True Trajectory and MovData, generated from its simulated movement data as input. FlwV also acts as a claimant and uses TAKE Rep Module, but it uses \( \text{TrajData}_f \) generated from FlwV’s True Trajectory and MovData generated from its movement data as input. If TgtV successfully reproduces the key in a key establishment test, we record one True Positive (TP) result. If FlwV does not reproduce the same key, we record one True Negative (TN) result. After repeating the tests for \( N_{Test} \) times, we will record \( N_{TP} \) TP results and \( N_{TN} \) TN results. We use these to calculate TPR and TNR. We also record in each test the time it takes TCC to finish running TAKE Gen Module as KGT, and the time it takes TgtV or FlwV to finish running TAKE Rep Module as KRT. After \( N_{Test} \) tests, we use the average of the recorded KGT and KRT as the estimated KGT and KRT for the test case. In addition, we collect the BinSmp generated in each test. After \( N_{Test} \) tests, we use all the collected BinSmps as input to the NIST Entropy Assessment Tool, and obtain the estimation of BME and BSME.

5) Implementations
TAKE algorithms are implemented in Python 3. The error-correcting code used in TrajKeyGen and TrajKeyRep is Reed-Solomon code, a linear code. It is implemented in Python package unireedsolomon [67]. Because the implementation has a maximum codeword length of 255 bits, if a BinSmp is longer than 255 bits, it is split into multiple blocks, and each block is combined with a different codeword and then concatenated together to form a Secure Sketch. The nonces for all the codewords are concatenated together as the input to KDF. The KDF used in TrajKeyGen and TrajKeyRep is PBKDF2, a widely used key derivation function [43]. The kernel hash function of the KDF is SHA-512.

The tests are run on a computer with 8 Intel(R) Core(TM) i5 CPU with 1.6 GHz clock speed and a 8 G RAM, in the Windows Subsystem Linux environment.

B. BASELINE CASE AND TEST CASES
We first describe the baseline case and the choice of parameter values in this case. Then we describe how the four variables are changed in the test cases.

1) Baseline Case
The parameter values that need to be determined are Footprint Sample Area (\( A \)), length of a location coordinate in the binary format (\( t_b \)), MRR (\( \text{MMR} \)), LER (\( \text{LER} \)), EET/HET (\( t_{fp}, t_{bs} \)), resolution (\( \text{Res} \)), number of erroneous bits allowed in one footprint (\( \text{eb} \)), and security level \( \text{KeyLen} \).

The Footprint Sample Area is set as a square area with the RSU coverage range as the length of the edge. As the RSU coverage range is assumed above to be 1 km, the range the longitude and latitude coordinates can change in the Footprint Sample Area is 1 km. Using (24), \( t_b \) is set as 10.

\( \text{KeyLen} \) is set as 128 bits, as it is a typical key length for AES, a standard encryption method. Then we calculate the number of footprints needed using (14), and set \( n_{fp} = 8 \) for \( \text{KeyLen} = 128 \) bits.

MMR and LER are set according to the performance of current vision-based vehicle localisation and tracking methods, evaluated in research settings using the KITTI benchmark test suite [64]. Among the performance metrics [68],
MOTA (Multi-Object Tracking Accuracy) is used to evaluate the probability of missing record error (MRR = 1 - MOTA), and LocA (Localisation Accuracy) is used to evaluate the localisation error (LER = 1 - LocA). The median values of MOTA and LocA out of 86 vehicle localisation and tracking methods are 0.7929 and 0.8472. The MMR and LER are thus set as 0.2 and 0.3. LER is set larger than the median value of localisation error, to allow for more errors that may be introduced by using GPS-based localisation methods.

The value of EET, \(t_{fp}\), is set as 1 m, as the upper bound of LocA is 1 m and a data point with a localisation error over 1 m is regarded as a missing record error. \(eb\) is set according to Eq. (16). As \(t_{fp} = 1\), \(eb \geq 1\). In the Baseline Case, \(eb\) is relaxed and set as 2 for more stable error-correcting performance. Then the value of HET, \(ts\), is set as 16, according to Eq. (15).

Resolution \(Res\) is set according to EET. As EET value is set as 1 m, \(Res\) is set to be 1 m. For test cases where Euclidean threshold is less than 1 m, \(Res\) is set as 0.1 m.

In addition, we need to decide number of repeated tests for the test cases. We set \(N_{KE}\) as 100. Then, we determine \(N_D\) by running 100 key establishment tests repeatedly using the parameters above, until TPR and TNR converge. That is, the mean values of TPR and TNR both fall into a confidence interval less than 0.001 at 95% confidence level. It is observed that in Baseline Case, TNR is 100% in all tests. The results for TPR is shown in Figure 5. Confidence interval of TPR falls below 0.001 after \(17 \times 100\) tests. We set \(N_D\) as 25, to allow for larger variation in TPR between tests in test cases. This is because parameter values in Baseline Case are chosen carefully to provide a reliable key establishment performance. In test cases, with larger sampling errors, smaller thresholds or larger key length, TPR may vary more widely.

![Figure 5: Test results of TPR in Baseline Case](image)

2) Test Cases

In each test case, one parameter value is varied, and all the other parameter values are the same as in Baseline Case.

- (Test Case 1: variable key length) \(KeyLen\) is varied from 128 bits to 256 bits, with a step of 16 bits.
- (Test Case 2: variable EET/HET) \(t_{fp}\) is varied from less than Baseline Case LER to more than the upper bound of LocA. Because \(t_{fp}\) affects \(n_{fp}\) and thus affects \(ts\), we set \(eb\) for each \(t_{fp}\) value instead of \(ts\) in accordance with each \(t_{fp}\) value. \(t_{fp}\) values used for Test Case 2 are (0.25 m, 2 bits), (0.5 m, 3 bits), (0.75 m, 3 bits), (1 m, 4 bits), (1.25 m, 4 bits), and (1.5 m, 4 bits).
- (Test Case 3: variable MRR) \(MRR\) is varied from 0.1 to 0.9 with a step of 0.1.
- (Test Case 4: variable LER) \(LER\) is varied from the median value to LocA upper bound. \(LER\) takes values at 0.3 m, 0.5 m, 0.7 m, 1 m, 1.3 m, 1.5 m, 1.7 m, 2 m.

C. RESULTS AND OBSERVATIONS

1) Test Case 1

TPR is shown in Figure 6(a). TNR is 100% for all tests. The results show that in Test Case 1, TAKE can reliably distinguish between TgtV and FlwV, and establishes keys only for TgtV and TCC. TPR shows small fluctuations as \(KeyLen\) changes, and is slightly higher for smaller key length (128 bits – 192 bits). But TPR is higher than 99% for all key length values. This means varying the security level does not have a large impact on reliability.

KGT and KRT are shown in Figure 6(b). Time needed for key establishment is longer as key length increases, in both TP and TN cases. KRT is higher than KGT in both TP and TN cases. This means the claimant has higher computational costs than the verifier. The higher costs may be caused by decoding and verification. The fast increase in KRT between 208 bits and 224 bits is due to the implementation. As \(KeyLen\) increases, a longer BinSmp is used, and one more codeword is needed when BinSmp length grows to more than 255 bits. KGT and KRT test results indicate efficiency decreases as security level improves.

BME and BSME are given in Table 3. Min-entropy per bit in the BinSmps is approximately 0.5 for all KeyLen values. As \(KeyLen\) increases, BSME increases linearly. This means the randomness of BinSmp increases as \(KeyLen\) increases. However, BSME is smaller than \(KeyLen\), which means in this test case, BinSmp does not ensure the expected security level. We explain the reason for this and how to improve the randomness of temporary secrets for better security in Further Discussions.

2) Test Case 2

TPR is shown in Figure 7(a). TNR is also 100% for all tests. Compared to the TPR in Baseline Case, TPR is lower when EET value is set as less than LER. As EET value increases to the same value as in Baseline Case (approximately 2 LER), TPR increases to the similar level as in Baseline Case. The test results also indicate using a smaller \(Res\) value does not significantly affect reliability.

KGT and KRT are shown in Figure 7(b). KGT shows fluctuations but does not increase or decrease significantly as...
Table 3: BinSmp Randomness in Test Case 1

<table>
<thead>
<tr>
<th>KeyLen (bits)</th>
<th>128</th>
<th>144</th>
<th>160</th>
<th>176</th>
<th>192</th>
<th>208</th>
<th>224</th>
<th>240</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>BME</td>
<td>0.5295</td>
<td>0.4953</td>
<td>0.5251</td>
<td>0.5159</td>
<td>0.5015</td>
<td>0.5063</td>
<td>0.5016</td>
<td>0.4983</td>
<td>0.4965</td>
</tr>
<tr>
<td>BSME</td>
<td>85</td>
<td>79</td>
<td>95</td>
<td>103</td>
<td>110</td>
<td>122</td>
<td>130</td>
<td>140</td>
<td>149</td>
</tr>
</tbody>
</table>

EET and HET change. However, KRT increases significantly, and KRT in TN cases increases more than KRT in TP cases. This is caused by the increase of EET, which causes an increase in \( n_{fp} \). The increase of both \( n_{fp} \) and \( eb \) causes an increase in HET, which leads to more computational costs in decoding. As the BinSmPs in TN cases have more error bits than those in TP cases, the decoding costs increase more in TN cases. Therefore, KRT increases faster in TN cases than in TP cases. KGT and KRT in Test Case 2 show efficiency decreases with higher error tolerance capability.

BME and BSME are given in Table 4. BME does not vary significantly, but BSME increases. This is due to the increase of BinSmp length. Due to the increase of EET value and the resulting increase of \( n_{fp} \), BinSmp length increases from 168 bits when \( t_{fp} \) is 0.25 m, to 196 bits when \( t_{fp} \) is 0.5 m and 0.75 m, and finally to 224 bits when \( t_{fp} \) is no less than 1 m. This means increasing error tolerance range improves the randomness of BinSmp. Compared to Baseline Case, BSME in Test Case 2 is larger when \( t_{fp} \) is 1 m. This is caused by using a smaller resolution. This means a smaller resolution could improve security.

Table 4: BinSmp Randomness in Test Case 2

<table>
<thead>
<tr>
<th>( t_{fp} ) (m), ( eb ) (bits)</th>
<th>(0.25, 2)</th>
<th>(0.5, 3)</th>
<th>(0.75, 3)</th>
<th>(1, 4)</th>
<th>(1.25, 4)</th>
<th>(1.5, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BME</td>
<td>0.5267</td>
<td>0.4810</td>
<td>0.5249</td>
<td>0.5444</td>
<td>0.5400</td>
<td>0.5403</td>
</tr>
<tr>
<td>BSME</td>
<td>88</td>
<td>94</td>
<td>103</td>
<td>122</td>
<td>121</td>
<td>121</td>
</tr>
</tbody>
</table>

3) Test Case 3
TPR is shown in Figure 8(a). TNR is 100% in all tests. TPR decreases faster as MRR keeps increasing. It decreases from 99.92% to 99.56% when MRR changes from 0.1 to 0.3. The drop accelerates for larger MRR, and TPR decreases from 98.84% when MRR is 0.4 to 5.04% when MRR is 0.9. This may be caused by the worsening performance of Mismatch Correction. With a smaller MRR, it is more likely to find more accurate closest footprints to the missing footprint. As MRR increases, more footprints are lost and it becomes harder to find accurate closest footprints in Raw Samples, so the estimation becomes less accurate.

KGT and KRT are shown in Figure 8(b). KGT and KRT vary in a small range for all MRR values. This may be because TrajSmp/BinSmp size and HET are the main factors that affect computational costs, and they do not change in Test Case 3. The results indicate change in MRR does not affect the efficiency of TAKE significantly.

BME and BSME are given in Table 5. BME is similar when MRR changes. Because the BinSmp size does not change, BSME is similar for all MRR values too. This means MRR does not affect the security of TAKE significantly.

Table 5: BinSmp Randomness in Test Case 3

<table>
<thead>
<tr>
<th>MRR</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>BME</td>
<td>0.5226</td>
<td>0.5204</td>
<td>0.5265</td>
<td>0.5255</td>
<td>0.5269</td>
<td>0.5272</td>
<td>0.5218</td>
<td>0.5185</td>
<td>0.4685</td>
</tr>
<tr>
<td>BSME</td>
<td>84</td>
<td>83</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>83</td>
<td>83</td>
<td>75</td>
<td>65</td>
</tr>
</tbody>
</table>
4) Test Case 4

TPR is shown in Figure 9 (a). TNR is 100% in all tests. TPR decreases as LER keeps increasing. The decrease is small when LER is still smaller than EET value, e.g., TPR decreases from 99.88% to 99.44% when LER changes from 0.3 m to 0.7 m. When LER is larger than EET value, TPR decreases linearly, from 97.08% when LER is 1 m to 61.88% when LER is 2 m. This means if the actual localisation errors in Raw Samples increase to more than the expected value given in system parameters, TAKE becomes less reliable.

KGT and KRT are shown in Figure 9 (b). KGT and KRT again vary in a small range for all LER values, for the same reason as in Test Case 3. This means change in LER does not affect the efficiency of TAKE significantly.

BME and BSME are given in Table 6. BME shows an increase as LER increases. This is because the localisation error per footprint varies in a larger range, as LER increases. Therefore, the location coordinates become more random, and the BinSmp bits become less correlated. Due to the increase of BME and the constant BinSmp size, BSME increases as LER increases. This means larger LER may improve the security of TAKE. However, this is not the best way to improve security, as it comes at the cost of reliability.

<table>
<thead>
<tr>
<th>LER (m)</th>
<th>0.3</th>
<th>0.5</th>
<th>0.7</th>
<th>1</th>
<th>1.3</th>
<th>1.5</th>
<th>1.7</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BME</td>
<td>0.5315</td>
<td>0.5338</td>
<td>0.5405</td>
<td>0.5538</td>
<td>0.5614</td>
<td>0.5699</td>
<td>0.5736</td>
<td>0.5668</td>
</tr>
<tr>
<td>BSME</td>
<td>85</td>
<td>85</td>
<td>86</td>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>91</td>
</tr>
</tbody>
</table>

D. FURTHER DISCUSSIONS

1) Improve Temporary Secret Randomness

From the test results above, we find that with some parameter values, the temporary secrets, i.e., BinSmp, do not have the randomness required by the security level parameter. This
may affect the level of assurance of mutual authentication and the confidentiality of the established key. The lower randomness may be caused by the worst case scenario we choose for the evaluation. During the simulation, we have observed that the vehicles are driving with constant speed for most of the journey. The traffic lights are green when the vehicles drive past the crossings. As traffic lights are the only factor that interrupts the driving, the vehicles only have two transitions of states: from a stop state to an acceleration state, and from the acceleration state to a stable state. Without transitions, the True Trajectory of TgtV is generated in a long period of stable state and is not high in randomness. As a result, the temporary secrets extracted from the sampled data of this True Trajectory does not have high randomness. However, with the randomisation and the choice of the sample size, the temporary secrets still provide significant amount of randomness, i.e., a min-entropy of at least 84 bits in all test cases with normal trajectory data quality.

Another possible cause for the lower entropy may be the repeated use of the same footprints in our experiments. As we did $25 \times 100$ runs for each test, and in each run at least 8 footprints are used, we used at least 20000 footprints per test. This is much larger than the total numbers of footprints generated as the trajectory data in the experiments, so we had to use repeated footprints in the experiment. If the same footprints are used in two instances, the BinSmps will have some identical bits. This makes it more easy to predict one BinSmp from another. However, in reality, it is unlikely that footprints have to be repeatedly used, as the claimant and the verifier are unlikely to repeat TAKE for this many times to establish a key for the same session, so they have enough footprints to choose from. Therefore, we can enforce TAKE to choose different footprints in every instance. This increases storage costs as TAKE needs to
store which footprints have been used, but it can reduce the correlation between trajectory samples of different instances. This can help improving forward secrecy as well, as even if the attackers have revealed a trajectory sample of one TAKE instance, it gives little help for them to reveal the trajectory sample in another instance, as different timestamps and location data are used, and the trajectory samples are less correlated.

We now discuss what other factors can contribute to a higher temporary secret randomness, i.e., higher BSME, apart from increasing the randomness of True Trajectory. We observe that BSME increases in three test cases: increasing KeyLen (Test Case 1), increasing EET (Test Case 2), and increasing LER (Test Case 4). In Test Case 1, the increase of BSME is due to the increase of both TrajSmp and BinSmp sizes. To achieve the security level indicated by a larger KeyLen, more footprints are used to form a TrajSmp/BinSmp pair of a larger size, thus improving total randomness. In Test Case 2, the increase is mainly caused by increased BinSmp size, due to the use of a smaller resolution. In Test Case 4, the increase is due to increasing min-entropy per bit, which is caused by larger localisation error, i.e., higher randomness in sampled data.

Therefore, to improve temporary secret randomness, we could choose from the following measures: (1) use True Trajectory with higher randomness, i.e., more transitions between states; (2) increase sample size, i.e., use more footprints; (3) reduce resolution value, i.e., increase the total bits a location coordinate is discretised into; (4) allow larger localisation error. For (1), we do not have much control on the transitions between states. The transitions are decided by the driving actions, which must obey traffic rules and attend to user preferences. We could choose to store more trajectory data to include more randomness in them, but this causes higher storage costs. Measure (2) is effective and do not lower reliability, but it causes higher communication and computational costs. Measure (3) causes higher costs too, and if the resolution is chosen to be too small, reliability may become lower. This is because errors between discretised Authentic TrajSmmps become larger and more difficult to correct with a smaller resolution. We cannot use (4), because reliability decreases considerably as localisation error increases. Also, we do not have much control over localisation error.

2) Improve Reliability

Improving reliability helps TAKE achieve better security. In the test cases, TAKE shows high reliability (TPR > 99% and TNR = 100%) when error tolerance capability is determined in accordance with typical MRR and LER values. We discuss how the high reliability is achieved with more analysis on trajectory data and EET/HET values. Then we summarise how to choose EET/HET to improve reliability.

Reliability can be achieved if there exist some EET/HET values which can correctly distinguish trajectory samples (TrajSmp/BinSmp) of different vehicles. Technically, it means when the distances between trajectory samples of the same vehicle (intra-distance) do not overlap with the distances between trajectory samples of different vehicles (inter-distance), a value can be chosen between the maximum intra-distance and the minimum inter-distance. This value can be used as a threshold to correctly distinguish trajectory samples of different vehicles from those of the same vehicle.

Figure 10 shows the statistics of intra-distance and inter-distance of TrajSmmps and BinSmmps in Test Case 3. Intra-distance and inter-distance between TrajSmmps are measured with Euclidean distance per pair of Matched footprints, while those between BinSmmps are measured with Hamming distance per pair of verifier’s and claimant’s BinSmmps. The statistics show there is no overlap between intra-distance and inter-distance. This is consistent with the test results. TNR is 100%, because no overlap between intra-distance and inter-distance of BinSmmps means decoding will always fail and a TN result is always produced in a TN test. For better reliability, we can set the HET value to be slightly above the maximum intra-distance of BinSmp and EET value to be approximately maximum intra-distance of TrajSmp.

3) Improve Key Randomness

To evaluate the security level of the established keys, we have collected the keys generated in Baseline Case and used the NIST Entropy Assessment Tools to estimate the entropy of the keys. The results show that the keys have at least 0.93 bit of entropy per key bit. This means the established keys have nearly full entropy, i.e., each key bit is nearly uniformly distributed. Thus, the keys have enough randomness to defend against Brute-Force-Key-Guess, and SR2 is satisfied.

The full entropy is achieved for two reasons. First, the nonce used for deriving the key is generated with a pseudo-random number generator, which should achieve full entropy for each bit of the nonce. Considering the theoretical analysis results, the nonce should have a min-entropy of approximately KeyLen. Second, the nonce is derived into a key with a secure KDF. It amplifies the randomness of its input securely and provides full entropy for every bit of the derived key.

X. CONCLUSIONS

This paper has investigated how to achieve secure and efficient key establishment in dynamic IoTs. We have used the IoV as an example and built a typical use case for key establishment, V2IKE. We have investigated the use case to identify potential threats and specify desirable requirements. Then, based on our analysis on existing work, we have proposed a key establishment solution, TAKE, which does not require any prior trust or secrets and meet all the requirements. TAKE uses real-time trajectory data to extract temporary secrets between authorised entities efficiently. By using the temporary secrets, it establishes keys with variable security levels, and achieves confidentiality, integrity protection and mutual authentication.

TAKE is evaluated with both theoretical analysis and experiments. Evaluation results show that TAKE can establish
keys with satisfying security and efficiency in a typical use case scenario. However, the performance of TAKE depends on the trajectory data quality and the choice of parameter values. If the trajectory data collection systems malfunction, TAKE may not be able to achieve the expected performance. TAKE may be less efficient than some cryptography-based key establishment solutions, as decoding can be complex and time-consuming, and extra steps of sample generation and processing are needed. However, this is a price to pay to achieve secret establishment without using prior secrets, which is necessary in dynamic IoTs and difficult to achieve with cryptography-based solutions. In addition, TAKE may be insecure if attackers find ways to learn more about the trajectories of their target vehicles. Attackers may try to follow the target vehicles for a long period while using different attacker vehicles to avoid being spotted, and they may try to obtain more accurate locations of the target vehicles, by onboard sensing, or by requesting the target vehicles to share their real-time locations to support some IoV services such as collision avoidance.

Our future work includes using TAKE in security services, such as privacy-preserving authentication for dynamic IoTs, applying TAKE to other use cases of dynamic IoTs, such as vehicle-to-vehicle key establishment where no trusted authority is present, and integrating TAKE with other authentication methods to facilitate multi-factor authentication.

APPENDIX A PSEUDOCODE FOR ALGORITHMS

A. TAKE ALGORITHMS

Algorithm 1 SmpGen

```
procedure SmpGen(TrajData_v, l, LEP, A, Res)
    KeyLen ← l
    t_fpv ← 2 · LEP
    H_inst_fpv ← − log2((πt_fpv^2)/A)
    n_fpv ← ⌈H_inst_fpv/H_inτ_Smp⌉
    eb ← ⌈log2(t_fpv/Res + 1)⌉
    for i ← 1 to n_fpv do
        choose j randomly from {1, ..., len(TrajData_v)}
        (ts_j, x_j, y_j) ← TrajData_v[j]
        Append ts_j to TS
        Append (x_j, y_j) to TrajSmp_c
    end for
    return TrajSmp_c, TS, t_fpv, KeyLen, eb
end procedure
```

Algorithm 2 SmpRep

```
procedure SmpRep(TrajData_c, TS, MovData)
    for ts in TS do
        if j ← search(ts, TrajData_v) then
            (ts_j, x_j, y_j) ← TrajData_v[j]
            Append (x_j, y_j) to TrajSmp_c
        else
            Mismatch Correction
            Find all timestamps ts_l, ts_r in TrajData_c such that ts_l < ts < ts_r AND there is no other timestamp between ts_l and ts, AND no other timestamp between ts and ts_r
            (ts_l, x_l, y_l) ← TrajData_c[l]
            (ts_r, x_r, y_r) ← TrajData_c[r]
        end if
        Find all timestamps \{ts_a\} in MovData such that ts_l ≤ ts_a ≤ ts_r
        if \{ts_a\} is not empty then
            for each ts_a in \{ts_a\} do
                (v, acc, θ) ← MovData[a]
                if ts_a < ts then
                    dx ← v · (ts_a − ts_a - 1) + 1/2 · acc · θ
                    dy ← v · (ts_a − ts_a - 1) + 1/2 · acc · θ
                end if
            end for
            end if
        end for
        Append (x, y) to TrajSmp_c
    end for
    return TrajSmp_c
end procedure
```

Algorithm 3 SmpProc

```
procedure SmpProc(TrajSmp, tb, Res)
    for each (ts_l, x_l, y_l) in TrajSmp do
        if ts_l < tb then
            bx ← bin(x_l * 10^5/Res mod 2^50)
            by ← bin(y_l * 10^5/Res mod 2^50)
            for i = 1 to 256 do
                Append bx_i and by_i to BinSmp
            end for
        end if
    end for
    return BinSmp
end procedure
```

Algorithm 4 TrajKeyGen

```
procedure TrajKeyGen(BinSmp_v, tbs, TS, KeyLen)
    n ← len(BinSmp_v)
    d ← 2tbs + 1
    k ← n − d + 1
    C ← Encoder(n, k, d)
    nonce ← PRNG(k)
    s ← PRNG(k) is a PNRG that generates a string of k bits
    nonce ← C(nonce)
    ss ← c_v = C(nonce)
    csum ← h(KeyLen∥|ts_v∥|ts_a||ss∥TS∥|BinSmp_v)
    KT_v ← KD(nonce, KeyLen)
    return KT_v, ss, csum
```

''
Algorithm 5 TrajKeyRep

\begin{algorithm}
\caption{TrajKeyRep($BinSmp_c$, $t_{bs}$, $t_{fp}$, $ss$, $csum$, $KeyLen$)}
\begin{algorithmic}
\Procedure{TrajKeyRep}{$BinSmp_c$, $t_{bs}$, $t_{fp}$, $ss$, $csum$, $KeyLen$}
\State $n \gets \text{len}(BinSmp_c)$
\State $d \gets 2t_{bs} + 1$
\State $k \gets n - d + 1$
\State $C \gets \text{Encoder}(n, k, d)$
\State $D \gets \text{Decoder}(n, k, d)$
\State $c_c \gets ss \oplus BinSmp_c$
\State $nonce \gets D(c_c)$
\State $c_v \gets C$ (nonce)
\State $S \gets c_v \oplus ss$
\State $\{errloc\} \gets \text{locations of different bits in } S$ and $BinSmp_c$
\If{$\text{len}(\{errloc\}) > t_{bs}$}
\State return \texttt{⊥}
\Else
\For{each $loc$ in $\{errloc\}$}
\State $fp_{bs} \gets S[[loc/\text{tb}] \times \text{tb} : [loc/\text{tb}] \times \text{tb} + \text{tb}]$
\State $fp_{br} \gets BinSmp_c[[loc/\text{tb}] \times \text{tb} : [loc/\text{tb}] \times \text{tb} + \text{tb}]$
\State $fp_s \gets \text{dec}(fp_{bs}) \triangleright \text{dec}[\cdot]$ transforms a binary string in to its decimal format
\State $fp_r \gets \text{dec}(fp_{br})$
\If{$|fp_r - fp_r| > t_{fp}$}
\State return \texttt{⊥}
\EndIf
\EndFor
\EndIf
\If{$h(KeyLen)[t_{fp}]t_{bs}[ss][TS][S] \neq csum$}
\State return \texttt{⊥}
\EndIf
\State $KT_c \gets \text{KDF}$ (nonce, $KeyLen$)
\State return $KT_c$
\EndProcedure
\end{algorithmic}
\end{algorithm}

B. DATASET GENERATION ALGORITHM: DATAGEN

TrajData and MovData used in the experiments are generated with Algorithm 6 using simulation data as input, where $LS = \{(x_i, y_i)\}$ is the location point sequence and $TSS = \{ts_i\}$ is the timestamp sequence, $MS = \{v_i, a_i, ang_i\}$ are the sequence of speed, acceleration and heading tuples, $smpr$ is the sampling rate.

Algorithm 6 DataGen

\begin{algorithm}
\caption{DataGen($LS$, $TSS$, $MS$, $MRR$, $LER$, $smpr$)}
\begin{algorithmic}
\Procedure{DataGen}{$LS$, $TSS$, $MS$, $MRR$, $LER$, $smpr$}
\State $n \gets \text{len}(\{ts_i\})$
\State $er \gets 6378137$
\For{i in n}
\If{$(ts_i \times smpr/10)$ is integer}
\If{$\text{rand}(0, 1) \geq MRR$}
\State $fper = \text{rand}(0, 1) \times LER$
\State $fp\alpha = \text{rand}(0, 1) \times \pi \times 2$
end if
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

APPENDIX B SECURITY ANALYSIS PROOFS

A. BRUTE-FORCE-TRAJSMP-GUESS

For clarity and without losing generality, we assume attackers try to impersonate authorised entity $A$ to authorised entity $B$. As claims the instance, and $B$’s TrajSmp and BinSmp are denoted as $TrajSmp_B$ and $BinSmp_B$. To succeed in BFTG, the attackers must guess the location in each footprint in the TrajSmp. We call this guess Brute-force-footprint-guess (BFFG). For BFFG to succeed, the guessed location must have a Euclidean distance of no more than $t_{fp}$ from the location in the Matched Footprint in $TrajSmp_B$. The probability for BFFG to succeed is

$$Pr[BFFG = \text{succ}] = \frac{\pi \cdot t_{fp}^2}{4 \pi R^2}, \quad (33)$$

where $R$ is the radius of Earth.

For BFTG on a TrajSmp consisting of $n_{fp}$ footprints to succeed, the attackers must succeed in $n_{fp}$ BFFG. Therefore, the probability for BFTG to succeed is

$$Pr[BFTG = \text{succ}] = \left(\frac{\pi \cdot t_{fp}^2}{4 \pi R^2}\right)^{n_{fp}}, \quad (34)$$

B. BRUTE-FORCE-BINSMMP-GUESS

To succeed in BFBG, the attackers must guess $n - t_{bs}$ out of the $n = 2 \cdot n_{fp} \cdot \text{tb}$ bits of the $BinSmp_B$ correctly. That is,

$$Pr[BFBG = \text{succ}] = 2^{-(n-t_{bs})}, \quad (35)$$

Using (13)(14)(15)(16)(26), we have

$$Pr[BFBG = \text{succ}] = 2^{-(n-t_{bs})} \approx 2^{-n_{fp}(2\text{th} - \text{eb})}, \quad (36)$$

where the approximation is due to ignoring the ceiling function. Then from (13) and (14), we have

$$2^{-KeyLen} \approx 2^{n_{fp} \log_2((\pi t_{fp}^2)/A)}. \quad (37)$$

We compare $2^{-KeyLen}$ with $Pr[BFBG = \text{succ}]$.

$$Pr[BFBG = \text{succ}] = \frac{2^{-n_{fp}(2\text{th} - \text{eb})}}{2^{2n_{fp} \log_2((\pi t_{fp}^2)/A)}} \quad (38)$$
Use logarithm on the right side,
\[
\log_2 \frac{2^{-n_{fp}(2tb - eb)}}{2^{n_{fp}(2tb - eb) - \log_2(\pi t_{fp}^2)/A}} = \log_2 \frac{2^{-n_{fp}(2tb - eb) - \log_2(\pi t_{fp}^2)/A}}{2^{n_{fp}(2tb - eb) - \log_2(\pi t_{fp}^2)/A}} = -n_{fp}(2tb - eb) - (n_{fp} \cdot \log_2((\pi t_{fp}^2)/A)) = -n_{fp}(2tb - eb + \log_2((\pi t_{fp}^2)/A)) = -n_{fp}(2tb - eb + \log_2(\pi + 2\log_2 t_{fp} - \log_2 A) \approx -n_{fp}(eb + \log_2 \pi).
\]

Therefore,
\[
Pr[BFBG = succ] \approx 2^{-KeyLen} \cdot 2^{-n_{fp}(eb+\log_2 \pi)}. \tag{39}
\]

This proves the probability of guessing a Valid BinSmp with Brute-Force Guess is approximately \(2^{-KeyLen} \cdot 2^{-n_{fp}(eb+\log_2 \pi)}\), which is smaller than using Brute-Force Guess to directly attack the key. In the cases where \(t_{bs}\) is relaxed, that is, \(t_{bs} \geq n_{fp} \cdot eb\), the probability can be treated as if there is an additional constant value \(cv\) and
\[
Pr[BFBG = succ] \approx 2^{-KeyLen} \cdot 2^{-n_{fp}(eb+log_2 \pi - cv)}.
\]

The equality holds when \(t_{bs}\) is set without relaxation, i.e., \(t_{bs} = n_{fp} \cdot eb\). When \(eb < log_2 \pi\), \(H_{\infty}(BS|SS)\) is greater than \(KeyLen\). As \(eb\) is set as integer, this means when \(eb \geq 2\) the likelihood is more than \(2^{-KeyLen}\).

For \(m = H_{\infty}(BS) = KeyLen\),
\[
H_{\infty}(BS|SS) = KeyLen - 2t_{bs}. \tag{44}
\]

This means the likelihood that attackers could guess BinSmp with the knowledge of the Secure Sketch but no knowledge of True Trajectory is less than \(2^{-KeyLen}\). When \(eb \geq 2\), the likelihood is more than \(2^{-KeyLen}\).

\section*{C. EAVESDROPPING-AND-GUESS}

For EAG to succeed, attackers guess footprints in the Footprint Sample Area, \(A\), and all their guesses should be within \(t_{fp}\) to the locations in Matched Footprints in \(TrajSmp_B\). The probability of EAG to succeed is
\[
Pr[EAG = succ] = (\frac{\pi \cdot t_{fp}^2}{A})^{n_{fp}}. \tag{40}
\]

Using (13) and (14), and applying logarithm on both sides, we have
\[
\log_2 Pr[EAG = succ] = n_{fp} \cdot \log_2(\frac{\pi \cdot t_{fp}^2}{A}) \leq KeyLen - \log_2(\frac{\pi \cdot t_{fp}^2}{A}) = -KeyLen. \tag{41}
\]

Therefore,
\[
Pr[EAG = succ] \leq 2^{-KeyLen}. \tag{42}
\]

\section*{D. GUESS-BY-SKETCH}

We treat a BinSmp as a sample value from a distribution \(BS\), and \(ss\) as a sample value from a distribution \(SS\).

We denote the average min-entropy of \(BS\) given \(SS\) as \(H_{\infty}(BS|SS)\). As we adopt the code-offset secure sketch construction, using conclusion given in [38], we have that the average min-entropy of \(BS\) after publishing the Secure Sketch \(ss\) generated on BinSmp is \(H_{\infty}(BS|SS) = m + k - n \leq m - 2t_{bs}\), where \(m\) is the min-entropy of \(BS\). When \(BS\) is a uniform distribution, the min-entropy of \(BS\) is \(H_{\infty}(BS) = KeyLen + n_{fp}(eb + \log_2 \pi)\) when the attackers do not eavesdrop, and \(H_{\infty}(BS) = KeyLen\) in the worst case of EAG attacks. We discuss the average min-entropy with these two min-entropy values respectively.

For \(m = H_{\infty}(BS) = KeyLen\) + \(n_{fp}(eb + \log_2 \pi)\),
\[
H_{\infty}(BS|SS) = KeyLen + n_{fp} \cdot eb + n_{fp} \cdot \log_2 \pi - 2t_{bs} \leq KeyLen + n_{fp} \cdot \log_2 \pi - t_{bs}. \tag{43}
\]

The equality holds when \(t_{bs}\) is set without relaxation, i.e., \(t_{bs} = n_{fp} \cdot eb\). When \(eb < log_2 \pi\), \(H_{\infty}(BS|SS)\) is greater than \(KeyLen\). As \(eb\) is set as integer, this means when \(eb \geq 2\) and \(t_{bs}\) is set without relaxation, the likelihood (maximum probability) that attackers could guess BinSmp value with the knowledge of the Secure Sketch but no knowledge of True Trajectory is less than \(2^{-KeyLen}\). When \(eb \geq 2\), the likelihood is more than \(2^{-KeyLen}\).

For \(m = H_{\infty}(BS) = KeyLen\),
\[
H_{\infty}(BS|SS) = KeyLen - 2t_{bs}. \tag{44}
\]

This means the likelihood that attackers could guess BinSmp with the knowledge of the Secure Sketch generated on BinSmp and the Footprint Sample Area all footprints are chosen from is \(2^{-(KeyLen-2t_{bs})}\), which is higher than \(2^{-KeyLen}\).

\section*{E. BRUTE-FORCE-KEY-GUESS}

The attackers may guess the key and the TrajSmp in an instance by guessing the nonce of the instance correctly. The attackers take the following steps. First, they choose a random guess for the nonce. Second, they compute the codeword by encoding the nonce. Third, they use the Secure Sketch and the codeword to reveal the BinSmp. Finally, they generate a hash with the revealed BinSmp using (28) and compare the hash value with the Checksum. If the two are equal, the attackers can verify that the guess is correct, and the key and the TrajSmp are revealed.

Provided that the PRNG for generating the nonce is secure, the probability of successfully guessing the nonce with BFKG is
\[
Pr[BFKG = succ] = 2^{-k} = 2^{-(n - 2t_{bs})} \geq 2^{-2n_{fp}(tb - eb)} \approx 2^{-KeyLen}. \tag{45}
\]

This means if the HET value, \(t_{bs}\), is set without relaxation, the probability for BFKG to succeed is approximately \(2^{-KeyLen}\).

\section*{References}


This article has been accepted for publication in IEEE Access. This is the author’s version which has not been fully edited and content may change prior to final publication. Citation information: DOI: 10.1109/ACCESS.2022.3215688


M. Li et al.: Trajectory-based Authenticated Key Establishment for Dynamic Internet of Things