# Time-Efficient Identifying Key Tag Distribution in Large-Scale RFID Systems

Yanyan Wang, *Member, IEEE*, Jia Liu, *Member, IEEE*, Zhihao Qu, *Member, IEEE,* Shen-Huan Lyu, *Member, IEEE,* Bin Tang, *Member, IEEE*, and Baoliu Ye, *Member, IEEE*

***Abstract*—With the proliferation of RFID-enabled applications, large-scale RFID systems often require multiple readers to ensure full coverage of numerous tags. In such systems, we sometimes pay more attention to a subset of tags instead of all, which are called key tags. This paper studies an under-investigated problem *key tag distribution identification*, which aims to identify which key tags are beneath which readers. This is crucial for efficiently managing specific items of interest, which can quickly pinpoint key tags and help RFID readers covering these tags collaborate to improve the tag inventory efficiency. We propose a protocol called Kadept that identifies the key tag distribution by designing a sophisticated Cuckoo filter that teases out key tags as well as assigns each of them a singleton slot for response. With this design, a great number of trivial (non-key) tags will keep silent and free up bandwidth resources for key tags, and each key tag is sorted in a collision-free way and can be identified with only 1-bit response, which significantly improves the time efficiency. To enhance the scalability and efficiency of Kadept for high key tag proportions, we propose E-Kadept protocol, which accelerates the identification process by designing an incremental Cuckoo filter that reduces false positives and improves space efficiency. We theoretically analyze how can we optimize protocol parameters of Kadept and E-Kadept, and conduct extensive simulations under different tag distribution scenarios. Compared with the state-of-the-art, E-Kadept can improve the time efficiency by a factor of 1.75×, when the ratio of key tags to all tags is 0.3.**

***Index Terms*—RFID, key tag distribution identification, time-efficient.**

## I. Introduction

Radio Frequency IDentification (RFID) has brought great benefits to a variety of applications, including but not limited to warehouse management [2]–[8], human-machine interface [9]–[13], object tracking and authentication [14]–[21]. In these applications, each object is attached with an RFID tag that has a unique ID to indicate the tagged object. The tag communicates with an RFID reader through backscatter. Since the communication distance between a reader and a tag is limited to a few meters, with the proliferation of RFID-enabled applications, multiple readers must be used to ensure the full coverage of desired inventory zones.

In a large-scale RFID system with multiple readers, we sometimes selectively focus on a subset of tags, referred to as key tags, for monitoring or management. For example, in a shopping mall with tens of thousands of goods, the staff might need to conduct an inventory over a specific category of goods, where the relevant tags are treated as key tags and the left are trivial tags out of interest. In another example, given a limited bandwidth resource, the shop owner would like to pay more attention to valuable goods (key tags) rather than cheap ones in a retail store [22]; a frequent check over key tags is necessary. In such cases, fast identifying key tags' distribution allows only readers that cover key tags to zoom into the corresponding regions to communicate with them, ensuring efficient key tag management. For clarity, we present a toy example shown in Fig. 1. If the distribution of key tags $k_{1-8}$ is known, we can schedule readers $R_1$ and $R_3$ to work concurrently, with each of them handling only the key tags that it covers. However, if we do not have this distribution knowledge, all readers $R_{1-4}$ need to work in a round-robin scheduling, with each reader querying key tags multiple times.

Yanyan Wang, Zhihao Qu, Shen-Huan Lyu, and Bin Tang are with the Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing 211100, China. Email: {yanyan.wang, quzhihao, lvsh, cstb}@hhu.edu.cn. Jia Liu and Baoliu Ye are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {jialiu, yebl}@nju.edu.cn.

A preliminary version [1] of this paper appeared in IEEE/ACM IWQoS'24.

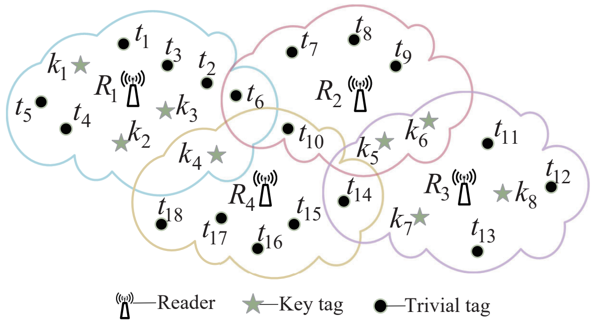


Fig. 1: The system model of key tag distribution.

In this paper, we study the problem of key tag distribution identification that quickly explores which key tags are beneath which reader's coverage, with the knowledge of all tag IDs a priori. This information plays a crucial role in various inventory operations. For example, it can help detect missing key tags [23] without the assumption that each reader knows its local tag distribution. It can also facilitate key tags information collection [24] by appropriately scheduling multiple readers that cover key tags.

An intuitive solution to this problem is to identify the distribution of all tags by running the tag identification protocols [25], [26] or the tag distribution identification protocol IB [27]. However, these approaches suffer from long time delays because they require all tags (including a great number of trivial tags) to transmit data, which is a waste of bandwidth resources. Besides global identification, tag polling is a more efficient solution: each reader just in turn broadcasts each key tag's ID to activate the key tag (if exist), which avoids the tag competition caused by trivial tags. However, this solution requires transmitting key tags' IDs and breaches the privacy, which is forbidden in some privacy-sensitive RFID systems [28], [29]. The IPP protocol, proposed in [30], is an advanced polling protocol that achieves remarkable efficiency by reducing the polling vector from 96 bits to only 1.6 bits. However, in our problem, each reader still needs to poll all tags to identify key tags. The state-of-the-art unknown tag identification protocol EUTI [31] can be adapted for our problem by first deactivating trivial tags and then collecting key tags. Nevertheless, it still incurs additional time overhead to resolve collisions among key tags after trivial tag elimination.

In light of this, we propose a key tag distribution identification protocol (*Kadept*), which targets at quickly identifying the distribution of key tags, without transmitting any tag IDs. Kadept consists of four operations: 1) Filtering. It constructs a Cuckoo filter to tease out key tags, avoiding interference from trivial tags. 2) Assigning. It assigns a unique, continuous slot for each key tag to make key tags respond to the reader's query without time waste. 3) Separating. It leverages flag technology to let each tag know whether it is a key tag or not so that key tags can be appropriately managed in subsequent inventory operations. 4) Identifying. It identifies key tag distribution: non-conflicting tags are identified in parallel, while conflicting tags are scheduled collision-free.

While Kadept efficiently identifies key tag distribution under low key tag densities, its performance degrades with increasing key tag sizes, potentially limiting its ability to provide real-time support for upper-level applications. Therefore, we propose an enhanced version of Kadept, called E-Kadept. In E-Kadept, we design an incremental Cuckoo filter with a lower false positive rate and higher space efficiency compared to the Cuckoo filter, enabling E-Kadept to maintain Kadept's core functionalities while further improving time efficiency.

We theoretically analyze how can we optimize the protocol parameters of Kadept and E-Kadept, and evaluate their performance via extensive simulations. The results show that both our protocols are far superior to the baseline IB [27], the polling protocol IPP [30], and an optimized version of the state-of-the-art unknown tag identification protocol EUTI [31]. For instance, in a multi-reader RFID system with 10,000 tags (7000 trival tags and 3000 key tags), E-Kadept and Kadept reduce the execution time of EUTI from 14s to 5.1s and 7.6s, respectively, producing 1.75× and 0.84× performance gains.

The remainder of this paper is organized as follows. Section II gives the preliminary. Section III details the Kadept protocol. Section IV presents the theoretical performance analysis of Kadept. Section V describes the E-Kadept protocol. Section VI analyzes the optimal parameters of E-Kadept. Section VII conducts extensive simulations to evaluate the performance of Kadept and E-Kadept. Section VIII briefly introduces the related work. Finally, Section IX concludes this paper.

## II. Preliminary

### A. System Model

We consider a large-scale RFID system comprising a back-end server, multiple readers, and a great number of tags. Each tag has a unique ID that exclusively indicates the item it is attaches to. The tags use backscattering to communicate with readers, limiting the communication distance between a reader and a tag to a few meters. To cover all tags in the surveillance regions, multiple readers are deployed with many overlapped interrogation regions and interference regions. Each reader covers a portion of the tags, while each tag may be covered by one or more readers. The server maintains an item list of all tag IDs but has no information about the tag distribution.

The communication between the readers and tags follows *Reader Talks First* mode as defined in C1G2 standard [32]. In this mode, one reader initiates a query with a continuous RF waveform (CW), energizing the tags it covers. The tags then transmit their information by reflecting the modulated CW back to the reader. We use $t_{ID}$ and $t_s$ to represent the lengths of two types of time slots: the former is for transmitting tag ID, and the latter is for transmitting 1-bit information. Similar to [33], the readers broadcast $\text{ACK}_l$ to label tags. The time for transmitting an ACK is referred to as $t_{ack}$.

### B. Problem Definition

Consider a multi-reader RFID system. The set of readers in the system is denoted as $\mathcal{R} = \{R_1, \dots, R_m\}$. The topology of readers can be depicted as a graph $\mathcal{G} = (\mathcal{R}, \mathcal{E})$, where $\mathcal{E}$ is the set of edges each connecting two readers which have overlapped monitor areas; these two readers are neighbors. For reader $R_i(R_i \in \mathcal{R}, 1 \le i \le m)$, its neighbor and non-neighbor reader sets are denoted as $\Gamma(R_i)$ and $\Upsilon(R_i)$, respectively. The tags are divided into two sets: the key tag set $K = \{k_1, \dots, k_x\}$ and the trivial tag set $T = \{t_1, \dots, t_y\}$. Symbol $K_i$ represents the set of key tags under the coverage region of $R_i$. Since each contentious key tag in the overlapped region is covered by two or more readers, we have $\bigcup_{i=1}^{m} K_i = K$ and $|K_i \bigcap K_j| \ge 0$ $(i \ne j)$. With the knowledge of reader topology and tag IDs, this paper is to identify which key tags are beneath which readers, i.e., identifying $K_i$ for $R_i$.

### C. Challenges and Design Guideline

To rapidly determine which key tags within a reader's coverage, it is essential to eliminate the interference of trivial tags and collect the responses of key tags in a time-efficient manner. The key challenges of key tag distribution identification are: 1) eliminating trivial tags efficiently, and 2) collecting key tag responses effectively.

To address these challenges, our design focuses on isolating key tags from a large RFID tag population and collect their distribution across multiple readers with minimal latency and memory use. Cuckoo filters are well suited for this task due to their compact fingerprint-based representation: each key tag's ID is hashed into a small fingerprint, which is then inserted into one of a small number of candidate buckets, maintaining a filter size approximately proportional to the number of key tags (e.g., ∼1000 buckets for 1000 key tags in our simulations). This near-one-to-one mapping ensures efficient use of resources while assigning slots for key tag identification. However, due to its probabilistic nature, standard Cuckoo filters may suffer from fingerprint collisions, making one-to-one slot assignments unreliable. To overcome this, we propose a new construction scheme that filters out trivial tags while assigning each key tag a unique slot within a continuous slot range, ensuring sequential responses.

In contrast, Bloom filter variants, such as Counting Bloom Filters and Cascade Filters, rely on multiple hash functions to reduce false positives, necessitating a larger number of slots (e.g., several times the number of key tags). This leads to many empty slots, increasing memory overhead and reducing efficiency in resource-constrained RFID readers. Cascade Filters, optimized for hierarchical filtering, require complex multi-level structures that are less suited for our single-pass design focused on immediate slot assignment.

Quotient Filters, like Cuckoo filters, use fingerprints for compact storage, but they require additional metadata to manage collisions, increasing storage demands. Moreover, their insertion process is more complex, involving slot shifting that adds computational overhead, whereas Cuckoo filters offer a simpler, faster approach tailored to our assigning operation's need for quick slot allocation.

In summary, while Counting Bloom Filters, Cascade Filters, and Quotient Filters could potentially be adapted for key tag distribution identification, Cuckoo filters strike an optimal balance of memory efficiency, time efficiency, and simplicity for our protocols.

TABLE I: Key Notations

| Symbol | Descriptions |
|---|---|
| $\mathcal{R}$ | the set of readers |
| $K$ | the set of key tags |
| $T$ | the set of trivial tags |
| $m$ | the number of readers |
| $x$ | the number of key tags |
| $y$ | the number of trivial tags |
| $h$ | the number of hash functions |
| $h_i$ | the $i$th hash function |
| $F_j$ | the fingerprint of the key tag $k_j$ |
| $d$ | the fingerprint length |
| $\Delta F$ | the difference between consecutive fingerprints |
| $\Delta d$ | the bit length of the maximum fingerprint difference |
| $\alpha$ | the given negative indicator threshold |
| $\theta$ | the number of identified key tags per unit time |

## III. Kadept Protocol

### A. Overview

Kadept aims to fast identify key tag distribution by simultaneously eliminating trivial tag interference and assigning a unique transmission slot to each key tag without extra cost. To achieve this, Kadept constructs a specialized Cuckoo filter that not only filters trivial tags but also assigns each inserted key tag a unique slot. Due to false positives in Cuckoo filters, some trivial tags may pass the filter and respond to a reader alongside key tags, leading to a failure in identifying the distribution of these collided key tags. To ensure complete identification, Kadept implements an iterative multi-round execution, as shown in Fig. 2. Each round consists of three phases: *constructing phase* that constructs a Cuckoo filter using key tags; *filtering-assigning-separating phase* that utilizes the Cuckoo filter to filter trivial tags, assign each key tag a unique and continuous slot, and separate key tags from trivial tags; *identifying phase* that determines the readers covering each key tag and the key tags covered by each reader by checking the assigned slot statuses. In Kadept, if a reader is sure about whether a key tag is under its coverage or not, we say that this key tag is identified by the reader, or unidentified otherwise. After all readers identify their respective key tags, they exchange and consolidate their results to determine the global key tag distribution. Details are given below.

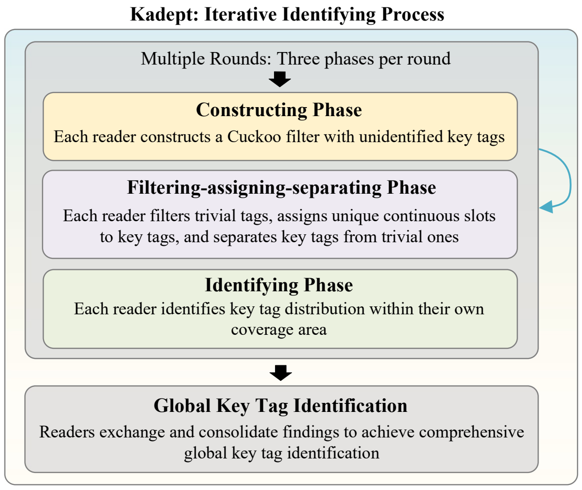

Kadept: Iterative Identifying Process
Multiple Rounds: Three phases per round
Constructing Phase
Each reader constructs a Cuckoo filter with unidentified key tags
Filtering-assigning-separating Phase
Each reader filters trivial tags, assigns unique continuous slots to key tags, and separates key tags from trivial ones
Identifying Phase
Each reader identifies key tag distribution within their own coverage area
Global Key Tag Identification
Readers exchange and consolidate findings to achieve comprehensive global key tag identification

Fig. 2: Overview of the Kadept Protocol.

### B. Constructing Phase

Consider an arbitrary reader $R_i$. Initially, $R_i$ has no knowledge of its local key tags and thus uses all key tags to construct a Cuckoo filter, denoted by $\mathcal{CF}$. The basic unit of $\mathcal{CF}$ is called a bucket, and $\mathcal{CF}$ consists of an array of buckets. $\mathcal{CF}$ stores key tags' fingerprints to reduce the filter size as well as avoid leaking key tag IDs. In the construction process, however, $\mathcal{CF}$ temporarily stores tag IDs, to restore and rehash an original key tag in a bucket to find its alternate location. Next, we design a new construction scheme to extend $\mathcal{CF}$ beyond filtering to perform slot assignment operations. By doing this, Kadept ensures the inserted key tags can respond to the reader one after another without empty or collision slots.

For a key tag $k_j (k_j \in K, 1 \leq j \leq x)$, the reader calculates its constant-sized fingerprint $F_j$ and the indexes $H_j^{\hbar} (1 \leq \hbar \leq h)$ of the $h$ candidate buckets as follows:

$$\begin{aligned} F_j &= H(ID_{k_j}, s_f) \mod 2^d \\ H_j^{\hbar} &= H(ID_{k_j}, s_{\hbar}) \mod f_{\mathcal{CF}}, \end{aligned} \tag{1}$$

where $H(\cdot)$ is hash function. Since each tag actually holds one hash function, which is shared by readers and tags, the reader broadcasts $h$ hash seeds to tags to mimic $h$ hash functions. $s_f$ and $s_{\hbar} (1 \leq \hbar \leq h)$ are the hash seeds used to calculate the fingerprints and candidate buckets, respectively; $f_{\mathcal{CF}}$ and $d$ are the lengths of $\mathcal{CF}$ and its fingerprint, respectively. Let $\mathcal{H}_j$ be the candidate bucket set of $k_j$, and we have $\mathcal{H}_j = \{H_j^1, \cdots, H_j^h\}$. Before inserting $k_j$ into $\mathcal{CF}$, we classify the fingerprints of all key tags and then insert $k_j$ following this classification. According to Eq. (1), the fingerprints in the set $\{0, \dots, 2^d\}$ can be divided into three categories: *useless fingerprints* mapped by no key tags, *singleton fingerprints* mapped by exactly one key tag, and *collision fingerprints* mapped by more than one key tag. We refer to the key tags with the same fingerprint as the same group. The reader $R_i$ inserts $k_j$ into $\mathcal{CF}$ in terms of the status of $F_j$, and the specific steps are as follows.

$F_j$ **is a singleton fingerprint.** If only some candidate $h$ buckets are assigned tags, $R_i$ randomly selects an empty bucket and inserts $k_j$ by putting $ID_{k_j}$ into it. Otherwise, $R_i$ picks one of the $h$ buckets at random and inserts $k_j$ into it, replacing a key tag previously in this bucket.

$F_j$ **is a collision fingerprint.** As described above, key tags with the same fingerprint belong to the same group. With the $h$ hash functions, $R_i$ can calculate all possible buckets that are selected by the other tags except $k_j$ from the same group. Let $\mathcal{B}_j$ represent the set of these possible buckets. If $\mathcal{H}_j \subseteq \mathcal{B}_j$, the insertion failures, then we will reconstruct $\mathcal{CF}$. Otherwise, $R_i$ removes the same bucket(s) in the two sets from $\mathcal{H}_j$, and the candidate bucket set of $k_j$ changes to $\mathcal{H}_j - \mathcal{B}_j$; where $1 \leq |\mathcal{H}_j - \mathcal{B}_j| \leq h$ and $|\cdot|$ represents the number of items in a set. Then $R_i$ inserts $k_j$ into one of its candidate buckets according to the insertion steps described when $F_j$ is a singleton fingerprint.

For the previously existing key tag that makes room for the new one, the reader uses the same insertion steps to relocate it. The reinsertion process will continue until the last key tag that is kicked out successfully maps to an empty bucket within the given maximum loops or fails to insert into $\mathcal{CF}$. If the last key tag to be inserted can map to an empty bucket within the given maximum loops, the reader puts each key tag's fingerprint into its bucket to replace of ID. Otherwise, we update $f_{\mathcal{CF}}$ to $f_{\mathcal{CF}} + n_l$ and reconstruct $\mathcal{CF}$ using the new filter size, where $n_l$ is the number of tags that fail to insert $\mathcal{CF}$.

For empty buckets, the reader fills them with a useless fingerprint. If none are available, the reader inserts a singleton fingerprint that does not belong to any key tag mapping to this bucket. For example, if the second bucket of $\mathcal{CF}$ is empty, it is filled with a singleton fingerprint that does not belong to any key tag with a hash value of 2, which also maps to this bucket. This strategy ensures that each key tag is inserted into exactly one of its candidate buckets, enabling a unique slot assignment for each key tag. So far, $\mathcal{CF}$ is successfully constructed, and its optimal parameters will be analyzed in Section IV.

Our proposed protocol, Kadept, shines in that it is able to filter interference tags and assign each key tag a unique and continuous slot simultaneously by designing a new construction scheme for the Cuckoo filter. However, traditional filters, such as the widely used Bloom filer, only function as a filter. After filtering, the reader has to issue a series of slotted frames to check each key tag's response in its randomly selected slot, which retards the identification process caused by useless (empty and collided) slots. Although collision resolution methods can mitigate this, they require additional processing time that our approach avoids.

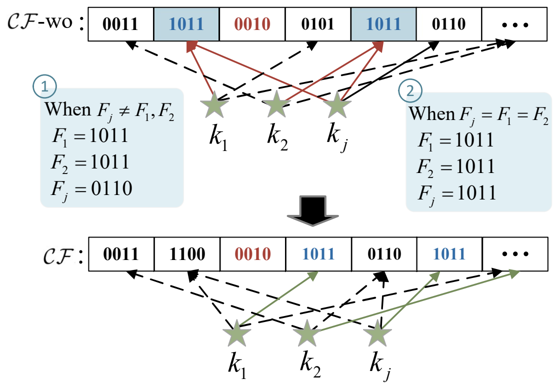


Fig. 3: The construction scheme with and without considering the interferences of collision fingerprints.

## C. Filtering-assigning-separating Phase

**Filtering.** After getting $\mathcal{CF}$, the reader broadcasts it and the construction parameters $\langle f_{\mathcal{CF}}, s_{1-h}, d, s_f \rangle$, and the useless fingerprint $F_u$ (if it exists) to tags. Upon receiving this request, each tag first calculates its fingerprint and $h$ candidate buckets according to Eq. (1). Then these $h$ buckets are read: if the fingerprint in only one bucket matches, the tag passes $\mathcal{CF}$ and keeps active; otherwise, it keeps silent.

**Assigning.** Since the Cuckoo filter has no false negatives, all key tags will keep active. Consider the key tag $k_j$. Let $SI_j$ be the slot index of $k_j$. We denote the number of buckets that store useful fingerprints preceding the matched bucket of $k_j$ as $b_j$. According to $\mathcal{CF}$, $k_j$ can get $b_j$, and set $SI_j = b_j + 1$. Note that each active tag only has one slot index because each key tag has one matched bucket; the false positive trivial tags that map to more than one matched bucket will keep silent.

**Separating.** To achieve the precise management of key tags without the interference of trivial tags, Kadept separates key tags from trivial tags by setting a flag for each tag. Specifically, Kadept lets each tag keep a flag with a value that is initially set to 0. A tag that cannot pass $\mathcal{CF}$ updates its flag to 1 before keeping silent, indicating that it is a trivial tag. After that, the reader broadcasts the flag 0 and then queries tags; only key tags will respond to the reader when the flag is 0.

Fig. 3 illustrates how collision fingerprints affect $k_j$'s slot assignment. $\mathcal{CF}$-wo is constructed without considering the interference of collision fingerprints, that is, all key tags are inserted into $\mathcal{CF}$-wo in the same way as key tags with singleton fingerprints. Additionally, the useless fingerprint 0010 is used to fill empty buckets. When $F_j$ is the singleton fingerprint 0110, after receiving $\mathcal{CF}$-wo, $k_j$ can find its matched bucket and get $b_j = 4$. Then it sets its slot index to $SI_j = b_j + 1 = 5$. However, when $F_j = F_1 = F_2$ is the collision fingerprint 1011, and $k_1$, $k_2$ map to two candidate buckets of $k_j$, $k_j$ is unsure whether $SI_j$ should be set to 2 or 5. Even if $k_j$ responds in either the second, the fifth, or both slots, the reader cannot identify which key tag caused the response when two or more key tags respond in the same slot. In contrast, in $\mathcal{CF}$, $k_j$ is inserted into one of its candidate buckets that are not mapped by the other tags from its group. The proposed construction scheme enables $\mathcal{CF}$ to assign a unique slot for each inserted tag in addition to filtering interference tags.

## D. *Identifying Phase*

This phase describes how readers identify noncontentious key tags in parallel first and then identify contentious key tags based on a schedule.

*1) Identifying Noncontentious Key Tags:* In practice, overlapped zones are usually smaller than interrogation zones, and the majority of tags are noncontentious tags. To accelerate the identification process, readers first concurrently identify noncontentious key tags; only noncontentious tags can listen to readers and participate in this step due to reader collisions.

Consider the reader $R_i$. It identifies the noncontentious key tags that it covers by constructing two arrays $E_i$ and $A_i$ of length $l_i$. The first array $E_i$ is constructed by the expected responses from the tags that pass $\mathcal{CF}$. From the filtering step in III-C, we know that the key tags used to construct $\mathcal{CF}$ and the active false positive trivial tags will pass $\mathcal{CF}$. According to tags' responses, the slots can be divided into three categories: *empty slots* picked by no tags, *singleton slots* picked by one tag, and *collision slots* picked by two or more tags. They are symbolized by '0', '1', and 'c', respectively. Since $R_i$ has prior knowledge of all active tags, it can predict the expected slot category information of tags that pass $\mathcal{CF}$, and accordingly generates $E_i$: each element in $E_i$ records each slot's expected category information. Note that $R_i$ virtually constructs $E_i$ without taking any communication overhead.

The second array $A_i$ is built on tags' actual responses. Since each tag gives a 1-bit response, the actual slots follow into two categories: *empty slots* picked by no tags, *non-empty slots* picked by one or more tags. They are symbolized by '0' and '1', respectively. Recall that each tag that passes $\mathcal{CF}$ gets its slot index and keeps active. Then, $R_i$ constructs $A_i$: each element in $A_i$ records each slot's actual category information.

Comparing $E_i$ and $A_i$, $R_i$ can identify whether a tag is or not its local tag. Two slots with the same index in $E_i$ and $A_i$ are called a slot pair. For a slot pair $\langle a_1, a_2\rangle$, $a_1$ and $a_2$ denote the slot categories in $E_i$ and $A_i$, respectively. By checking each slot pair, $R_i$ updates three tag sets that it stores: the local noncontentious key tag set $K_i^{L_n}$, the unidentified noncontentious key tag set $K_i^{U_n}$, and the active noncontentious trivial tag set $T_i^{A_n}$. These sets initially contain zero key tags, all key tags, and all false positive trivial tags, respectively. By analyzing $l_i$ slot pairs, $R_i$ acts as follows.

- $\langle 1, 1\rangle$. This slot pair indicates that the key tag that is expectedly in this slot replies. The reader sends ACK$_l$ to label it, puts it into $K_i^{L_n}$, and removes it from $K_i^{U_n}$. The labeled key tags keep silent after this round.
- $\langle 1, 0\rangle$ and $\langle c, 0\rangle$. These two kinds of slot pairs can be used to identify the noncontentious key tags that are not in the coverage regions of $R_i$. $R_i$ removes the corresponding key tags from $K_i^{U_n}$ and the trivial tags from $T_i^{A_n}$.
- $\langle c, 1\rangle$. This slot pair cannot be served as the vehicle to identify key tags. Because the reader $R_i$ cannot infer the response(s) comes from which tag(s).

After the first round, most of the noncandidate key tags and the false positive trivial tags are removed from $K_i^{U_n}$ and $T_i^{A_n}$, respectively. With the updated $K_i^{L_n}$, $K_i^{U_n}$, and $T_i^{A_n}$, $R_i$ executes the above three phases to identify the unidentified key tags. The process is repeated for several rounds and finishes when there are no $\langle c, 1\rangle$ slot pairs in one round. After that, the noncontentious key tags are all identified.

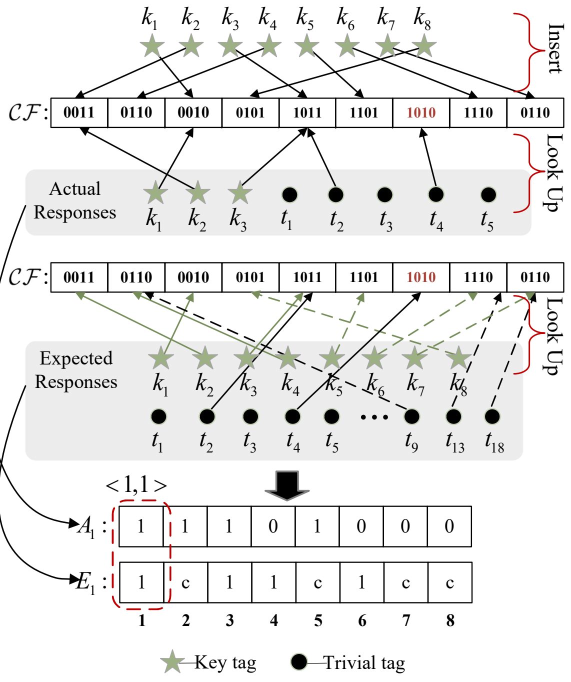


Fig. 4: An illustration of the execution of Kadept.

*2) Identifying Contentious Key Tags:* Readers can identify contentious key tags by working in different time windows with a specific scheduling algorithm, such as Colorwave [34]. In this scenario, each identification round consists of multiple schedules, and the readers inside the same schedule work in parallel. Consider the reader $R_i$. In one schedule, $R_i$ maintains another three tag sets: the local contentious key tag set $K_i^{L_c}$, the unidentified contentious key tag set $K_i^{U_c}$, and the active contentious trivial tag set $T_i^{A_c}$. They are initialized to empty set, $K - \sum_{i=1}^{m} K_i^{L_n}$, and $\bigcup_{\theta=1}^{|\Gamma(R_i)|+1} T_\theta^{A_n}$, respectively; where $R_\theta \in \{\Gamma(R_i) \bigcup R_i\}$. Similar to the method used for identifying noncontentious key tags, $R_i$ identifies contentious key tags by constructing and comparing two arrays. In addition, the labeled contentious key tags will keep silent after this round while not this schedule. In this way, each reader can identify all the contentious key tags it covers. Furthermore, by sharing information with other readers, each reader can know that each of its local contentious key tags is covered by which readers.

## E. *Case Study*

For clarity, we now use the scale-down RFID system shown in Fig. 1 to illustrate the execution process of one round in Kadept. We here only consider the case that the readers $R_{1-4}$ concurrently identify the noncontentious key tags $k_{1-3}$ and $k_{7-8}$, and we detail the identification process of $R_1$, as seen in Fig. 4. In the first round, $R_1$ constructs the Cuckoo filter $\mathcal{CF}$ using all key tags $k_{1-8}$, since it has no prior knowledge about tag distribution. $K_1^{L_n}$, $K_1^{U_n}$, and $T_1^{A_n}$ are initiated to $\emptyset$, $\{k_1, k_2, \cdots, k_8\}$, and $\{t_2, t_4, t_9, t_{13}, t_{18}\}$, respectively. $R_1$ inserts $k_{1-8}$ into $\mathcal{CF}$ with three specified hash functions first and then replaces the IDs of $k_{1-8}$ with their fingerprints. Besides, $R_1$ fills the empty bucket with the useless fingerprint '1010'. After getting $\mathcal{CF}$, $R_1$ broadcasts it and the construction parameters in the system. With the received information, the tags $k_{1-3}$ and $t_{1-5}$ that can listen to $R_1$ check whether they can pass $\mathcal{CF}$ or not. The key tags $k_{1-3}$ and the false positive trivial tag $t_2$ find their own fingerprint in one of their three candidate buckets, besides, their fingerprints are not identical with '1010', then they pass $\mathcal{CF}$. On this basis, each tag that passes the filter sets its slot index. For $k_1$, it sets its slot index to 3 on account of the observation that it maps to the third non-empty bucket of $\mathcal{CF}$, and replies $R_1$ in the third slot. Similarly, $k_2$ and $k_3$ reply $R_1$ in the first and fifth slot, respectively. According to the actual slot status, $R_1$ constructs $A_1$. With the information of key tags $k_{1-8}$ and trivial tags $t_{1-18}$, $R_1$ constructs $E_1$. Comparing $E_1$ and $A_1$, $R_1$ achieves eight slot pairs, based on which it labels tags and updates tag sets. For example, according to the first slot pair $\langle 1,1\rangle$, $R_1$ infers $k_2$ that expectedly maps to this slot is under its nonoverlapped region. It transmits ACK$_l$ to label $k_2$, adds $k_2$ to $K_1^{L_n}$, and removes $k_2$ from $K_1^{U_n}$. After this round, $K_1^{L_n}$, $K_1^{U_n}$, and $T_1^{A_n}$ update to $\{k_1,k_2\}$, $\{k_3,k_4\}$, and $\{t_2,t_{18}\}$, respectively.

## IV. Performance Analysis of Kadept

We first analyze the execution time of Kadept, denoted by $T_{Kadept}$, and then give the optimized parameters that can meet the given accuracy requirement and minimize the average time cost. From the global view of Kadept, $T_{Kadept}$ consists of two parts: 1) The time $T_n$ for readers to identify noncontentious key tags in parallel, and 2) the time $T_c$ for readers to identify contentious key tags based on the schedule algorithm Colorwave. According to Kadept, we have $T_n = \max\{T_i\}$ $(1 \le i \le m)$ and $T_c = \sum_{j=1}^{C} \max\{T_i^j\}$, where $C$ is the number of schedules. Consider the reader $R_i$. In the $\tau$th $(1 \le \tau \le w_i)$ round, the time for $R_i$ transmitting $\mathcal{CF}_\tau$ is $\lceil \frac{f_\tau \times d_\tau}{96} \rceil t_{ID}$, and the time for active tags transmitting one bit is $n_{k_\tau} \times t_s$; where $w_i$ is the number of rounds, $f_\tau$ is the length of $\mathcal{CF}_\tau$, and $n_{k_\tau}$ is the number of key tags used for constructing $\mathcal{CF}_\tau$. Besides, in each slot, the reader transmits an ACK to instruct the next action of tags in the corresponding slot. Hence, we have $T_i = \sum_{\tau=1}^{w_i} \{\lceil \frac{f_\tau \times d_\tau}{96} \rceil t_{ID} + n_{k_\tau} \times (t_s + t_{ack})\}$. We can achieve $T_{Kadept}$ by adding $T_n$ and $T_c$. Because each reader identifies noncontentious and contentious key tags in the same way, without loss of generality, we only analyze the time for $R_i$ identifying noncontentious key tags.

### A. Performance Efficiency

Due to the false positive of Cuckoo filter, some trivial tags may keep active and participate in the identification process, which causes the reader to fail to identify the key tags that map to collided buckets. We consider the worst case that each false positive trivial tag results in a key tag not being identified, and let the number of false negative key tags meet the following constraint:

$$I_{fn} = \frac{n_{fp}}{n_k} \le \alpha, \tag{2}$$

where $I_{fn}$ is the false negative indicator, $n_{fp}$ and $n_k$ are the numbers of false positive trivial tags and key tags. The parameter $\alpha$ $(0 < \alpha < 1)$ is the indicator threshold that limits false positive trivial tags, which affects the number of key tags that may remain unidentified in a single round. It determines the minimum fingerprint length $d$ for the Cuckoo filter. Since $\alpha$ defines only the lower bound for $d$, setting it too small may lead to an unnecessarily large $d$, potentially exceeding its optimal value, which will be analyzed shortly. Therefore, we recommend setting $\alpha$ to a moderate value (e.g., $\alpha = 0.1$) to ensure that the optimal fingerprint length remains achievable.

In one round, we aim to optimize the number of key tags to be identified and the number of false positive trivial tags to be filtered, which together minimize the total identification time. Recall that key tags can be identified by the slot pairs $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$, and the false positive trivial tags can be filtered by the slot pairs $\langle c,0\rangle$. We denote $E_{11}$, $E_{10}$, and $E_{c0}$ as the expected numbers of tags in $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$, respectively. For one round of $R_i$ with execution time $t$, the identification efficiency, denoted by $\theta$, is defined as the number of identified key tags per unit time:

$$\theta = \frac{E_{11} + E_{10} + E_{c0}}{t}, \tag{3}$$

where $t = \{\lceil \frac{f_{\mathcal{CF}} \times d}{96} \rceil t_{ID} + n_k \times (t_s + t_{ack})\}$.

We take the following lemmas to analyze $\theta$. Lemma 1 gives the expected values of $E_{11}$, $E_{10}$, $E_{c0}$, and $p_f$, where $p_f$ is the false positive ratio of $\mathcal{CF}$.

***Lemma 1:*** The expected values of $E_{11}$, $E_{10}$, $E_{c0}$, and $p_f$ are:

$$\begin{aligned} E_{11} &= n_k^l (1 - \frac{p_f}{n_k})^{n_t} \\ E_{10} &= (n_k - n_k^l)(1 - \frac{p_f}{n_k})^{n_t} \\ E_{c0} &= (n_k - n_k^l)\{(n_t - n_t^l)\frac{p_f}{n_k} \\ &\quad + \sum_{j=1}^{n_t - n_t^l} \binom{n_t - n_t^l}{j} (\frac{p_f}{n_k})^j (1 - \frac{p_f}{n_k})^{n_t - n_t^l - j}\} \\ p_f &= \frac{h}{2^d}(1 - \frac{h-1}{2^d}), \end{aligned} \tag{4}$$

where $n_k$, $n_t$, $n_k^l$, and $n_t^l$ are the numbers of key tags, trivial tags, local key tags, and local trivial tags, respectively.

*Proof:* In this round, the reader first constructs $\mathcal{CF}$ using the $n_k$ unidentified key tags and then broadcasts it to all tags. With the knowledge of $n_t$ trivial tags, the reader can predict the false positive trivial tags that will pass $\mathcal{CF}$; by incorporating the tags' real responses, it constructs the corresponding slot pairs. A slot pair $\langle 1,1\rangle$ occurs when a bucket of $\mathcal{CF}$ is inserted by one local key tag, with no trivial tag(s) mapping to it. Recall that empty buckets are filled with the useless fingerprint, so the corresponding empty slots are skipped. Consequently, the slot frame length, denoted by $f$, is equal to the number of nonempty buckets in $\mathcal{CF}$, that is, $f = n_k$. Let $p_{11}$ denote the probability of $\langle 1,1\rangle$ occurring. In the $f$ slots, the probability of a key tag is assigned to one slot is $\frac{1}{f}$, and the probability of that slot is mapped by a trivial tag is $\frac{p_f}{f}$. Thus, we have:

$$p_{11} = (1 - \frac{p_f}{f})^{n_t} \times \frac{n_k^l}{f} = (1 - \frac{p_f}{n_k})^{n_t} \times \frac{n_k^l}{n_k}. \tag{5}$$

For a bucket, the probability that a tag that is not inserted into this bucket and returns a false-positive successful match is at most $1/2^d$. After making $h$ such comparisons, the probability that only one successful match is:

$$p_f = \frac{h}{2^d}(1-\frac{1}{2^d})^{h-1} \approx \frac{h}{2^d}(1-\frac{h-1}{2^d}). \tag{6}$$

Hence, we have the expected number $E_{11}$ of identified key tags in slot pairs $\langle 1,1\rangle$:

$$E_{11} = p_{11}\times f = n_k^l(1-\frac{p_f}{n_k})^{n_t}, \tag{7}$$

we will give a new cardinality estimation method to estimate the values of $n_k^l$ and $n_t^l$ shortly.

The slot pair $\langle 1,0\rangle$ occurs when one bucket is inserted by only one nonlocal key tag. The probability that $\langle 1,0\rangle$ occurs, denoted by $p_{10}$, can be derived:

$$p_{10} = \frac{n_k-n_k^l}{n_k}(1-\frac{p_f}{n_k})^{n_t}. \tag{8}$$

With $p_{10}$, we can calculate the expected value of $E_{10}$:

$$E_{10} = (n_k-n_k^l)(1-\frac{p_f}{n_k})^{n_t}. \tag{9}$$

The slot pair $\langle c,0\rangle$ occurs when one bucket is inserted by one nonexclusive key tag and mapped by one or more false positive trivial tags in $T_i^{A_n}$. Similarly, we can derive the probability that $\langle c,0\rangle$ occurs, denoted by $p_{c0}$:

$$p_{c0} = \frac{n_k-n_k^l}{n_k}\sum_{j=1}^{n_t-n_t^l}\binom{n_t-n_t^l}{j}(\frac{p_f}{n_k})^j(1-\frac{p_f}{n_k})^{n_t-n_t^l-j}. \tag{10}$$

Since there is one key tag and one or more trivial tags in $\langle c,0\rangle$, we can derive the expected value of $E_{c0}$:

$$\begin{aligned}E_{c0} &= n_k\frac{n_k-n_k^l}{n_k}\sum_{j=1}^{n_t-n_t^l}\binom{n_t-n_t^l}{j}(j+1)(\frac{p_f}{n_k})^j(1-\frac{p_f}{n_k})^{n_t-n_t^l-j}\\ &= (n_k-n_k^l)\{(n_t-n_t^l)\frac{p_f}{n_k}\\ &+\sum_{j=1}^{n_t-n_t^l}\binom{n_t-n_t^l}{j}(\frac{p_f}{n_k})^j(1-\frac{p_f}{n_k})^{n_t-n_t^l-j}\}.\end{aligned} \tag{11}$$

■

## B. The Parameters and Reconstruction of Cuckoo Filter

*1) The Parameters of Cuckoo Filter:* Lemma 2 gives the optimized construction parameters of the Cuckoo filter $\mathcal{CF}$ that ensures the maximal $\theta_i$ in Eq. (3).

***Lemma** 2:* Given the false negative indicator $\alpha$, the values of $h$, $f_{\mathcal{CF}}$, and $d$ are:

$$\begin{aligned}h &= \lceil \ln n_k\rceil\\ f_{\mathcal{CF}} &= n_k+1\\ d &= \max(\theta) \quad s.t. \quad \frac{h}{2^d}(1-\frac{h-1}{2^d}) \le \frac{\alpha n_k}{n_t},\end{aligned} \tag{12}$$

where $f_{\mathcal{CF}}$ is the initial size of $\mathcal{CF}$.

*Proof:* To find the optimal values of $f_{\mathcal{CF}}$, $h$, and $d$ that maximize $\theta$, we analyze the constrain conditions of these three parameters. To ensure all $n_k$ key tags can be successfully inserted into $\mathcal{CF}$, the minimal value of $f_{\mathcal{CF}}$ is $n_k$, that is, $f_{\mathcal{CF}} \ge n_k$. Further, we derive the requirement that $h$ needs to satisfy to insert $n_k$ key tag into $f_{\mathcal{CF}}$ buckets. For one bucket, the probability that no tag is mapped to it by using $h$ hash functions is $(1-\frac{1}{f_{\mathcal{CF}}})^{n_k h}$, and the number of empty buckets is $f_{\mathcal{CF}}(1-\frac{1}{f_{\mathcal{CF}}})^{n_k h}$. Let $f_{\mathcal{CF}}(1-\frac{1}{f_{\mathcal{CF}}})^{n_k h} < 1$, we have the second constrain condition $h > \frac{f_{\mathcal{CF}}\ln f_{\mathcal{CF}}}{n_k}$. Besides, with Eq. (6), we can derive the number of false positive tags $n_{fp}$:

$$n_{fp} = n_t\times p_f = \frac{n_t h}{2^d}(1-\frac{h-1}{2^d}). \tag{13}$$

From Eq. (2) and Eq. (13), we know that the length $d$ of the fingerprint needs to satisfy $\frac{h}{2^d}(1-\frac{h-1}{2^d}) \le \frac{\alpha n_k}{n_t}$. With the above three constrain conditions, we can use one optimization function, such as fmincon, to solve the optimal solution of Eq. (3). The result is $f_{\mathcal{CF}} = n_k+1$, $h = \lceil \ln n_k\rceil$, and $d = \min(\tilde{d})$, where $\hat{d}$ meets $\frac{h}{2^{\hat{d}}}(1-\frac{h-1}{2^{\hat{d}}}) \le \frac{\alpha n_k}{n_t}$. Moreover, the optimal solution is not influenced by the initial values of $f_{\mathcal{CF}}$, $h$, and $d$. The initial value of $d$ is set to $\lceil log_2(n_k)\rceil$ for readers have no prior knowledge about the numbers of their local key tags and local trivial tags. ■

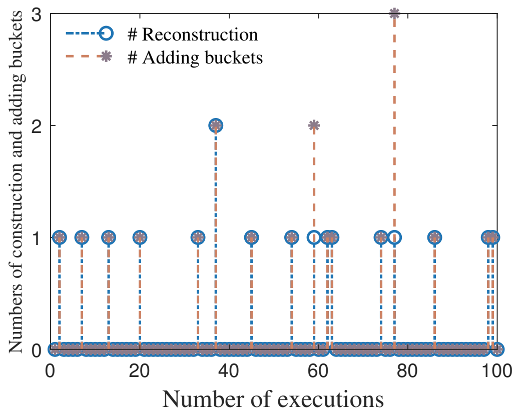


Fig. 5: An illustration of the reconstruction of Cuckoo filter.

*2) The Reconstruction of Cuckoo Filter:* With the optimal parameter settings, we study the construction of Cuckoo filter, as one reader may require multiple attempts to successfully construct the filter. Fig. 5 plots the reconstruction of Cuckoo filter in 100 independent simulations, where $n_k$, $n_t$, $n_k^l$, and $n_t^l$ are set to 500, 2000, 50 and 200, respectively. We observe that the number of reconstructions is zero in most cases, that is, a reader typically constructs the Cuckoo filter successfully on the first try. Even when reconstruction is needed, the number of reconstructions and additional buckets required is minimal. This suggests that the reader can efficiently construct Cuckoo filter with the optimal parameter settings.

## C. Cardinality Estimation

To achieve the optimal Cuckoo filter construction parameters in Lemma 2, one reader needs to estimate the cardinalities $n_k^l$ and $n_t^l$ for the sets of local key tags and still active trivial tags. However, utilizing the separate tag cardinality estimation protocols will increase the execution time. We propose an estimation scheme without extra time consumption by using the information identified. Consider any arbitrary round and the reader $R_i$. After this round, $R_i$ precisely counts the number of slot pairs $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$, denoted as $n_{11}$, $n_{10}$, and $n_{c0}$, respectively. Since one key tag corresponds to exactly one slot pair, the number of key tags that map to $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$ is equal to $n_{11}$, $n_{10}$, and $n_{c0}$, respectively. However, for the slot pair $\langle c,0\rangle$, both one key tag and one or more trivial tags map to it. We denote the number of trivial tags mapping to $\langle c,0\rangle$ as $n^t_{c0}$. With $n_{11}$ and Eq. (23), we have:

$$n^l_k = n_{11} e^{\frac{p_f n_t}{n_k}}. \tag{14}$$

Based on this, we derive the expected number of trivial tags in slot pairs $\langle c,0\rangle$, denoted by $E^t_{c0}$:

$$\begin{aligned} E^t_{c0} &= n_k \frac{n_k - n^l_k}{n_k} \sum_{j=1}^{n_t - n^l_t} \binom{n_t - n^l_t}{j} j \left(\frac{p_f}{n_k}\right)^j \left(1 - \frac{p_f}{n_k}\right)^{n_t - n^l_t - j} \\ &= (n_k - n^l_k)(n_t - n^l_t)\frac{p_f}{n_k}. \end{aligned} \tag{15}$$

With $n^l_k$, $n^t_{c0}$, and $E^t_{c0}$, we can calculate $n^l_t$:

$$n^l_t = n_t - \frac{n^t_{c0} n_k}{p_f (n_k - n^l_k)}. \tag{16}$$

According to the obtained parameters, the reader can update the parameters used in the next round to identify the left key tags. Lemma 3 gives how to achieve the updated $n_k$, $n_t$, $n^l_k$, and $n^l_t$, which are denoted as $\hat{n_k}$, $\hat{n_t}$, $\hat{n^l_k}$, and $\hat{n^l_t}$, respectively.

***Lemma** 3:* The expected values of $\hat{n_k}$, $\hat{n_t}$, $\hat{n^l_k}$, and $\hat{n^l_t}$ are:

$$\begin{aligned} \hat{n_k} &= n_k - n_{11} - n_{10} - n_{c0} \\ \hat{n_t} &= n_{fp} - n^t_{c0} \\ \hat{n^l_k} &= n^l_k - n_{11} \\ \hat{n^l_t} &= n^l_t. \end{aligned} \tag{17}$$

*Proof:* Since each of the slot pairs $\langle 1,1\rangle$, $\langle 1,0\rangle$, and $\langle c,0\rangle$ corresponds to an identified key tag, the number of key tags that are removed from $K_i^{U_n}$ is $n_{11}+n_{10}+n_{c0}$. Therefore, $\hat{n_k}$ is equal to $n_k - n_{11} - n_{10} - n_{c0}$. Moreover, after filtering, the $n_{fp}$ trivial tags that are still active are further identified by slot pairs $\langle c,0\rangle$ and removed from $T_i^{A_n}$. Hence, the number $\hat{n_t}$ of active trivial tags is $n_{fp} - n^t_{c0}$. Remember that the local key tags that map to $\langle 1,1\rangle$ will be labeled and do not participate in the following identification. Therefore, the expected value of $\hat{n^l_k}$ is $n^l_k - n_{11}$. Since the reader cannot infer the information of local trivial tags from $\langle c,1\rangle$, all the local trivial tags keep active, we have $\hat{n^l_t} = n^l_t$. ■

## V. Enhanced Kadept Protocol

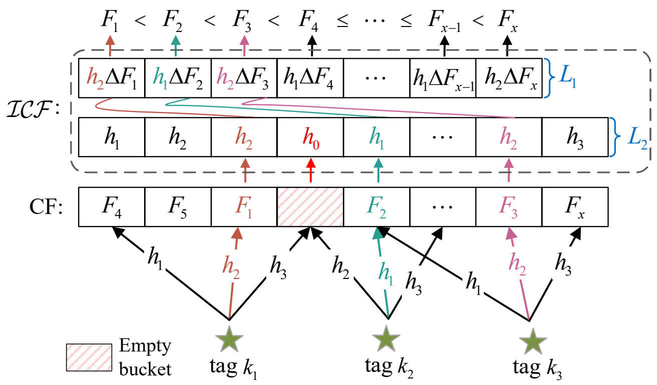
Kadept, while efficient in identifying the distribution of key tags, experiences performance declines as the number of key tags increases. This is because the Cuckoo filter used in Kadept expands with the number of key tags, requiring more time for the reader to transmit the filter. To enhance Kadept's scalability and efficiency, we propose an incremental Cuckoo filter that stores the differences between consecutive fingerprints of key tags instead of the full fingerprints, along with the index of the hash equation used for successful insertions. This filter is more space-efficient, has a lower false positive ratio, and grows more slowly than the traditional Cuckoo filter. Building on the new filter, we further propose the enhanced Kadept protocol, referred to as E-Kadept. In what follows, we first present the incremental Cuckoo filter and then describe the E-Kadept protocol in detail.

### *A. Incremental Cuckoo Filter*

In Kadept, the false positive ratio $p_f$ of the Cuckoo filter $\mathcal{CF}$ it constructs increases with the number of hash equations $h$ and decreases with the fingerprint length $d$. To minimize $p_f$, we can decrease $h$ and increase $d$. However, this trade-off leads to higher storage overhead for $\mathcal{CF}$. Reducing $h$ increases the number of buckets to ensure that all key tags are successfully inserted into $\mathcal{CF}$, while increasing $d$ raises the number of bits stored in each bucket. This finally increases the time for the reader to transmit $\mathcal{CF}$. To address this challenge, we design the incremental Cuckoo filter, denoted by $\mathcal{ICF}$. $\mathcal{ICF}$ stores the difference between consecutive fingerprints instead of the full fingerprints themselves, effectively reducing the size of each bucket. Furthermore, to filter out most non-key tags and assign a unique slot for each inserted key tag, $\mathcal{ICF}$ stores the indexes of the hash equations used to calculate the location and fingerprint of each inserted tag. This section describes how to construct the $\mathcal{ICF}$, and how tags perform lookup operations.

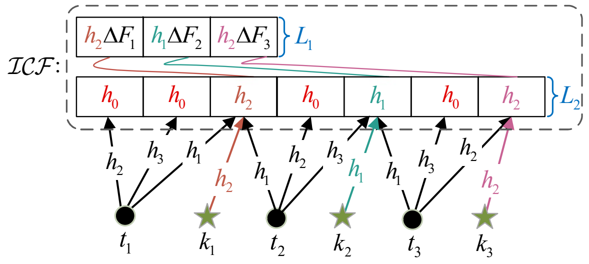
*1) Construction:* The $\mathcal{ICF}$ is constructed as follows. First, the reader constructs a traditional Cuckoo filter (CF) with key tags. Specifically, for one key tag $k$, the reader checks the $h$ buckets in the CF by calculating $H(ID_k, s_\hbar) \mod f_{2_\mathcal{I}}$, where $f_{2_\mathcal{I}}$ is the length of $\mathcal{ICF}$'s second layer. The optimal value of $f_{2_\mathcal{I}}$ will be analyzed shortly. If not all $h$ buckets are empty, the reader inserts $k$ into the CF by putting its ID into any one of the empty buckets. Otherwise, the reader randomly selects one of the $h$ buckets and replaces the existing key tag with $k$. The displaced key tag is relocated with the same insertion steps. This process continues until all key tags are successfully inserted or the maximum iterations are reached. If the construction fails, the reader increases the size of the CF and reconstructs it until all key tags are successfully inserted. Each non-empty bucket stores the fingerprint of key tag using $H(ID_k, s_f)$, and empty buckets can be left empty.

Second, the reader builds $\mathcal{ICF}$ based on CF, which consists of two layers. To insert $x$ key tags, the reader first sorts their fingerprints in ascending order, denoted as $F_1 < F_2 \le \cdots \le F_x$. The reader then calculates the difference between consecutive fingerprints, denoted as $\Delta F_i = F_i - F_{i-1}$, resulting in the fingerprint difference set $\{\Delta F_1, \Delta F_2, \cdots, \Delta F_x\}$, where $F_0 = 0$. In the first layer, for the $i$th bucket, the reader stores $\Delta F_i$ along with the index of the hash equation $h_\hbar$ used to successfully insert the $i$th key tag into the CF. Therefore, the first layer has a length of $f_{1_\mathcal{I}} = x$. In the second layer, the $i$th bucket only stores the index of the hash equation $h_\hbar$ that successfully inserts the key tag into the $i$th bucket of the CF. For empty buckets, the reader fills them with a random useless hash equation's index, denoted as $h_u$. A useless hash equation is one that is not used to insert any key tag within the set $\{1, 2^{\lceil \log_2(h) \rceil}\}$. However, when $\lceil \log_2(h) \rceil = \log_2(h)$, no



$F_1 < F_2 < F_3 < F_4 \leq \cdots \leq F_{x-1} < F_x$
$\mathcal{ICF}$:
$h_2\Delta F_1$ $h_1\Delta F_2$ $h_2\Delta F_3$ $h_1\Delta F_4$ $\cdots$ $h_1\Delta F_{x-1}$ $h_2\Delta F_x$
$L_1$
$h_1$ $h_2$ $h_2$ $h_0$ $h_1$ $\cdots$ $h_2$ $h_3$
$L_2$
CF:
$F_4$ $F_5$ $F_1$ $F_2$ $\cdots$ $F_3$ $F_x$
$h_1$ $h_2$ $h_3$ $h_2$ $h_1$ $h_3$ $h_1$ $h_2$ $h_3$
Empty bucket
tag $k_1$
tag $k_2$
tag $k_3$

Fig. 6: Illustration of incremental Cuckoo filter construction.



$\mathcal{ICF}$:
$h_2\Delta F_1$ $h_1\Delta F_2$ $h_2\Delta F_3$
$L_1$
$h_0$ $h_0$ $h_2$ $h_0$ $h_1$ $h_0$ $h_2$
$L_2$
$h_2$ $h_3$ $h_1$ $h_2$ $h_1$ $h_2$ $h_3$ $h_1$ $h_1$ $h_3$ $h_2$ $h_2$
$t_1$ $k_1$ $t_2$ $k_2$ $t_3$ $k_3$

Fig. 7: An example of incremental Cuckoo filter lookup.

unused hash equation exists. In this case, the reader constructs a vector $EI$ to record the indexes of empty buckets.

For the reconstruction of the incremental Cuckoo filter, since it builds upon the initial Cuckoo filter constructed in Kadept, its reconstruction scenario is analogous to that of the Cuckoo filter, as detailed in Section IV-B2. For brevity, we do not repeat the details here.

*2) Lookup:* We now present the lookup process of the $\mathcal{ICF}$. Given a tag $k$, it first checks each bucket of the first layer. For the $i$th ($1 \leq i \leq f_{1_\mathcal{I}}$) bucket, the tag calculates its hash value, denoted as $v_k^i$, using the hash equation index stored in this bucket and Eq. (1). It then checks the $v_k^i$th bucket in the second layer. This check involves two cases.

**Case 1**: If no useless hash equations exist ($h_u$ does not exist) and $v_k^i$ is in $EI$, the $v_k^i$th bucket is considered empty.

**Case 2**: If at least one useless hash equation exists ($h_u$ is a randomly selected useless hash equation's index) and the value in the $v_k^i$th bucket is $h_u$, the $v_k^i$th bucket is empty.

In both cases, if the $v_k^i$th layer bucket is empty, $k$ skips the current lookup and moves to the next bucket in the first layer. Otherwise, $k$ checks whether $v_k^i$ matches the value stored in the $v_k^i$th bucket. If they do not match, $k$ skips to the next bucket in the first layer. If they match, $k$ further calculates its fingerprint $F_k$ using Eq. (1) and checks if it equals the sum of the fingerprint differences in the first $i$ buckets of the first layer. If $F_k = \sum_{i=1}^{i} \Delta F_i$, $k$ increments its success counter, denoted as $p_k$ (initially 0), by 1 and proceeds. Otherwise, it skips. The lookup continues until $h$ checks or all buckets are checked. If $p_k = 1$, $k$ passes the lookup; otherwise, it deactivates itself. This approach helps filter out the false positive tags that match multiple buckets. Note that the reader resolves the fingerprint collisions during the $\mathcal{ICF}$ construction, ensuring that each key tag maps to only one bucket.

*3) Illustration:* Fig. 7 illustrates the lookup process in a scale-down system with three key tags $k_{1-3}$ and three trivial tags $t_{1-3}$. The reader constructs the $\mathcal{ICF}$ using $k_{1-3}$ with fingerprints $F_1 < F_2 < F_3$, as shown in Fig. 6, and broadcasts it with its construction parameters to tags. All tags that receive the $\mathcal{ICF}$ perform the lookup. We take $k_2$ and $t_3$ as examples.

For $k_2$, it begins by checking the first bucket in the first layer and calculates the hash value using the hash equation index $h_2$ stored in that bucket, which is 4. Next, it checks the fourth bucket in the second layer, which contains $h_0$, prompting it to move to the second bucket of the first layer containing $h_1$. Using $h_1$, $k_2$ calculates a hash value of 5 and checks the fifth bucket in the second layer, which matches $h_1$. It then compares the sum of the fingerprint differences in the first two buckets of the first layer with its fingerprint $F_2$. Since $F_2 = \Delta F_1 + \Delta F_2 = F_1 + (F_2 - F_1) = F_2$, $k_2$ increments its success counter $p_2$ by 1 and proceeds. Consequently, $k_2$ passes the lookup with $p_2 = 1$. Similarly, $k_1$ and $k_3$ can also successfully pass the lookup.

For $t_3$, it calculates a hash value of 7 using $h_2$ from the first bucket. The seventh bucket in the second layer contains $h_2$, prompting $t_3$ to compare the fingerprint in the first bucket of the first layer with its fingerprint $F_3$. However, comparing $F_3$ with $\Delta F_1$ reveals a mismatch. $t_3$ then checks the remaining buckets, but all fail. Thus, $t_3$ fails the lookup and deactivates itself. Similarly, $t_1$ and $t_2$ fail to pass the lookup.

## *B. E-Kadept Protocol Description*

Due to false positives in the incremental Cuckoo filter, some false positive trivial tags may cause the misidentification of the distribution of key tags. To identify all key tags accurately, E-Kadept runs multiple rounds. Each round has three phases: 1) the constructing phase to construct an incremental Cuckoo filter with the unidentified key tags; 2) the filtering-assigning-separating phase to filter out trivial tags, assign each key tag a unique and continuous slot, and separate key tags from trivial tags; 3) the identifying phase that identifies the distribution of key tags. Following describes these phases separately.

*1) Constructing Phase:* In this phase, a reader builds an incremental Cuckoo filter $\mathcal{ICF}$ with the unidentified key tags, using the method described in Section V-A1.

*2) Filtering-assigning-separating Phase:* After getting the $\mathcal{ICF}$, the reader broadcasts it and its construction parameters $\langle f_{1_\mathcal{I}}, f_{2_\mathcal{I}}, s_{1-h}, s_f, d, \Delta d, \Delta h, h_u, EI \rangle$ to tags, where $\Delta d$ and $\Delta h$ represent the number of bits needed to represent the maximum fingerprint difference and the hash equation index, respectively. Additionally, if there is no useless hash equation, $h_u$ is set to $\emptyset$; otherwise, $EI$ is set to $\emptyset$. Upon receiving this query request, each tag according to the lookup process described in Section V-A2 to determine whether it can pass the $\mathcal{ICF}$ or not. If a tag cannot pass the $\mathcal{ICF}$, it updates its flag to 1 before keeping silent, indicating that it is a trivial tag. Otherwise, it keeps active and calculates its slot index used for the identification phase as follows.

Let $SI_j$ denote the slot index of key tag $k_j$. $k_j$ records the index of the bucket in the first layer where it passes the $\mathcal{ICF}$, denoted as $p_j$, and sets $SI_j = p_j$. Note that only tags that successfully pass the $\mathcal{ICF}$ are assigned a unique slot index, as tags matching multiple buckets will fail the $\mathcal{ICF}$ and keep silent. Alternatively, each key tag can calculate its slot index by counting the number of non-empty buckets up to its matched bucket in the second layer. Both methods ensure each key tag has a unique slot index, but may assign different values. For simplicity, we use the first method.

*3) Identifying Phase:* Similar to Kadept, E-Kadept concurrently identifies the distribution of noncontentious key tags and then identifies the distribution of contentious key tags using a reader schedule algorithm. The details are described in Section III-D and are omitted here.

So far, we have proposed two protocols Kadept and E-Kadept. For a given application, the user can select the appropriate protocol based on their requirements. Kadept, with its simpler design, is preferable in small-scale systems or when the proportion of key tags is low, particularly when upper-layer applications are not time-sensitive. Otherwise, E-Kadept is recommended due to its higher time efficiency.

## VI. Performance Analysis of E-Kadept

In this section, we analyze the execution time $T_{E-Kadept}$ to minimize the average time for fully identifying all key tags. Similar to Kadept, $T_{E-Kadept} = T_n + T_c = \max\{T_i\} + \sum_{j=1}^{C}\max\{T_i^j\}$. Where $T_n$ is the parallel identification time for noncontentious key tags; $T_c$ is the time for identifying contentious key tags through reader scheduling; $T_i$ is the time for $R_i$; $T_i^j$ is the time for $R_i$ to identify contentious key tags in the $j$th schedule; and $C$ is the number of schedules. The definitions of these parameters align with those in Kadept, while the value of $T_i$ differs in E-Kadept. For reader $R_i$, in the $\tau$th round, the total time $T_{i_\tau}$ includes the time for transmitting $\mathcal{ICF}$, sending ACK, and receiving one-bit responses from key tags. In addition, if no useless hash equations exist, the reader transmits $EI$ to tags. Thus, $T_{i_\tau}$ is:

$$\begin{cases} \lceil\frac{(f_{1_{\mathcal{I}_\tau}}+f_{2_{\mathcal{I}_\tau}})\lceil\log_2 h\rceil+f_{1_{\mathcal{I}_\tau}}\times\Delta d_\tau}{96}\rceil t_{ID}+n_{k_\tau}(t_s+t_{ack}), & h_u\neq\emptyset,\\ \lceil\frac{(f_{1_{\mathcal{I}_\tau}}+f_{2_{\mathcal{I}_\tau}})\lceil\log_2 h\rceil+f_{1_{\mathcal{I}_\tau}}\times\Delta d_\tau}{96}\rceil t_{ID}+n_{k_\tau}(t_s+t_{ack})+t_{EI}, & h_u=\emptyset, \end{cases} \tag{18}$$

where $t_{EI}$ is the time for transmitting $EI$, $f_{1_{\mathcal{I}_\tau}}$ and $f_{2_{\mathcal{I}_\tau}}$ are the lengths of the first and second layers of $\mathcal{ICF}_\tau$, and $\Delta d_\tau$ is the bit length for the maximum fingerprint difference in $\mathcal{ICF}_\tau$. From the construction of $EI$, we have:

$$t_{EI} = \frac{\lceil\log_2 f_{2_{\mathcal{I}_\tau}}\rceil\times(f_{2_{\mathcal{I}_\tau}}-n_{k_\tau})}{96}t_{ID}. \tag{19}$$

The total time $T_i$ for reader $R_i$ is $\sum_{\tau=1}^{w_i}T_{i_\tau}$, where $w_i$ denotes the number of rounds.

### *A. Performance Efficiency*

We optimize the parameters of the incremental Cuckoo filter to maximize single-round tag identification efficiency $\theta$ in Eq. (3). In E-Kadept, parameters such as $n_{f_p}$, $E_{11}$, $E_{10}$, $E_{c0}$, $n_k^l$, $n_t^l$, and $t$ are similar to those in Kadept, but with different values. In particular, $t$ corresponds to $t_\tau$ in Eq. (18). Lemma 4 gives the expected values of $E_{11}$, $E_{10}$, $E_{c0}$, $n_k^l$, and $n_t^l$.

***Lemma 4:*** The expected values of $E_{11}$, $E_{10}$, $E_{c0}$, $n_k^l$, and $n_t^l$ are:

$$\begin{aligned} E_{11} &= n_k^l(1-\frac{\frac{1}{2^d}e^{-\frac{1}{2^d}}}{n_k})^{n_t}\\ E_{10} &= (n_k-n_k^l)(1-\frac{\frac{1}{2^d}e^{-\frac{1}{2^d}}}{n_k})^{n_t}\\ E_{c0} &= (n_k-n_k^l)\{(n_t-n_t^l)\frac{\frac{1}{2^d}e^{-\frac{1}{2^d}}}{n_k}\\ &\quad+\sum_{j=1}^{n_t-n_t^l}\binom{n_t-n_t^l}{j}(\frac{\frac{1}{2^d}e^{-\frac{1}{2^d}}}{n_k})^j(1-\frac{\frac{1}{2^d}e^{-\frac{1}{2^d}}}{n_k})^{n_t-n_t^l-j}\}\\ n_k^l &= n_{11}e^{\frac{n_t}{n_k}\frac{1}{2^d}e^{-\frac{1}{2^d}}}\\ n_t^l &= n_t-\frac{n_{c0}^t n_k}{\frac{1}{2^d}e^{-\frac{1}{2^d}}(n_k-n_{11}e^{\frac{n_t}{n_k}\frac{1}{2^d}e^{-\frac{1}{2^d}}})}, \end{aligned} \tag{20}$$

where $n_k$, $n_t$, and $\alpha$ are the numbers of key tags, trivial tags, and given false negative indicator threshold, respectively.

*Proof:* The reader constructs the $\mathcal{ICF}$ with $n_k$ tags, then broadcasts it and its parameters. Each tag performs a lookup, and only those that pass the filter respond to the reader in their assigned slots according to their slot indexes. We analyze the false positive probability $p_{f_I}$ of the incremental Cuckoo filter. For one bucket in the first layer the $\mathcal{ICF}$, the probability that a tag that is not inserted into this bucket still returns a successful match (i.e., a false positive) is at most $1/2^d$, as the tag must match the $d$-bit fingerprint calculated from the corresponding buckets' fingerprint differences. According to the incremental Cuckoo filter construction, each tag checks at most $h$ times during a lookup. Let $h_c$ be the number of checks, where $1 \le h_c \le h$. The probability of one successful match after $h_c$ comparisons is:

$$\begin{aligned} p_{f_I} &= \binom{h_c}{1}\left(\frac{1}{h_c}\times\frac{1}{2^d}\right)\left(1-\frac{1}{h_c}\times\frac{1}{2^d}\right)^{h_c-1}\\ &= \frac{1}{2^d}(1-\frac{1}{h_c\times 2^d})^{h_c-1}\\ &\approx \frac{1}{2^d}e^{-\frac{1}{2^d}}. \end{aligned} \tag{21}$$

By comparing expected and actual slot statuses, the reader determines slot pairs (e.g., $\langle 1,1\rangle$, $\langle 1,0\rangle$, $\langle c,0\rangle$). As the $\mathcal{ICF}$ assigns a unique slot index to each key tag, the slot frame size $f$ is also $n_k$. A $\langle 1,1\rangle$ slot pair occurs when a bucket in the $\mathcal{ICF}$ is filled by a local key tag without any false positive trivial tags. Let $p_{11}$ denote this probability. We have:

$$p_{11} = (1-\frac{p_{f_I}}{f})^{n_t}\times\frac{n_k^l}{f}. \tag{22}$$

Hence, the expected number $E_{11}$ is:

$$E_{11} = p_{11}\times f = n_k^l(1-\frac{\frac{1}{2^d}e^{-\frac{1}{2^d}}}{n_k})^{n_t}. \tag{23}$$

The value of $d$ will be analyzed shortly. Given $d$, the values of $n_k^l$ and $n_t^l$ can be calculated using the cardinality estimation method described in Section IV-C. From Eq. (14), we have:

$$\begin{aligned} n_k^l &= n_{11} e^{\frac{p_{f_I}}{n_k} n_t} \\ &= n_{11} e^{\frac{n_t}{n_k} \frac{1}{2^d} e^{-\frac{1}{2^d}}}, \end{aligned} \tag{24}$$

where $n_{11}$ is the number of $\langle 1,1 \rangle$ slot pairs counted by the reader. Moreover, based on Eq. (16), we can calculate $n_t^l$:

$$\begin{aligned} n_t^l &= n_t - \frac{n_{c0}^t n_k}{p_{f_I}(n_k - n_k^l)} \\ &= n_t - \frac{n_{c0}^t n_k}{\frac{1}{2^d} e^{-\frac{1}{2^d}} (n_k - n_{11} e^{\frac{n_t}{n_k} \frac{1}{2^d} e^{-\frac{1}{2^d}}})}. \end{aligned} \tag{25}$$

The slot pair $\langle 1,0 \rangle$ occurs when a slot is assigned by exactly one nonlocal key tag and no false positive trivial tags. Thus, the probability $p_{10}$ of $\langle 1,0 \rangle$ is:

$$\begin{aligned} p_{10} &= \frac{n_k - n_k^l}{f}(1 - \frac{p_{f_I}}{f})^{n_t} \\ &= \frac{n_k - n_k^l}{n_k}(1 - \frac{\frac{1}{2^d} e^{-\frac{1}{2^d}}}{n_k})^{n_t}. \end{aligned} \tag{26}$$

And the expected value of $E_{10}$:

$$E_{10} = (n_k - n_k^l)(1 - \frac{\frac{1}{2^d} e^{-\frac{1}{2^d}}}{n_k})^{n_t}. \tag{27}$$

Next, a slot pair $\langle c,0 \rangle$ occurs when a bucket is filled by a nonlocal key tag and one or more false positive trivial tags. We can derive the expected value of $E_{c0}$ that represents the expected number of tags (key tags and trivial tags) in $\langle c,0 \rangle$:

$$\begin{aligned} E_{c0} &= n_k \frac{n_k - n_k^l}{n_k} \sum_{j=1}^{n_t - n_t^l} \binom{n_t - n_t^l}{j} (j+1) (\frac{p_{f_I}}{n_k})^j (1 - \frac{p_{f_I}}{n_k})^{n_t - n_t^l - j} \\ &= (n_k - n_k^l)\{(n_t - n_t^l) \frac{\frac{1}{2^d} e^{-\frac{1}{2^d}}}{n_k} \\ &+ \sum_{j=1}^{n_t - n_t^l} \binom{n_t - n_t^l}{j} (\frac{\frac{1}{2^d} e^{-\frac{1}{2^d}}}{n_k})^j (1 - \frac{\frac{1}{2^d} e^{-\frac{1}{2^d}}}{n_k})^{n_t - n_t^l - j}\}. \end{aligned} \tag{28}$$

For clarity, we keep $n_k^l$, $n_t^l$, and $d$ in the expressions for $E_{11}$, $E_{10}$, and $E_{c0}$. Their values can be calculated using the known parameters $n_k$, $n_t$, $n_{11}$, $n_{c0}$, and $\alpha$. ■

## B. The Parameters of Incremental Cuckoo Filter

Lemma 5 provides the optimized construction parameters of the incremental Cuckoo filter $\mathcal{ICF}$ that maximize single-round tag identification efficiency $\theta$.

***Lemma** 5:* Given the false negative indicator $\alpha$, the values of $h$, $f_{1_\mathcal{I}}$, $f_{2_\mathcal{I}}$, d, and $\Delta d$ are:

$$\begin{aligned} h &= 3 \\ f_{1_\mathcal{I}} &= n_k \\ f_{2_\mathcal{I}} &= \lfloor 1.0633 n_k \rceil \\ \Delta d &= \lceil log_2 \lceil \frac{\ln n_k + \gamma}{n_k} \times 2^d \rceil \rceil \\ d &= \max(\theta) \quad s.t. \quad \min(d) = \lceil log_2 \frac{n_t - \alpha n_k}{\alpha n_k} \rceil, \end{aligned} \tag{29}$$

where $\gamma$ is Euler's constant.

*Proof:* From the $\mathcal{ICF}$ construction, $f_{1_\mathcal{I}} = n_k$. As stated in Lemma 4, $\theta$'s numerator $(E_{11} + E_{10} + E_{c0})$ depends solely on $d$ with fixed parameters. The denominator $t$ (Eq. (18)) depends on both $h$ and $d$, as $f_{2_\mathcal{I}}$ is related to $h$. To maximize $\theta$, we first find the optimal $h$ that minimizes $h$-dependent terms in $t$, then optimize $d$ with this $h$. To insert all $n_k$ key tags into the $\mathcal{ICF}$, we require $f_{2_\mathcal{I}} \geq n_k$. For one bucket in the second layer, the probability that no tag is mapped to it by using $h$ hash functions is $(1 - \frac{1}{f_{2_\mathcal{I}}})^{n_k h}$, and the expected number of empty buckets is $f_{2_\mathcal{I}}(1 - \frac{1}{f_{2_\mathcal{I}}})^{n_k h}$. Therefore, we have:

$$f_{2_\mathcal{I}} - f_{2_\mathcal{I}}(1 - \frac{1}{f_{2_\mathcal{I}}})^{n_k h} = n_k. \tag{30}$$

From the above equation, we can derive the $h$-$f_{2_\mathcal{I}}$ relationship:

$$h = \frac{-\ln(1 - \frac{n_k}{f_{2_\mathcal{I}}})}{\frac{n_k}{f_{2_\mathcal{I}}}}. \tag{31}$$

By varying $f_{2_\mathcal{I}}$ from $n_k$ to $1.5n_k$, we obtain the corresponding values $h$ and $\lceil \log_2 h \rceil$, as shown in Fig. 8(a). The total number of bits related to $h$ and $f_{2_\mathcal{I}}$ in $t$, denoted as $B(h)$, is:

$$B(h) = (f_{1_\mathcal{I}} + f_{2_\mathcal{I}})\lceil \log_2 h \rceil + e(h), \tag{32}$$

where $e(h)$ is the number of bits for transmitting the $EI$ to tags. If $\lceil \log_2(h) \rceil = \log_2(h)$, $e(h) = \lceil \log_2 f_{2_\mathcal{I}} \rceil \times (f_{2_\mathcal{I}} - n_k)$; otherwise, $e(h) = 0$. By minimizing $B(h)$, we determine the optimal $h$ that minimizes the $h$-dependent bits transmitted by the reader in $t$. As illustrated in Fig. 8(b), the optimal $h$ is 3. With increasing $f_{2_\mathcal{I}}$, both $h$ and $\lceil \log_2 h \rceil$ decrease in discrete steps. Correspondingly, $B(h)$ decreases when $\lceil \log_2 h \rceil$ drops; however, when $\lceil \log_2 h \rceil$ remains constant, $B(h)$ may vary. For instance, when $h = 3$ and $h = 4$, both have $\lceil \log_2 h \rceil = 2$, but $B(h)$ for $h = 3$ is smaller. This is because $h = 4$ requires additional $e(h)$ bits due to $\lceil \log_2 h \rceil = \log_2(h)$. The optimal $h$ of 3 is independent of the number of tags used to construct the $\mathcal{ICF}$. In Fig. 8(c), when $n_k$ is 1000 and $f_{2_\mathcal{I}}$ varies from 1000 to 1500, the optimal $h$ remains 3. To determine the optimal $f_{2_\mathcal{I}}$, we solve Eq. (31) by finding the intersection of $e^{3(1-n_k/f_{2_\mathcal{I}})}$ and $e^3(1 - n_k/f_{2_\mathcal{I}})$. Using a numerical technique like bisection, we obtain $n_k/f_{2_\mathcal{I}} \approx 0.9405$, $f_{2_\mathcal{I}} \approx 1.0633n_k$. As shown in Fig. 8(d), for $n_k = 500$, the optimal $f_{2_\mathcal{I}}$ is 532.

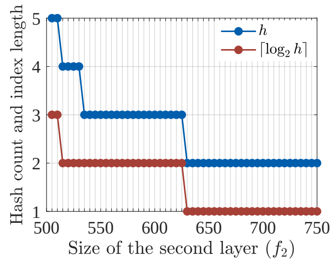
To optimize $d$, we determine the minimum fingerprint length $\min(d)$ to meet the identification accuracy and calculate the expected value of $\Delta d$. Substituting $\Delta d$ and other parameters into Eq. (3), we numerically solve $d$ to maximize $\theta$. From Eq. (21), the number of false positive tags of the $\mathcal{ICF}$ is:

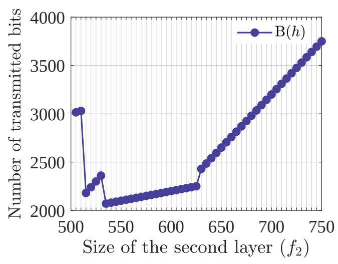
$$n_{fp} = n_t \times p_{f_I} = \frac{n_t}{2^d} e^{-\frac{1}{2^d}}. \tag{33}$$

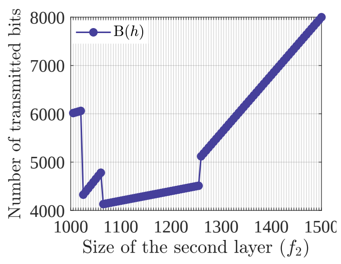
Based on Eq. (2), the minimum fingerprint length is:

$$\min(d) = \lceil log_2 \frac{n_t - \alpha n_k}{\alpha n_k} \rceil. \tag{34}$$

We now analyze the expected value of $\Delta d$, which is given by $\lceil \log_2 \max(\Delta F_1, \Delta F_2, \cdots, \Delta F_{n_k}) \rceil$. To compute this, we first determine $E[\max(\Delta F_1, \Delta F_2, \cdots, \Delta F_{n_k})]$. This can be reformulated as the problem of finding the expected length of


(a) Required $h$ vs. $f_{2_{\mathcal{I}}}$


(b) Bit cost $B(h)$ vs. $f_{2_{\mathcal{I}}}$ ($n_k$ = 500)


(c) Bit cost $B(h)$ vs. $f_{2_{\mathcal{I}}}$ ($n_k$ = 1000)

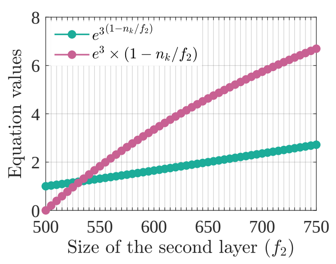

(d) Optimal $f_{2_{\mathcal{I}}}$ at $h$ = 3

Fig. 8: Analysis of $h$ and $f_{2_{\mathcal{I}}}$ for optimal $\mathcal{ICF}$ construction.

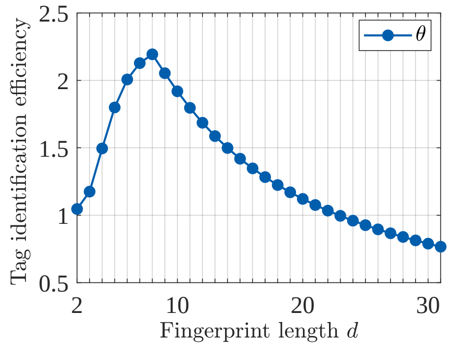
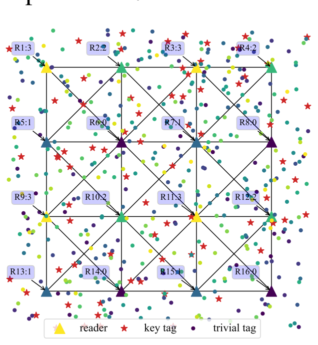


Fig. 9: Fingerprint length vs. tag identification efficiency.


(a) Regular readers & random tags

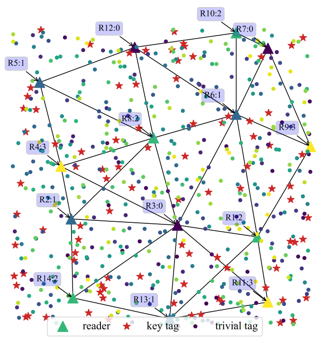

(b) Random readers & random tags

Fig. 10: Uniform tag deployment with grid and random readers.

the longest segment in a rope of length $2^d$ that is randomly divided into $n_k$ segments. Let $F_1$, $F_2$, ..., $F_{n_k}$ denote the ordered positions of these cuts, with $\Delta F_i = F_i - F_{i-1}$ representing the length of the $i$-th segment. The probability that any specific subset of $k$ segments simultaneously exceeds given thresholds $c_1, c_2, \cdots, c_k$ ($\sum_{i=1}^{k} c_i \leq 1$) is $(1-c_1-c_2-\cdots-c_k)^{n_k-1}$ [35]. Furthermore, applying the inclusion-exclusion principle, when $c_1 = c_2 = \cdots = c_k = z$, the probability that the maximum segment length exceeds $z$ is:

$$\begin{aligned} &P(\max(\Delta F_1, \Delta F_2, \cdots, \Delta F_{n_k}) > z) \\ &= \sum_{k=1}^{n_k} (-1)^{k-1} \binom{n_k}{k} (1-kz)^{n_k-1}. \end{aligned} \tag{35}$$

And the expected value of $\max(\Delta F_1, \Delta F_2, \cdots, \Delta F_{n_k})$ is:

$$\begin{aligned} &E(\max(\Delta F_1, \Delta F_2, \cdots, \Delta F_{n_k}) > z) \\ &= 2^d \int_0^\infty P(\max(\Delta F_1, \Delta F_2, \cdots, \Delta F_{n_k}) > z) dz \\ &= 2^d \sum_{k=1}^{n_k} (-1)^{k-1} \binom{n_k}{k} \int_0^{\frac{1}{k}} (1-kz)^{n_k-1} dz \\ &= 2^d \sum_{k=1}^{n_k} (-1)^{k-1} \binom{n_k}{k} \frac{1}{kn_k} \\ &= 2^d \frac{1}{n_k} \sum_{k=1}^{n_k} \frac{1}{k} \\ &= 2^d \frac{H_{n_k}}{n_k}, \end{aligned} \tag{36}$$

where $H_{n_k}$ is the $n_k$th harmonic number [36]. Approximating $H_n$ as $\ln n + \gamma$, where $\gamma \approx 0.5772156649\cdots$ is the Euler's constant [37], the expected value of $\Delta d$ is:

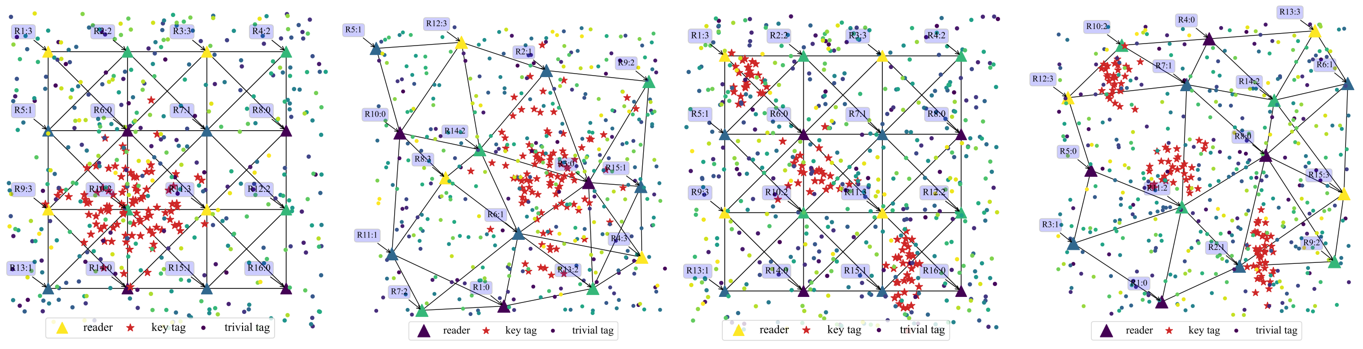
$$\Delta d = \lceil log_2 \lceil \frac{\ln n_k + \gamma}{n_k} \times 2^d \rceil \rceil. \tag{37}$$

We numerically optimize $d$ to maximize $\theta$ by incorporating the analyzed parameters into Eq. (3). Fig. 9 shows the relationship between $\theta$ and $d$ when $n_k = 1000$, $n_t = 10000$, $n_k^l = 800$, and $n_t^l = 5000$. The optimal $d$ is set to the maximum $\theta$, with its initial value set to $\min(d)$. ■

## VII. Performance Evaluation

### A. Simulation Configurations

*1) Parameter Setting:* Similar to the existing work [24], [38]–[40], the parameter settings follow the specifications of the C1G2 standard [32]. In C1G2, the tag-to-reader transmission rate ranges from 40kbps to 640kbps when FM0 is used and 5kbps to 320kbps when Miller is used. We take the lower bound 40kbps in the intersection set 40kbps to 320kbps as the tag-to-reader data rate. Similarly, the reader-to-tag data rate varies from 26.7kbps to 128kbps and we adopt 26.7kbps. Any consecutive communications between the reader and tags are separated by different time intervals. When the reader transmits commands, tags wait for time $T1$ before replying. After receiving replies from the tags, the reader waits for time $T2$ before further communication. In our simulation, we set $T1 = 100\mu s$ and $T2 = 50\mu s$, which comply with the C1G2 standard. Therefore, $t_{ID} = 37.45 \times 96 = 3595.2\mu s$, $t_s = (25 \times 1 + 150) = 175\mu s$, and $t_{ack} = 37.45 \times 2 = 74.9\mu s$. There is only one time interval between a tag transmission and a corresponding reader ACK. We use Python-based simulator to generate reader-tag deployment data, which is then fed in a Matlab-based simulator to compare the performance of our protocols with the exiting tag distribution identification protocol IB [27], the leading polling method IPP [30], and the state-of-the-art tag anti-collision protocol EUTI [31] across various scenarios.



(a) Regular readers & Uniform trivial tags and Gaussian key tags (b) Random readers & Uniform trivial tags and Gaussian key tags (c) Regular readers & Uniform trivial tags and multi-Gaussian key tags (d) Random readers & Uniform trivial tags and multi-Gaussian key tags

Fig. 11: Gaussian key tag deployment with grid and random readers.

*2) Readers & Tags Deployment Strategy:* We evaluate our protocols on RFID systems with various reader (grid, random) and tag (Uniform, Gaussian) deployment strategies. First, we establish a baseline scenario with uniformly distributed tags, where trivial and key tags are randomly placed across the entire area, as shown in Fig. 10(a) and Fig. 10(b). Second, we model typical real-world scenarios where key tags correspond to valuable goods of the same type or owned by the same merchant. These items are often stored together in one or more locations. To simulate such non-uniform distribution, we generate one or more Gaussian distributions with means ranging from 0 to 1 and variances from 0 to 0.1, from which the horizontal and vertical coordinates of key tags are independently sampled, as depicted in Fig. 11.

Additionally, to model potential reader interference, we employ an undirected graph, with edges representing neighboring readers [24], [27]. In a multi-reader RFID system, the required number of readers to completely cover the surveillance area depends on each reader's communication radius, which is set by the reader's power. To investigate the impact of reader coverage, we vary the communication radius of the readers in a unit space while maintaining a minimum number of readers for full coverage. For example, when the communication radius of the reader is set to 0.18 and 0.12, approximately 23 and 48 readers are required, respectively.

## *B. Time Efficiency*

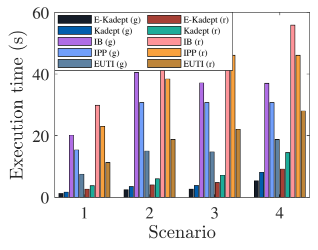
In the following simulations, we evaluate the time efficiency of Kadept in RFID systems with different parameter settings. In Fig. 12, Fig. 13, and Fig. 14, we compare the execution time of Kadept with the IB, IPP, and EUTI across Uniform, Gaussian, and multi-Gaussian distribution systems, as shown in Fig. 10(a) and Fig. 10(b), Fig. 11(a) and Fig. 11(b), Fig. 11(c) and Fig. 11(d), respectively. In addition, to identify key tags correctly and completely, we eliminate the negative interference and clock difference caused by wave superpose [40], which is exploited in IB. Hence, in IB, we let tags respond to readers with one bit in a slot according to the specifications of C1G2 standard. For EUTI, it is designed for unknown tag identification (identifying newly added tags), whereas our work focuses on determining the distribution of a subset of known tags, pursuing fundamentally different goals. To adapt EUTI for our problem, we treat its known and unknown tags as our trivial and key tags, respectively. For a fair comparison, we modify its identification phase to allow each key tag to transmit only a single bit instead of its full ID. Additionally, we adjust EUTI's deactivation phase to prevent silent tags (those mapped to singleton slots) from transmitting RN16, thereby significantly reducing time overhead in our scenario.

In Fig. 12(a), Fig. 13(a), and Fig. 14(a), we comprehensively compare the execution time of E-Kadept, Kadept, IB, IPP, and EUTI under four distinct scenarios. For clarity, we use $m_g$ and $m_r$ to represent the number of readers in regular and random reader deployment scenarios, respectively. Moreover, we define tag density ($\rho$) as the ratio of total tags to the number of readers, and key tag ratio ($\kappa$) as the ratio of key tags to total tags. Thus, $\rho = \frac{x+y}{m_g(m_r)}$ and $\kappa = \frac{x}{x+y}$, where $x$ and $y$ are the number of key tags and trivial tags, respectively. In scenario 1, we set $x + y = 10,000$, $\kappa = 0.1$, $m_g = 16$, and $m_r \approx 23$. Without changing other parameters, we double the tag size $x + y$ in scenario 2, i.e., $x + y = 20,000$, $\kappa = 0.1$, $m_g = 16$, and $m_r \approx 23$. In scenario 3, we increase the number of readers to $m_g = 36$ and $m_r \approx 48$, by adjusting the communication radius. In turn, we double $\kappa$ in scenario 4. To examine the execution time of our protocols, we take a closer look at scenario 4 in Fig. 12(a), Fig. 13(a), and Fig. 14(a).

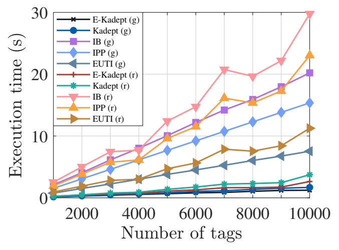
In Fig. 12(a), when key tags follow a Uniform distribution, IB takes 37s (55.9s) in a grid (random) reader deployment. IPP reduces the execution time to 30.7s (46s), and EUTI further decreases it to 18.7s (28s). Kadept makes a significant advancement by filtering trivial tags and assigning a unique slot to each key tag for replying, which substantially shortens the execution time to 8.1s (14.5s). E-Kadept achieves the best performance, reducing the execution time to 5.3s (9.2s) by leveraging an incremental Cuckoo filter design.

In Fig. 13(a), when key tags follow a Gaussian distribution, IB takes 39s (48.4s) with grid (random) reader deployment. IPP reduces the execution time to 30.7s (38.4s), and EUTI further reduces it to 20.9s (26s). Kadept substantially improves performance to 9.6s (14.4s). E-Kadept outperforms all protocols, achieving 6.1s (9.2s).

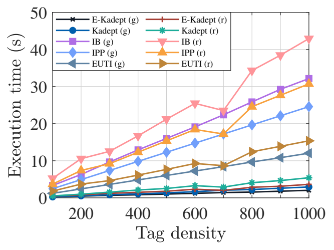
In Fig. 14(a), the execution time of all protocols fluctuates across different cases but generally trends higher than in the Uniform and Gaussian scenarios due to uneven reader loads from the multi-Gaussian distribution of key tags. Despite these variations, E-Kadept consistently outperforms Kadept, EUTI, IPP, and IB. This performance trend holds across all scenarios:



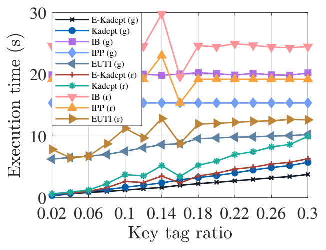
(a)



(b)

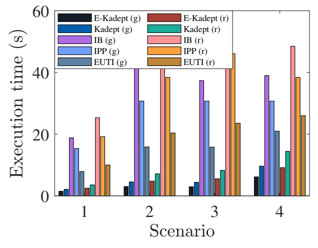


(c)

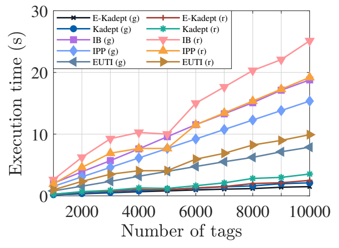


(d)

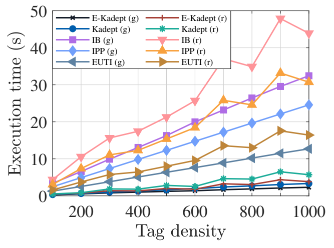
Fig. 12: Execution time with Uniform key tag distribution.



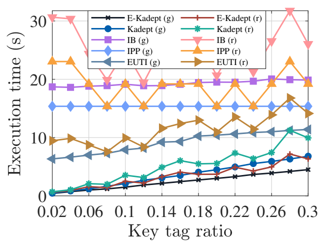
(a)



(b)

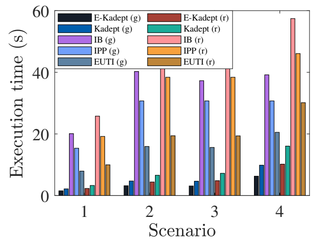


(c)

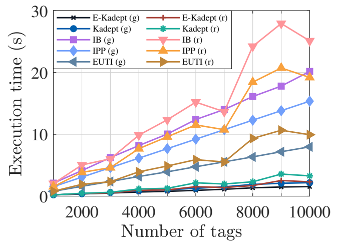


(d)

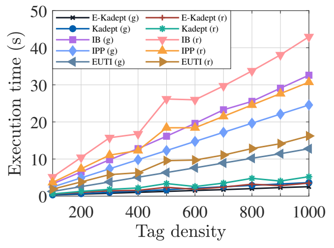
Fig. 13: Execution time with Gaussian key tag distribution.



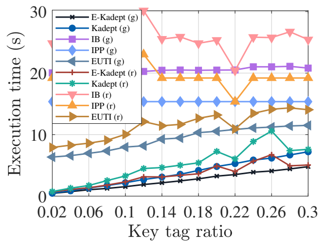
(a)



(b)



(c)



(d)

Fig. 14: Execution time with multi-Gaussian key tag distribution.

E-Kadept performs the best, Kadept follows, then EUTI, with IPP and IB being the least efficient.

Now, we study the impact of system scale, tag density, and key tag ratio on the execution time of E-Kadept, Kadept, IB, IPP, and EUTI in the Uniform and Gaussian distribution systems.

**The Uniform Distribution System.** First, we study the impact of the system scale on the execution time of the four protocols. In Fig. 12(b), we fix $\kappa = 0.1$ and vary $x + y$ from 1000 to 10,000 with a step size of 1000. The minimum number of readers required to cover all tags may vary in random reader deployment scenarios. As $x+y$ increases, the execution time of E-Kadept, Kadept, IB, IPP, and EUTI generally increases and fluctuates. This is because the time required for these protocols depends not only on the number of tags but also on the degree of parallelism among readers. A larger $x + y$ requires these protocols to handle more tags, naturally increasing the time. Additionally, an increased number of schedules reduces reader parallelism, further adding to the execution time. For example, in random scenarios, when $x+y$ is 7000 and 8000, the numbers of schedules are 6 and 5, respectively, these protocols take longer to handle 7000 tags than 8000 tags. The higher the reader parallelism, the less the execution time.

In Fig. 12(c), we investigate how the tag density influences the execution time of E-Kadept, Kadept, IB, IPP, and EUTI. With 16 fixed readers, we vary the total number of tags $x + y$ from 1600 to 16,000, corresponding to the tag density $\rho$ ranging from 100 to 1000 in steps of 100. As expected, the execution time of these protocols increases as $\rho$ rises due to the increased tag load per reader. In random reader deployments, fluctuations occur due to variations in reader parallelism.

Fig. 12(d) shows the impact of the key tag ratio on execution time. We fix $x+y = 10,000$ and vary $\kappa$ from 0.02 to 0.3 with a step size of 0.02. The execution time of E-Kadept, Kadept, and EUTI increases with $\kappa$, while IB and IPP remain stable as they handle all tags. This is because E-Kadept, Kadept and EUTI require more bit transmission for the filter to deactivate trivial tags and more response bits from key tags as $\kappa$ increases. Nevertheless, our protocols still outperform IB, IPP, and EUTI, especially when $\kappa$ is small, a scenario for which our protocols were specifically designed. Furthermore, E-Kadept, with its efficient incremental Cuckoo filter, shows less performance degradation than Kadept as $\kappa$ increases.

**The Gaussian Distribution System.** Fig. 13(b) illustrates the execution time of E-Kadept, Kadept, IB, IPP, and EUTI in relation to the system scale. We vary $x + y$ from 1000 to 10,000 with a step length of 1000. Similar to Fig. 12(b), the execution time of all protocols increases with $x + y$; however, in most cases, E-Kadept, Kadept, IB, and EUTI take more time in the Gaussian distribution systems. This is due to the uneven tag distribution across readers, with those covering fewer tags finishing earlier but having to wait for others, thus wasting time. For IPP, its execution time does not change when $x + y$ and the number of schedules are fixed. Therefore, it performs stably in grid reader scenarios but fluctuates in random ones.

In Fig. 13(c), we vary the tag density $\rho$ from 100 to 1000 with a step length of 100. As $\rho$ increases, the execution time of E-Kadept, Kadept, EUTI, IPP, and IB generally increases. In regular reader deployments, the execution time of all protocols increases steadily with $\rho$. In random reader deployments, the execution time of all protocols fluctuates due to the combined effects of reader parallelism and individual reader load. For instance, when $\rho = 700$ and $\rho = 1000$, both use 5 schedules. The higher tag density at $\rho = 1000$ leads to more time for all protocols. However, at $\rho = 900$, the reduced reader parallelism to 6 schedules increases the time compared to $\rho = 1000$.

The parameter settings in Fig. 13(d) are the same as in Fig. 12(d). The execution time of E-Kadept, Kadept, IB, IPP, and EUTI varies in random reader scenarios, while IB and IPP remain stable in regular ones. Despite the increasing trend in execution time with $\kappa$, our protocols, particularly E-Kadept, exhibit better performance than IB, IPP, and EUTI.

**The multi-Gaussian Distribution System.** The parameter settings here remain consistent with the Uniform and Gaussian distribution systems. With key tags distributed across multiple Gaussian distributions, readers covering key tag hotspots experience increased load while other readers maintain relatively balanced coverage, affecting protocol performance differently as illustrated in Fig. 14(b), Fig. 14(c), and Fig. 14(d).

From Fig. 13(b) and Fig. 14(b), we observe that the more uneven tag distribution in the multi-Gaussian system leads to greater variability in reader scheduling. For example, the number of schedules varies with the total number of tags: 4 at 7000, 6 at 8000 and 9000, and 5 at 10,000. This results in more execution time fluctuations of all protocols. When $x + y$ is 8000 and 9000, higher schedule counts further increase the execution time of all protocols.

In Fig. 14(c), the execution time of all protocols increases with tag density and decreases with higher reader parallelism. For example, at tag densities of 400 and 500, the number of schedules increases from 5 to 6, respectively, leading to more time at $\rho = 500$. When the number of schedules remains at 5, increasing tag density from 600 to 1000 raises the per-reader tag load (1232, 1518, 1838, 1962, and 2110, respectively), resulting in more time for EUTI, IB, and IPP. Furthermore, the key tag distribution also affects the execution time of our protocols. For example, even with increasing overall tag density, a lower average number of key tags per reader, from 188 at $\rho = 800$ to 167 at $\rho = 900$, reduces execution time.

In Fig. 14(d), as $\kappa$ increases, the execution time of IB and IPP remains stable in regular reader scenarios but fluctuates in random scenarios due to varying reader parallelism and load. E-Kadept and Kadept, however, are influenced by both $\kappa$ and reader load, leading to fluctuations in both deployments. For example, the number of schedules is 6 at $\kappa = 0.12$, 4 at $\kappa = 0.22$, and 5 at other $\kappa$ values, which impacts the time fluctuations of all protocols. Moreover, the higher average tags per reader at $\kappa = 0.26$ (904) compared to $\kappa = 0.3$ (778) further increases the time. Despite the increase in the time as $\kappa$ rises, our protocols still outperform EUTI, IPP, and IB. For example, at $\kappa = 0.1$ and $\kappa = 0.3$, our E-Kadept achieves $3.3\times$ and $1.75\times$ performance gains over the state-of-the-art EUTI, reducing the execution time from 10s to 2.3s and from 14s to 5.1s, respectively.

### C. Discussion: Adaptability to Dynamic Environments

Simulation results show that our protocols perform well in static RFID systems under various reader and tag deployment strategies, where the tag set and distribution are fixed.

In real-world scenarios, however, tags may dynamically enter, leave, or move within the system. Given our protocols' rapid execution times (under 30 seconds for dealing with 10,000–20,000 tags), the system can be treated as static for individual runs if tag sets and distributions remain stable during each execution. Nevertheless, we acknowledge that tag movement during execution may lead to two impacts: (1) changes in per-reader tag counts may result in suboptimal parameters, but if only trivial tags move, the distribution of key tags remains unaffected; (2) if key tags relocate across reader regions, the distribution may change. In such cases, re-executing Kadept and E-Kadept can re-identify the current key tag distribution. We maintain that our protocols are well-suited for scenarios with low to moderate tag mobility, where the system can be considered static for several minutes, a condition achievable in many real-world applications.

## VIII. RELATED WORK

Tag identification is a fundamental problem for RFID systems. Its objective is to collect all the tags' IDs under the reader's coverage in an efficient way. Since the tags can only communicate with the reader and cannot self-regulate their radio transmissions, the key issue for tag identification is to avoid tag-to-tag collisions. Prior work on the tag identification falls into two categories: Aloha-based [26], [41], [42] and tree-based [25], [43]. The former is to let each tag select a time slot to transmit its ID, and the tag ID can be successfully transmitted to the reader when a slot is selected by only one tag. In the tree-based protocols, the collided tags are continuously divided into two subsets until only at most one tag in each set.

In recent years, research has shifted towards collecting functional RFID data. For instance, tag searching [44] focuses on determining whether a group of interested tags is present in a given tag set; multi-group tag searching [38] aims to concurrently search multiple groups of selected tags using a group-based filtering mechanism; key tag tracking [39] is to track the number of key tags in multi-tenant systems; missing key tag identification [22], [23] aims to identify whether and which key tags are absent; information collection [24], [40] targets at collecting the tagged product's information or the sensing data of sensor-augmented tags; unknown tag identification [31] focuses on identifying newly added tags by eliminating interference from previously known tags.

Identifying the distribution of key tags enables efficient collection of their functional data, crucial for various inventory operations. Although advanced tag identification protocols like TagMap [45] and Tash [46] can rapidly collect all tag IDs, our focus is figure out the key tags' distribution. Collecting data from non-target tags incurs unnecessary overhead, and collecting IDs from all tags may pose a privacy risk in privacy-sensitive applications. While the tag distribution identification protocol IB [27] and the tag polling protocol IPP [30] can identify key tags' distribution without leaking privacy, they both suffer from long time delays in dealing with all tags. We therefore propose E-Kadept and Kadept, which identify key tag distribution by designing novel filters to eliminate interference from trivial tags and assign a unique slot for each key tag, while ensuring key tag privacy.

## IX. Conclusion

In this paper, we study the problem of key tag distribution identification in large-scale RFID systems. Two protocols, E-Kadept and Kadept, are proposed to fast identify the distribution of all key tags and separate key tags from trivial tags. By ingeniously constructing the Cuckoo filter, Kadept is able to assign a unique, continuous slot for each key tag when it filters trivial tags. By checking the statuses of assigned slots, readers can identify which key tags are under their own coverage. E-Kadept accelerates the identification process by design an incremental Cuckoo filter that reduces false positives and improves space efficiency. Extensive simulations show good performance of our protocols.

## References

[1] Y. Wang, J. Liu, S.-H. Lyu, Z. Qu, B. Tang, and B. Ye, "Identifying key tag distribution in large-scale RFID systems," in *Proc. of IEEE/ACM IWQoS*. IEEE, 2024, pp. 1–10.

[2] A. D. Smith, A. A. Smith, and D. L. Baker, "Inventory management shrinkage and employee anti-theft approaches," *International Journal of Electronic Finance*, vol. 5, no. 3, pp. 209–234, 2011.

[3] J. Yu, L. Chen, R. Zhang, and K. Wang, "Finding needles in a haystack: Missing tag detection in large RFID systems," *IEEE transactions on communications*, vol. 65, no. 5, pp. 2036–2047, 2017.

[4] S. Li, S. Li, M. Chen, C. Song, and L. Lu, "Frequency scaling meets intermittency: Optimizing task rate for RFID-scale computing devices," *IEEE Transactions on Mobile Computing*, vol. 23, no. 2, pp. 1689–1700, 2023.

[5] X. Chen, J. Liu, H. Huang, Y.-E. Sun, X. Zhang, and L. Chen, "Revisiting cardinality estimation in cots RFID systems," in *Proc. of ACM MobiCom*, 2023, pp. 1–14.

[6] X. Liu, Y. Huang, Z. Xi, J. Luo, and S. Zhang, "An efficient RFID tag search protocol based on historical information reasoning for intelligent farm management," *ACM Transactions on Sensor Networks*, 2023.

[7] M. Jin, K. Li, X. Tian, X. Wang, and C. Zhou, "Graph based RFID grouping for fast and robust inventory tracking," *IEEE Transactions on Mobile Computing*, 2024.

[8] K. Lin, H. Chen, N. Yan, Z. Ni, Z. Wang, and J. Yu, "Double polling-based tag information collection for sensor-augmented RFID systems," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 3496–3509, 2023.

[9] J. Liu, X. Chen, S. Chen, X. Liu, Y. Wang, and L. Chen, "TagSheet: Sleeping Posture Recognition with an Unobtrusive Passive Tag Matrix," in *Proc. of IEEE INFOCOM*, 2019, pp. 874–882.

[10] Y. Wang and Y. Zheng, "TagBreathe: Monitor Breathing with Commodity RFID Systems," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 969–981, 2020.

[11] S. Zhang, Z. Ma, K. Lu, X. Liu, J. Liu, S. Guo, A. Y. Zomaya, J. Zhang, and J. Wang, "Hearme: Accurate and real-time lip reading based on commercial RFID devices," *IEEE Transactions on Mobile Computing*, vol. 22, no. 12, pp. 7266–7278, 2022.

[12] J. Liu, J. Yu, D. Niyato, R. Zhang, X. Gao, and J. An, "Covert ambient backscatter communications with multi-antenna tag," *IEEE Transactions on Wireless Communications*, vol. 22, no. 9, pp. 6199–6212, 2023.

[13] J. Zhang, X. Liu, S. Chen, X. Tong, Z. Deng, T. Gu, and K. Li, "Toward robust RFID localization via mobile robot," *IEEE/ACM Transactions on Networking*, vol. 32, no. 4, pp. 2904 – 2919, 2024.

[14] Z. An, Q. Lin, P. Li, and L. Yang, "General-purpose deep tracking platform across protocols for the internet of things," in *Proc. of IEEE MobiSys*, 2020, pp. 94–106.

[15] G. Wang, C. Qian, L. Shangguan, H. Ding, J. Han, K. Cui, W. Xi, and J. Zhao, "Corrections to "hmo: Ordering RFID tags with static devices in mobile environments"," *IEEE Transactions on Mobile Computing*, vol. 20, no. 04, pp. 1746–1746, 2021.

[16] X. Liu, J. Zhang, S. Jiang, Y. Yang, K. Li, J. Cao, and J. Liu, "Accurate localization of tagged objects using mobile RFID-augmented robots," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1273–1284, 2021.

[17] B. Liang, P. Wang, R. Zhao, H. Guo, P. Zhang, J. Guo, S. Zhu, H. H. Liu, X. Zhang, and C. Xu, "Rf-chord: Towards deployable RFID localization system for logistic networks," in *Proc. of USENIX NSDI*, 2023, pp. 1783–1799.

[18] W. Gong, H. Wang, S. Li, and S. Chen, "Glac: High-precision tracking of mobile objects with cots RFID systems," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2331 – 2343, 2024.

[19] M. I. Ahmed, A. Bansal, K. Yuan, S. Kumar, and P. Steenkiste, "Battery-free wideband spectrum mapping using commodity RFID tags," in *Proc. of ACM MobiCom*, 2023, pp. 1–16.

[20] Y. Zhu and Q. Zhang, "Loprint: Mobile authentication of RFID-tagged items using cots orthogonal antennas," in *Proc. of IEEE INFOCOM*. IEEE, 2024, pp. 1551–1560.

[21] X. Liu, B. Zhang, S. Chen, X. Xie, X. Tong, T. Gu, and K. Li, "A wireless signal correlation learning framework for accurate and robust multi-modal sensing," *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 9, pp. 2424 – 2439, 2024.

[22] J. Yu, W. Gong, J. Liu, L. Chen, F. Wang, and H. Pang, "Practical key tag monitoring in RFID systems," in *Proc. of IEEE/ACM IWQoS*, 2018, pp. 1–2.

[23] H. Chen, Z. Wang, F. Xia, Y. Li, and L. Shi, "Efficiently and completely identifying missing key tags for anonymous RFID systems," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2915–2926, 2017.

[24] X. Liu, J. Yin, S. Zhang, B. Xiao, and B. Ou, "Time-efficient target tags information collection in large-scale RFID systems," *IEEE Transactions on Mobile Computing*, vol. 20, no. 9, pp. 2891–2905, 2020.

[25] L. Zhang, W. Xiang, and X. Tang, "An efficient bit-detecting protocol for continuous tag recognition in mobile RFID systems," *IEEE Transactions on Mobile Computing*, vol. 17, no. 3, pp. 503–516, 2018.

[26] J. Su, A. X. Liu, Z. Sheng, and Y. Chen, "A partitioning approach to RFID identification," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2160–2173, 2020.

[27] F. Zhu, B. Xiao, J. Liu, B. Wang, Q. Pan, and L.-J. Chen, "Exploring tag distribution in multi-reader RFID systems," *IEEE Transactions on Mobile Computing*, vol. 16, no. 5, pp. 1300–1314, 2017.

[28] K. Bu, M. Xu, X. Liu, J. Luo, S. Zhang, and M. Weng, "Deterministic detection of cloning attacks for anonymous RFID systems," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1255–1266, 2015.

[29] J. Liu, J. Yu, X. Chen, R. Zhang, S. Wang, and J. An, "Covert communication in ambient backscatter systems with uncontrollable RF source," *IEEE Transactions on Communications*, vol. 70, no. 3, pp. 1971–1983, 2022.

[30] J. Liu, B. Xiao, X. Liu, K. Bu, L. Chen, and C. Nie, "Efficient polling-based information collection in RFID systems," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 948–961, 2019.

[31] C. Chu, J. Niu, W. Zheng, J. Su, and G. Wen, "A time-efficient protocol for unknown tag identification in large-scale RFID systems," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13 024–13 040, 2021.

[32] "EPC radio-frequency identity protocols generation-2 UHF RFID standard," GS1, ISO/IEC 18000-63, Jul. 2024, https://www.gs1.org/standards/epc-rfid/uhf-air-interface-protocol.

[33] B. Sheng, Q. Li, and W. Mao, "Efficient continuous scanning in RFID systems," in *Proc. of IEEE INFOCOM*, 2010, pp. 1–9.

[34] J. Waldrop, D. W. Engels, and S. E. Sarma, "Colorwave: an anticollision algorithm for the reader collision problem," in *Proc. of IEEE ICC*, vol. 2, 2003, pp. 1206–1210.

[35] H. A. David and H. N. Nagaraja, *Order statistics*. John Wiley & Sons, 2004.

[36] D. E. Knuth, *The art of computer programming*. Pearson Education, 1997, vol. 3.

[37] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics*. Addison-Wesley Professional, 1994, vol. 2.

[38] S. Zhang, X. Liu, S. Guo, A. Y. Zomaya, and J. Wang, "Why queue up? fast parallel search of RFID tags for multiple users," in *Proc. of ACM Mobihoc*, 2020, pp. 211–220.

[39] X. Liu, X. Xie, K. Li, B. Xiao, J. Wu, H. Qi, and D. Lu, "Fast tracking the population of key tags in large-scale anonymous RFID systems,"

*IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 278–291, 2016.

[40] J. Liu, S. Chen, Q. Xiao, M. Chen, B. Xiao, and L. Chen, “Efficient information sampling in multi-category RFID systems,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 159–172, 2018.

[41] J. Yu, P. Zhang, L. Chen, J. Liu, R. Zhang, K. Wang, and J. An, “Stabilizing frame slotted aloha-based iot systems: A geometric ergodicity perspective,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 3, pp. 714–725, 2020.

[42] S.-R. Lee, S.-D. Joo, and C.-W. Lee, “An enhanced dynamic framed slotted aloha algorithm for RFID tag identification,” in *Proc. of IEEE MobiQuitous*, 2005, pp. 166–172.

[43] J. Su, Z. Sheng, C. Huang, G. Li, A. X. Liu, and Z. Fu, “Identifying RFID tags in collisions,” *IEEE/ACM Transactions on Networking*, vol. 31, no. 4, pp. 1507 – 1520, 2022.

[44] J. Yu, W. Gong, J. Liu, L. Chen, and K. Wang, “On efficient tree-based tag search in large-scale RFID systems,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 42–55, 2018.

[45] Z. An, Q. Lin, L. Yang, W. Lou, and L. Xie, “Acquiring bloom filters across commercial RFIDs in physical layer,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1804–1817, 2020.

[46] Q. Lin, L. Yang, C. Duan, and Z. An, “Tash: Toward selective reading as hash primitives for gen2 RFIDs,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 819–834, 2019.

**Yanyan Wang** is an associate professor with the College of Computer and Information at Hohai University, Nanjing, China. Before that, she received the B.E. degree in Electronic Information Engineering from the PLA Information Engineering University, in 2011. She received the M.S. degree in Computer Science and Technology from Zhejiang University of Technology, in 2015. She received the Ph.D. degree in computer science and technology from Nanjing University, in 2020. Her research interests include RFID technologies.

**Jia Liu** is an associate professor with the Department of Computer Science and Technology at Nanjing University, Nanjing, China. Before that, he received the B.E. degree in software engineering from Xidian University, Xi’an, China, in 2010. He received the Ph.D. degree in computer science and technology from Nanjing University, Nanjing, China, in 2016. His research mainly focuses on RFID identification and passive sensing. He is a member of the IEEE, ACM, and CCF.

**Zhihao Qu** (Member, IEEE) received his B.S. and Ph.D. degrees in computer science from Nanjing University, Nanjing, China, in 2009, and 2018, respectively. He is currently an associate professor in the School of Computer Science and Software Engineering at Hohai University. His research interests are mainly in the areas of edge computing, knowledge distillation, and distributed machine learning.

**Shen-Huan Lyu** is an Assistant Researcher at the College of Computer Science and Software Engineering, Hohai University, China. He received the Ph.D. degree in computer science from Nanjing University in 2022. Before that, he received the B.Sc. degree in statistics from University of Science and Technology of China in 2017. His research interests include ensemble learning, learning theory, and optimization. He is a recipient of the Hong Kong Scholars (2024).

**Bin Tang** (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Nanjing University, Nanjing, China, in 2007 and 2014, respectively. He was an Assistant Researcher with Nanjing University from 2014 to 2020, and also a Research Fellow with The Hong Kong Polytechnic University, Hong Kong, in 2019. He is currently a Professor with Hohai University, Nanjing. His research interests include area of edge computing, network coding, and distributed machine learning.

**Baoliu Ye** (Member, IEEE) received the Ph.D. degree in computer science from Nanjing University, China, in 2004. He was a Visiting Researcher with The University of Aizu, Japan, from 2005 to 2006. He is currently a Full Professor at the Department of Computer Science and Technology, Nanjing University. He has published more than 100 articles in major conferences and journals. His current research interests mainly include distributed systems, cloud computing, and wireless networks.