Tash: Toward Selective Reading as Hash Primitives for Gen2 RFIDs

Qiongzheng Lin, *Member, IEEE, ACM*, Lei Yang, *Member, IEEE*, Chunhui Duan, *Student Member, IEEE*, and Zhenlin An, *Student Member, IEEE*

***Abstract*— Deployment of billions of commercial off-the-shelf (COTS) radio frequency identification (RFID) tags has drawn much of the attention of the research community because of the performance gaps of current systems. In particular, hash-enabled protocol (HEP) is one of the most thoroughly studied topics in the past decade. HEPs are designed for a wide spectrum of notable applications (e.g., missing detection) without need to collect all tags. HEPs assume that each tag contains a hash function, such that a tag can select a random but predictable time slot to reply with a one-bit presence signal that shows its existence. However, the hash function has never been implemented in COTS tags in reality, which makes HEPs a ten-year untouchable mirage. This paper designs and implements a group of analog on-tag hash primitives (called Tash) for COTS Gen2-compatible RFID systems, which moves prior HEPs forward from theory to practice. In particular, we design three types of hash primitives, namely, tash function, tash table function, and tash operator. All of these hash primitives are implemented through the selective reading, which is a fundamental and mandatory functionality specified in Gen2 protocol, without any hardware modification and fabrication—a feature allowing zero-cost fast deployment on billions of Gen2 tags. We further apply our hash primitives in one typical HEP application (i.e., missing detection) to show the feasibility and effectiveness of Tash. Results from our prototype, which is composed of one ImpinJ reader and 3000 Alien tags, demonstrate that the new design lowers 70% of the communication overhead in the air. The tash operator can additionally introduce an overhead drop of 29.7%.**

***Index Terms*— RFID, hash function, hash table function, EPCglobal Gen2.**

I. Introduction

RFID systems are increasingly used in everyday scenarios, which range from object tracking, indoor localization [1], vibration sensing [2], to medical-patient management, because of the extremely low cost of commercial RFID tags (e.g., as low as 5 cents per tag). Recent reports show that many industries like healthcare and retailing are moving towards deploying RFID systems for object tracking, asset monitoring, and emerging Internet of Things [3]. The Electronic Product Code global is an organization established to accomplish the worldwide adoption and standardization of EPC technology. It published the Gen2 air protocol [4] for RFID system in 2004. A Gen2 RFID system consists of a reader and many passive tags. The passive tags without batteries are powered up purely by harvesting radio signals from readers. This protocol has become the mainstream specification globally, and has been adopted as a major part of the ISO/IEC 18000-6 standard.

Embedding Gen2 tags into everyday objects to construct ubiquitous networks has been a long-standing vision. However, a major problem that challenges this vision is that the Gen2 RFID system is not efficient [5]. First, the RFID system utilizes simple modulations (e.g., ON-OFF keying or BPSK) due to the lack of traditional transceiver [6], which prevents tags from leveraging a suitable channel to transmit more bits per symbol and increase the bandwidth efficiency. Second, tags cannot hear the transmissions of other tags. They merely reply on the reader to schedule their medium access with the Framed Slotted ALOHA protocol, which results in many empty and collided slots. This condition also retards the inventory process. These two limitations force a reader to go through a long inventory phase when it collects all the tags in the scene.

Hash Enabled Protocol: Motivated by the aforementioned performance gaps, the research community opened a new focus on HEP design approximately 10 years ago. The key idea that underlies HEP is that each tag selects a time slot according to the hash value of its EPC and a random seed. It then replies a one-bit presence signal rather than the entire EPC number in the selected slot. HEPs treat all tags as if they were a virtual sender, which outputs a gimped hash table (i.e., a *presence bitmap*) when responding to a challenge (i.e., a random seed). Most importantly, HEPs assume the backend server and every tag share a *hash function*, and the resulting bitmap is random but predicable when the EPCs and seeds are known. To better understand HEP, Fig. 1 shows a toy example with $n = 8$ tags, each of which contains a unique EPC number presented in binary format (e.g., 101010_2), to illustrate the HEP concept. The reader divides the time into d time slots (e.g., $d = 8$.) and challenges these tags with the random seed r. Each tag selects the $(h_d(\texttt{EPC}, r))^{th}$ time slot to reply

Manuscript received December 20, 2017; revised August 22, 2018; accepted January 25, 2019; approved by IEEE/ACM Transactions on Networking Editor T. Spyropoulos. Date of publication February 14, 2019; date of current version April 16, 2019. This work was supported in part by the NSFC General Program under Grant 61572282, in part by University Grants Committee/Early Career Scheme under Grant 25222917, in part by the Shenzhen Basic Research Schema under Grant JCYJ20170818104855702, and in part by the Alibaba Innovative Research Program. *(Corresponding author: Lei Yang.)*

Q. Lin, L. Yang, and Z. An are with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: lin@tagsys.org; young@tagsys.org; an@tagsys.org).

C. Duan is with the School of Software, Tsinghua University, Beijing 100084, China (e-mail: duan@tagsys.org).

Digital Object Identifier 10.1109/TNET.2019.2896348

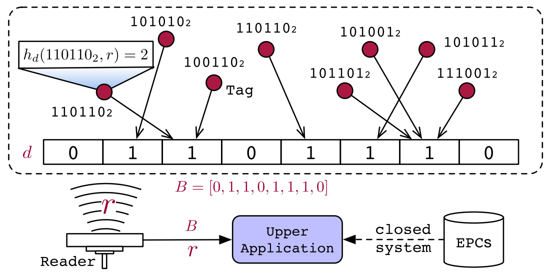


Fig. 1. Hash enabled protocol illustration. In the figure, 8 tags emit one-bit signals in the $h_d(\texttt{EPC}, r)^{th}$ time slots respectively, which are challenged by the random seed r and the frame length d. Finally, the reader abstracts tags' responses as a presence bitmap.

the one-bit signal, where $h(\cdot)$ is a common hash function (e.g., MD5, SHA-1) and $h_d(\cdot) = h(\cdot) \bmod d$. The reader can recognize two possible results for each time slot, namely, *empty* and *non-empty*. The reader abstracts the reply results into a bitmap (i.e., $B = [0, 1, 1, 0, 1, 1, 1, 0]$), where each element contains two possible values, that is, 0 and 1, that corresponds to empty and non-empty slots, respectively. The upper layer then utilizes this returned bitmap to explore many notable applications (see Sec. VIII).

HEP is advantageous in terms of speed and privacy. HEP is faster than all prior per-tag reading schemes for two reasons. First, collecting all the EPCs of the tags is time consuming because of the aforementioned low-rate modulation, whereas one-bit presence signals of HEPs save significantly communication time. Second, collisions are considered as one of the major reasons that drag down the reading. On the contrary, HEPs tolerate and consider collisions as informative. When privacy issues are considered, the tag's identification may be unacceptable in certain instances. HEPs allow tags to send out non-identifiable information (i.e., one-bit signals).

However, after 10 years of enthusiastic discussion about the opportunities that HEPs provide, the reality is beginning to settle: the functionality of hashing (i.e., hash function and hash table function) has never been implemented in any Gen2 RFID tags and considered by any RFID standard. No hint shows that this function will be widely accepted in the near future. The internal random functions inside tags seem to be an option. Actually, they fail to fulfill the requirement of upper application for two reasons: first, these random functions are inaccessible because no program API is provided in current COTS tags. Second, although the outputs of both hash function and random function distribute randomly, the random function is unpredictable and changes every time. The output of HEP is desired to be predicable such that the upper layer can test whether a known tag replies at an expect slot (see Sec. VII).

Why Is the Hash Function Unfavored?: A large number of recent work have attempted to supplement hash functionality to RFID tags, which can be categorized into three groups. First group, like [7] and [8], modifies the common hash functions to accommodate resource-constrained RFID tags. The second group [8]–[15] designs new lightweight and efficient hash functions dedicatedly for RFID tags. The third group seeks new design of RFID tags like WISP [16] and Moo [17], which gives tags more powerful computing capabilities

TABLE I
OVERVIEW OF CURRENT HASH FUNCTIONS [1]

Hash functions	Key size	GE	Power	Clock cycles
SHA-256 [7]	256	10,868	$15.87\mu A$	1,128
SHA-1 [7]	160	8,120	$10.68\mu A$	1,274
AES [19]	128	3,400	$8.15\mu A$	1,032
MAME [9]	256	8,100	$5.16\mu A$	96
MD5 [7]	128	8,400	-	612
MD4 [7]	128	7,350	-	456
PRESENT-80 [10]	80	1,570	-	32
PRESENT-80 [11]	80	1,075	-	563
PRESENT-128 [20]	128	1,886	-	32
DES [8]	56	2,309	-	144
mCrypton [12]	96	2,608	-	13
TEA [13]	128	2,355	-	64
HIGHT [14]	128	3,048	-	34
DESXL [8]	184	2,168	-	144
Grain & Trivium [15]	80	2,599	-	1

(e.g., hashing [18]). Unfortunately, as far as we know, none of these work has been really applied in COTS RFID systems yet.

A term called as *Gate Equivalent* (GE) is widely used to evaluate a hardware design with respect to its efficiency and availability. One GE is esquivalient to the area which is required by the two-input NAND gate with the lowest deriving strength of the corresponding technology. A glance at Table I shows the available designs of hash functions for RFID tags require a significant number of GEs, which are completely unaffordable by current COTS tags. For example, the most compact hash functions requires thousands of GEs (e.g., $1,075$ GEs for PRESENT-80), which incur extremely high energy consumption and manufacture cost. Thus, relatively few RFID-oriented protocols that appeal to a hash function can be utilized. RFID was expected to be one of the most competitive automatic identification technologies due to its many attractive advantages (e.g., simultaneous reading, NLOS, etc.) compared with others (e.g., barcode). However, this progress has been hindered for many years by the final obstacle that the industry is attempting to overcome (i.e., the price). The industry is extremely sensitive to the cost being doubled or tripled by the hash, although HEPs actually introduce significant outperformance.

Our Contributions: This work designs a group of hash primitives, *Tash*, which takes advantage of existing fundamental function of *selective reading* specified in Gen2 protocol, *without* any hardware modification and fabrication. Our design and implementation both strictly follow the Gen2 specification, so it can work in any Gen2-Compatible RFID system. These mimic (or analog) hash primitives act as we embedded real hash circuits on tags,[2] while we actually implement them in application layer. Specifically, we design the following three kinds of hash primitives to revive prior HEPs:

• We design a hash function (aka tash function) over existing COTS Gen2 tags. The hash function outputs a hash

[1] '-' means the algorithm is presented in theory and does not have specific power consumption.

[2] This work does not target at designing any analog circuit on readers or tags, but offers a mimic hash function acting as we embed a hash circuit on each tag.

value associated with the EPC of the tag and a random seed, as HEPs require.

• We design a hash table function (aka tash table function) over all tags in the scene. It can produce a hash table (aka tash table), which is more informative than a bitmap, over the all tags in the scene. In particular, each entry indicates the exact number of tags hashed into this entry.

• Major prior HEPs require multiple acquisitions of bitmaps to meet an acceptable confidence. We design three tash operators (i.e., tash AND, OR and XOR) to perform entry-wise set operations over multiple tash tables on tag in the physical layer, which offers a one-stop acquisition solution.

Summary: It has been considered that HEPs are hardly applied in practice because of the 'impossible mission' of implementing hash function on COTS Gen2 tags [20]. In this work, our main contribution lies in the practicality and usability, that is, enabling billions of deployed tags to benefit performance boost from prior well-studied HEPs, with our hash primitives. To the best of our knowledge, this is the first work to implement the hash functionality over COTS Gen2 tags. Second, we provide an implementation of Tash and show its feasibility and efficiency in two typical usage scenarios. Third, we investigate several leading RFID products in market including 18 types of tags and 10 types of readers, in terms of their compatibility with Gen2, and conduct an extensive evaluation on our prototype with COTS devices.

II. Motivation and Applications

To drive our key point, we conduct a brief survey on HEP applications. We list several key usage scenarios that we would like to support. Our objective is not to complete the list, but to motivate our design.

(1) Cardinality estimation. Estimating the size of a given tag population is required in many applications, such as privacy sensitive systems and warehouse monitoring. Kodialam and Nandagopal [21] presented a pioneer estimator. Given that tags select the time slots uniformly because of hashing, the expected number of '0's equals $n_0 = d(1-1/d)^n \approx de^{-n/d}$, i.e., n and d are the number of tags and frame length. Counting n_0 in an instance yields a "zero estimator", i.e., $\widehat{n} \approx -d\ln(n_0/d)$. For example, $\widehat{n} = -8 \times \ln(3/8) = 7.8$ in our toy example, as shown in Fig. 1. In the past decade, dozens of estimators [22]–[36] have been proposed. For example, Qian *et al.* [26] proposed an estimation scheme called lottery frame. Shahzad and Liu [28] estimated the number based on the average run-length of ones in a bit string received using the FSA. However, their schemes are limited to the application of cardinality estimation. By contrast, tash operate as a general solution that provides hash service to various applications including the estimation.

(2) Missing detection. Consider a major warehouse that stores thousands of apparel, shoes, pallets, and cases. How can a staff *immediately* determine if anything is missing? Tan *et al.* [37] conducted the early study on the fast detection of missing-tag events by using the presence bitmap. They assumed all EPCs were known in a closed system. Given that hash results are predicable, the system can generate an *intact* bitmap at the backend. We can identify the missing tags in a probabilistic approach by comparing the intact and instanced bitmaps. For example, if the second entry equals 0 (which is supposed to be 1), the the tag 101010_2 must be missing in our toy example. The missing detection problem was firstly mentioned in [37]. Thereafter, many follow-up works [38]–[52] have started to study the issue of false positives resulting from the collided slots by using multiple bitmaps. Additional details regarding this application are introduced in Sec. VII.

(3) Continuous reading. The traditional inventory approach starts from the beginning each time it interrogates all the tags, thereby making it highly time-inefficient. These works [53]–[55] have proposed continuous reading protocols that can incrementally collect tags in each step using the bitmap. For example, Sheng *et al.* [53] aimed to preserve the tags collected in the previous round and collect only unknown tags. Xie *et al.* [54] conducted an experimental study on mobile reader scanning. Liu *et al.* [55] initially estimated the number of overlapping tags in two adjacent inventories and then performed an effective incremental inventory.

(4) Data mining. These works [23], [56]–[59] discuss how to discover potential information online through bitmaps. For example, Sheng *et al.* [23] proposed to identify the popular RFID categories using the group testing technique. Xie *et al.* [56] found histograms over tags through a small number of bitmaps. Luo *et al.* [57], [58] determined whether the number of objects in each group was above or below a threshold. Liu *et al.* [59] proposed a new online classification protocol for a large number of groups.

(5) Tag searching. These works [60], [61] have studied the tag searching problem that aims to find wanted tags from a large number of tags using bitmaps in a multiple-reader environment. Zheng and Li [61] utilized bitmaps to aggregate a large volume of RFID tag information and to search the tags quickly. Liu *et al.* [60] first used the testing slot technique to obtain the local search result by iteratively eliminating wanted tags that were absent from the interrogation region.

(6) Tag polling. Qiao *et al.* [62], [63] and Li *et al.* [64] consider how to quickly obtain the sensing information from sensor-augmented tags. The system requires to assign a time slot to each tag using the presence bitmap. In summary, all the aforementioned HEP designs have allowed RFID research to develop considerably in the past decade. All the work can be boosted by our hash primitives.

III. A Primer on Gen2 Protocol

The Gen2 standard defines air communication between readers and tags. It adopts the reader-talks-first mode, in which the reader dominates communication and all the tags follow its commands. On the basis of [4] and [65], we introduce its four central functions (i.e., F1 $\sim$ F4) that we will employ:

F1 (Memory Model): Gen2 specifies a simple tag memory model [4, pp. 44–46]. Each tag contains four types of non-volatile memory blocks (called *memory banks*): (1) MemBank-0 is reserved for password associated with the tag. (2) MemBank-1 stores the EPC number. (3) MemBank-2 stores the TID that specifies the unchangeable tag and vendor

TABLE II
ACTIONS IN THE `Select` COMMAND

Action code	Tag matching	Tag not-matching
0	assert SL	deassert SL
1	assert SL	do nothing
2	do nothing	deassertSL
3	negate SL	do nothing
4	deassert SL	assert SL
5	deassert SL	do nothing
6	do nothing	assert SL
7	do nothing	negate SL

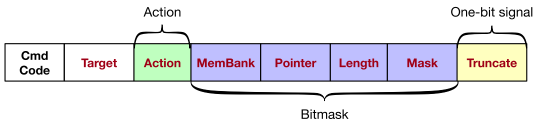


Fig. 2. The format of `Select` command. The command is composed of 8 fields, which are combined to select a subset of the tags based on a bitmask and specify their reply actions.

specific information. (4) `MemBank-3` is a customized storage that contains user-defined data.

F2 (Selective Reading): Gen2 specifies that each inventory must be started with `Select` command [4, pp. 72–73]. The reader can use this command to choose a subset of tags that will participate in the upcoming inventory round. In particular, each tag maintains a flag variable `SL`. The reader can use the `Select` command to turn the `SL` flags of tags into `asserted` (i.e., true) or `deasserted` (i.e., false). The `Select` command comprises six mandatory fields and one optional field apart from the constant cmd code (i.e., 1010_2), as shown in Fig. 2.

- `Target`. This field allows a reader to change `SL` flags or the inventoried flags of the tags. The inventoried flags are used when multiple readers are present. Such scenario is irrelevant to our requirements. Thus, we aim to change `SL` flags only by setting `Target` $= 100_2$.
- `Action`. This field specifies an action that will be performed by the tags. Table II lists eight action codes to which the tag makes different responses. For example, the matching or not-matching tags assert or deassert their `SL` flags when `Action` $= 0$. We leverage this useful feature to design tash operators.
- `MemBank`, `Pointer`, `Length` and `Mask`. These four fields are combined to compose a *bitmask*. The bitmask indicates which tags are matched or not-matched for an `Action`. The `Mask` contains a variable length binary string that should match the content of a specific position in the memory of a tag. The `Length` field defines the length of the `Mask` field in bits. The `Mask` field can be compared with one of the four types of memory banks in a tag. The `MemBank` field specifies which memory bank the `Mask` will be compared with. The `Pointer` field specifies the starting position in the memory bank where the `Mask` will be compared with. For example, if we use a tuple (b, p, l, m) to denote the four fields, then

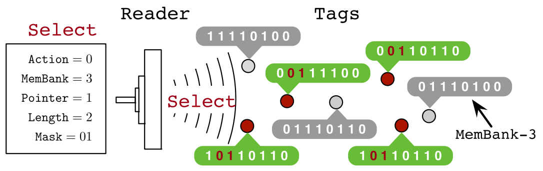


Fig. 3. Illustration of selective reading in Gen2. There are total 7 tags covered by a reader. The reader initiates a selective reading using a `Select` command with parameters: `Action` $= 0$, `MemBank` $= 3$, `Pointer` $= 1$, `Length` $= 1$, `Mask` $= 01$. This command means that these tags (highlighted with dark red) whose data starting at the second bit with a length of 2 bits in the `MemBank-3` equals 01_2 are selected to participate in the incoming inventory, while other tags (with gray color) that do not meet the condition remain silent. As a result, only 4 tags are collected in this round of inventory.

only the tags with data starting at the p^{th} bit with a length of l bits in the b^{th} memory bank that is equal to m are matched.

To visually understand the selective reading, we show an example in Fig. 3 in which 4 out of 7 tags are selected to participate in the incoming inventory. Complex and multiple subsets of tags can be facilitated by issuing a group of `Select` commands to choose a subset of tags before an inventory round starts. For example, we can issue two `Select` commands: one for division and another for one-bit reply. Note the `Truncate` enabled `Select` command must be the last one if multiple selection commands are issued [4].

F3 (Truncated Reply): Gen2 allows tags to reply a *truncated* reply (i.e., replying a part of `EPC`) through a special `Select` command with an enabled `Truncate` field, making a one-bit presence signal possible. When `Truncate` is enabled (i.e., set to 1), then the corresponding bitmask is not used for the division of tags, but lets tags reply with a portion of their `EPC`s following the pattern specified by the bitmask. Note that when `Truncate` is enabled, the `MemBank` must be set to the `EPC` bank (i.e., `MemBank` $= 1$) and such `Select` command must be the last one.

F4 (Query Model): Followed by a group of `Select` commands, `Query` command [4, pp. 76–80] starts a new *inventory round* over a subset of tags, chosen by the previous `Select` commands. There are 7 fields in the `Query` command. We only focus on `Sel` field, which is most tightly relevant to the selective reading. As mentioned above, the `Select` command has divided the tags into two opposite subsets with asserted and deasserted `SL` respectively. The `Sel` field specifies which subset will reply in the current inventory round. If `Sel` $= 11_2$, the tags with asserted `SL` reply. If `Sel` $= 10_2$, the tags with deasserted `SL` reply. We choose the tags with asserted `SL` by default.

IV. TASH DESIGN

Tash is a framework providing a group of hash primitives to Gen2 RFIDs. The proposed hash primitives are the tash function, the tash table function, and three kinds of tash operators (i.e., tash AND, OR and XOR). In this section, we present the technical designs of these primitives.

A. Design of the Tash Function

An l-bit *tash function* is actually a hash function $f_l(t, r) : \mathcal{T} \times \mathcal{R} \rightarrow 2^l$, where $\mathcal{T}$ and $\mathcal{R}$ are the domains of `EPC`s of the tags and random seeds. It outputs an l-bit integer v:

$$v = f_l(t, r) \tag{1}$$

We call l the *dimension* of tash function (i.e., $l = 0, 1, 2, \ldots$). The tash value v is an integer $\in [0, 2^l - 1]$.

A tash function is essentially a hash function that is indispensable to the presence bitmap. We design the tash function while following the three principles outlined as follows. The first principle requires that the tash result must be dependent on the input EPC and the seed. Moreover, it must be predictable as long as all the input parameters are known. The second principle requires the output values to be random, i.e., uniformly distributed in $[0, 2^l - 1]$. Even a one bit difference in the input will result in a completely different outcome. The third principle requires a method that can access the tash result of a tag directly or indirectly.

We have constructed the tash function as follows by applying the aforementioned principles: given a tag with an EPC of t, we firstly calculate the *hash value* of the EPC offline, using a common perfect hash function like 128-bit MD5 or SHA-1. Let $h(t)$ denote the calculated hash value. We then write $h(t)$ into the tag's user-defined memory bank of the tag, i.e., MemBank-3, for later use.

Definition 1 (Tash Value): The l-bit tash value of tag t challenged by seed r is defined as the value of the sub-bitstring starting from the r^{th} bit and ending at the $(r + l - 1)^{th}$ bit in the MemBank-3 of the tag.

The above definition formally presents the tash value. Evidently, $f_l(t, r)$ is actually a portion of $h(t)$, and thus, the parameter $r \in [0, \mathcal{L} - 1]$ and $l \in [1, \mathcal{L} - r]$, where $\mathcal{L}$ is the length of the hash value (e.g., 128 bits). Fig. 4 shows a toy example wherein the MemBank-1 and MemBank-3 of the tag store its EPC t and the hash value $h(t)$, respectively. When $r = 5$ and $l = 4$ are inputted, the tash value that this tag outputs is 1010_2, which is the sub-bitstring of $h(t)$ starting from the 5^{th} bit and ending at the 8^{th} bit in MemBank-3, i.e., $f_4(t, 5) = 1010_2$.

Our design does not require a tag to equip a real hash function or the engagement of its chip. It clearly applies the preceding principles. First, $f_l(t, r)$ is evidently repeatable, predicable and dependent on the inputs. Second, the randomness of $f_l(t, r)$ is derived from $h(t)$ and r, which are supposed to have a good randomness quality. Third, we have two ways to access the tash value. We can use the memory Read command to access MemBank-3 of a tag directly, or use the selective reading function to access the tash value indirectly (discussed later).

Discussion: A few points are worth-noting about the design:

• As the tash value is a portion of the hash value, if two random numbers may cover a common sub-string. For example, if r_1 and r_2 differ by 1, there exist $l - 1$ same bits with 50% of probability that two hash values are same, although such case occurs with a small probability, i.e., $\approx 127/(128 \times 128) \times 0.5 = 0.0039$. If some upper applications require extremely strong independence, we should generate the second random number r_2 meeting the condition of $r_2 < r_1 - l$ and $r_2 \geq r + l$, so as to avoid the common coverage and potential relevance.

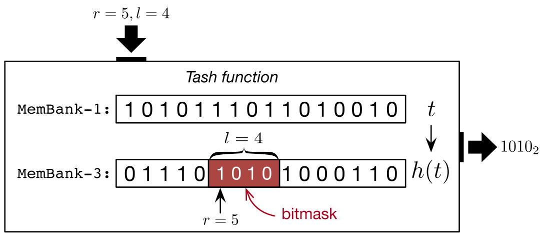


Fig. 4. Illustration of a tash function. Suppose the EPC of the tag is t, we pre-store the real hash value of t (i.e., $h(t)$) in its MemBank-3. The tash function fetches a substring of the real hash as its output. For example, suppose the input parameters are $(5, 4)$, then the result of tash function is equal to 1010_2, i.e., the substring within $[5, 9)$.

• The design of tash function involves the MemBank-3, i.e., the user-defined storage. We can use Write command to store any data into this memory bank. Our compatibility report (shown in Sec. VI) suggests that almost all types of tags support both MemBank-3 and Write command except one read-only type (i.e., ImpinJ Monza R6). Our approach is generally practicable.

• Our design targets at enabling COTS tags, billions of which have been deployed in recent years, to obtain performance advantages from well-studied hash based protocols, instead of enhancing their security or privacy preservation. Our design still follows the current COTS tag's security mechanism, i.e., password protected memory access.

• Tash function also offers a good feature that the computation is one way and irreversible, i.e., the output reveals nothing about the input. This feature is inherited from the hash function. It may be useful for privacy protection in practice. However, this topic is beyond the scope of this work.

B. Design of the Tash Table Function

We next introduce the design of tash table function, which is formally defined as below.

Definition 2 (Tash Table Function): An l-bit tash table function can assign each tag t from a given set into the i^{th} entry of a hash table (aka tash table) with a random seed r, where $i = f_l(t, r)$. Each entry of the tash table is the number of tags tashed into it.

Let B and $\mathcal{F}_l$ denote a tash table and a tash table function respectively. The tash table function takes a set of tags (i.e., $T = \{t_1, t_2, \ldots, t_n\}$) and a random number r as input and outputs a tash table B, denote by:

$$B = \mathcal{F}_l(T, r) \tag{2}$$

where $B[i] = |\{t | f_l(t, r) = i\}|$ (i.e., the number of tags tashed into the i^{th} entry) for $\forall t \in T$. Let $L = 2^l$, which is defined as the *size* of the tash table. The tash table function is the core function that HEPs expect. HEPs consider the reader as well as all tags as a virtual node equipping with tash table function. When inputing a random seed, the node would output a tash table. HEPs then utilize such table to provide various services (e.g., missing detection or cardinality estimation.). It worths noting that superior to the bitmap employed in prior HEPs, our tash table is a perfect table that contains the exact

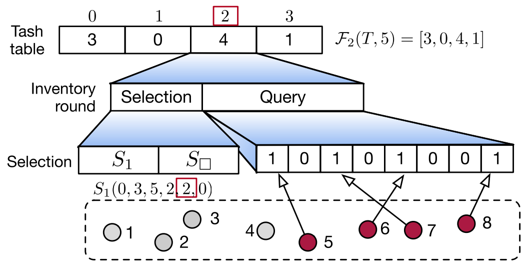


Fig. 5. Illustration of creating a tash table. Given that $r = 5$ and $l = 2$, $\mathcal{F}_2(T,5) = [3,0,4,1]$. Zooming into the 3rd entry-inventory, tags t_5, t_6, t_7 and t_8 are selected to join the inventory. $\square$ of $S_\square$ indicates the ending symbol.

number of tags tashed into each entry. Clearly, the table is completely backward compatible with prior HEPs because it can be forcedly converted into a presence bitmap.

Here, we leverage the selective reading (see Sec. III) to design the tash table function. For simplicity, we use

$$S(\underbrace{a}_{\texttt{Action}},\overbrace{b}^{\texttt{MemBank}},\underbrace{p}_{\texttt{Pointer}},\overbrace{l}^{\texttt{Length}},\underbrace{m}_{\texttt{Mask}},\overbrace{u}^{\texttt{Truncate}})$$

to denote a `Select` with an `Action` (a), a `MemBank` (b), a `Pointer` (p), a `Length` (l), a `Mask` (m) and a `Truncate` (u). The command aims to select a subset of tags with a sub-bitstring that starts from the p^{th} bit and ends at the $(p+l-1)^{th}$ bit in the b^{th} memory bank that is equal to m. These selected tags are requested to take an action a. The action codes are shown in Table. II. In particular, if $u = 1$, then each tag will reply with a truncated `EPC` number.

The tash table function is designed as follows. An l-bit table B consists of a total of 2^l entries, each of which contains the amount of tags mapped into it. In particular, the index number of each entry, which ranges from 0 to $2^l - 1$, is actually the tash values of the tags mapped into this entry, i.e., $B[\underline{i}] = |\{t|f_l(t,r) = \underline{i}\}|$. When constructing the i^{th} entry, the reader performs the selective reading with two selection commands as follows:

$$S_1(0,3,r,l,i,0) \quad \text{and} \quad S_\square(1,1,1,1,1,1)$$

Command S_1 selects a subset of tags with a sub-bitstring that starts from the r^{th} bit and ends at the $(r+l-1)^{th}$ bit in the `MemBank-3` that is equal to i. Notably, the involved sub-bitstring is the tash value of a tag, i.e., $f_l(t,r)$, which refers to Definition. 1. Consequently, only tags with tash values equal to i are selected to participate in the incoming inventory, i.e., counted by the i^{th} entry. The second command $S_\square$ enables the selected tags to reply with the first bit of their `EPC` numbers for the one-bit signals. We call such inventory round as an *entry-inventory*. In this manner, we can obtain the whole tash table by launching 2^l entry-inventories.

To visually understand the procedure, we illustrate an example in Fig. 5, where $r = 5$ and $l = 2$. The tash table contains 2^2 entries; hence, four entry-inventories are launched. Their

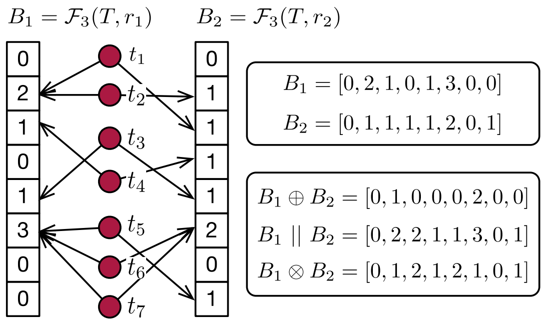

$B_1=\mathcal{F}_3(T,r_1)$
$B_2=\mathcal{F}_3(T,r_2)$
0 2 1 0 1 3 0 0
0 1 1 1 1 2 0 1
t_1 t_2 t_3 t_4 t_5 t_6 t_7
$B_1=[0,2,1,0,1,3,0,0]$
$B_2=[0,1,1,1,1,2,0,1]$
$B_1\oplus B_2=[0,1,0,0,0,2,0,0]$
$B_1 \,||\, B_2=[0,2,2,1,1,3,0,1]$
$B_1\otimes B_2=[0,1,2,1,2,1,0,1]$

Fig. 6. Illustration of tash operators. The left shows two independent tash tables, while the right shows the results of the two tash tables with tash AND, OR and XOR.

selection commands are defined as follows:

❶ $S_1(0,3,\underline{5},\underline{2},\underline{0},0)$ and $S_\square(1,1,1,1,1,1)$
❷ $S_1(0,3,\underline{5},\underline{2},\underline{1},0)$ and $S_\square(1,1,1,1,1,1)$
❸ $S_1(0,3,\underline{5},\underline{2},\underline{2},0)$ and $S_\square(1,1,1,1,1,1)$
❹ $S_1(0,3,\underline{5},\underline{2},\underline{3},0)$ and $S_\square(1,1,1,1,1,1)$

For the third entry-inventory, the `Mask` field is set to 2 because the index of the third entry is 2. 4 tags (i.e., t_5, t_6, t_7 and t_8) are selected to join in this entry-inventory. Thus, $\mathcal{F}_2(T,5)[2] = 4$.

For a tash table, note that (1) the sum of all its entries is equal to the total number of tags, and (2) it allows an application to selectively construct the entries of a tash table becaues each entry-inventory are independent of each other and completely controllable. For example, we can skip the inventories of these entries that are predicted to be empty.

C. Design of Tash Operators

Most prior HEPs adopt probabilistic ways and their results are guaranteed with a given confidence level. To meet the level, they usually combine multiple bitmaps, which are acquired through multiple rounds and challenged by different seeds. We abstract such combination into three basic tash operators, namely, tash AND, OR and XOR. These operators can comprise other complex operations. Let $B_1 = \mathcal{F}_l(T,r_1)$ and $B_2 = \mathcal{F}_l(T,r_2)$ denote two tash tables acquired twice with two different seeds, r_1 and r_2. Our objective is to obtain the final tash table B by performing one of the subsequent tash operators on B_1 and B_2.

Definition 3 (Tash AND): The tash AND (denoted by $\oplus$) of two tash tables is to obtain the intersection of two corresponding entry sets. Formally, $B = B_1 \oplus B_2$, where $B[i] = |\{t|f_l(t,r_1) = i \& f_l(t,r_2) = i\}|$.

Tash AND: The tash AND is aimed at obtaining the common intersection of corresponding entries from two tash tables. For example, as shown in Fig. 6, $B_1[1]$ and $B_2[1]$ count $\{t_1,t_2\}$ and $\{t_2\}$ respectively. However, $(B_1 \oplus B_2)[1] = |\{t_2\}| = 1$, which counts t_2 only. Let $B = B_1 \oplus B_2$, then each entry of B denotes the number of tags that are concurrently mapped into the corresponding entries of B_1 and B_2. The selection commands for the i^{th} entry-inventory are defined as follows:

$$S_1(\underline{0},3,r_1,l,i,0), S_2(\underline{2},3,r_2,l,i,0), S_\square$$

From the action codes shown in Table. II, the purpose of S_1 with action code of 0 is to select tags $\in B_1[i]$ and deselect tags $\notin B_1[i]$. S_2 with action code of 2 deselects tags $\notin B_2[i]$ and results in tags $\in B_2[i]$ doing nothing. After S_1 is received, each tag exhibits one of two states, i.e., selected or deselected. Then, S_2 will make the selected tags remain in their selected states if they match its condition (i.e., doing nothing); otherwise, it changes their states to the deselected states (i.e., selected $\rightarrow$ deselected), which is equivalent to removing tags $\notin B_2[i]$ from tags $\in B_1[i]$. Meanwhile, the tags deselected by S_1 remain in their states regardless of whether they match (i.e., do nothing) or not match (i.e., deselected $\rightarrow$ deselected) the condition of S_2. Finally, $S_\Box$ is reserved for the one-bit presence signal.

Definition 4 (Tash OR): The tash OR (denoted by $||$) of two tash tables is to merge two corresponding entry sets. Formally, $B = B_1||B_2$, where $B[i] = |\{t|f_l(t,r_1) = i|| f_l(t,r_2) = i\}|$.

Tash OR: The tash OR is aimed at obtaining the total number of tags mapped into the corresponding entries in two tash tables. Note tash OR is not the same as the entry-wise sum, i.e., $B_1||B_2 \neq B_1 + B_2$ because the tags twice mapped into a same entry are counted only once. As shown in Fig. 6, $(B_1||B_2)[5] = |\{t_5, t_6, t_7\}| = 3$ although $B_1[5] + B_2[5] = 5$ because t_6 and t_7 appear twice in the two tash tables. Let $B = B_1||B_2$, then each entry of B is the number of tags that mapped into the corresponding entry of either B_1 or B_2. The selection commands for the i^{th} entry-inventory are defined as follows:

$$S_1(\underline{0}, 3, r_1, l, i, 0), S_1(\underline{1}, 3, r_2, l, i, 0), S_\Box$$

Similarly, S_1 selects tags $\in B_1[i]$ and deselect tags $\notin B_1[i]$. S_2 with action code of 1 (see Table. II) allows tags $\in B_2[i]$ to be selected as well, but tags $\notin B_2[i]$ remain in their states (i.e., do nothing), some of these tags may have been selected by S_1. The process is equivalent to holding the tags selected by S_1 and incrementally adding the new tags selected by S_2.

Definition 5 (Tash XOR): The tash XOR (denoted by $\otimes$) is to remove the intersection of two corresponding entry sets from the first entry set. Formally, $B = B_1 \otimes B_2$ such that $B[i] = |\{t|f_l(t,r_1) = i \;\&\; f_l(t,r_2) \neq i\}|$.

Tash XOR: The tash XOR is aimed at obtaining the total number of the set difference. As Fig. 6 shows, $B_1[5] = |\{t_5, t_6, t_7\}|$ and $B_2[5] = |\{t_6, t_7\}|$. Then $(B_1 \otimes B_2)[5] = |\{t_5\}| = 1$. Let $B = B_1 \otimes B_2$, then each entry of B is the number of tags that are mapped into the corresponding entry of B_1 but not into the entry of B_2. The selection commands for the i^{th} entry-inventory are defined as follows:

$$S_1(\underline{0}, 3, r_1, l, i, 0), S_2(\underline{5}, 3, r_2, l, i, 0), S_\Box$$

Similarly, S_2 allows tags $\in B_2[i]$ to be deselected (i.e., removed from tags $\in B_1[i]$) and tags $\notin B_2[i]$ to do nothing. This process is equivalent to removing tags $\in B_2[i]$ from tags $\in B_1[i]$.

Tash Hybrid: The aforementioned three operators can be further applied to a hybrid operation. When k seeds (i.e., $r_1, \cdots, r_k$) are given, we can obtain k tash tables. The selection commands for the i^{th} entry-inventory can be designed as follows:

$$S_1(0, 3, r_1, l, i, 0), S_2(\texttt{AC}, 3, r_2, l, i, 0), \cdots, S_k(\texttt{AC}, 3, r_k, l, i, 0), S_\Box$$

where AC represents the `Action` code, which is set to 2, 1 and 5 for tash AND, OR and XOR, respectively. The action code of the first command is always set to 0. For example, the selection commands in the i^{th} entry-inventory for $\mathcal{F}_l(T,r_1) \oplus \mathcal{F}_l(T,r_2) || \mathcal{F}_l(T,r_3) \otimes \mathcal{F}_l(T,r_4)$ are given by:

$$S_1(\underline{0}, 3, r_1, l, i, 0), S_2(\underline{2}, 3, r_2, l, i, 0), S_3(\underline{1}, 3, r_3, l, i, 0), S_4(\underline{5}, 3, r_4, l, i, 0), S_\Box$$

We leverage the action of a selection command to perform an operation in the physical layer before an entry-inventory starts, therefore, we introduce minimal additional communication overhead, i.e., broadcasting multiple `Select` commands. Compared with the multiple acquisitions of bitmaps used by prior HEPs, our solution provides a one-stop solution that can significantly reduce the total overhead in such situation.

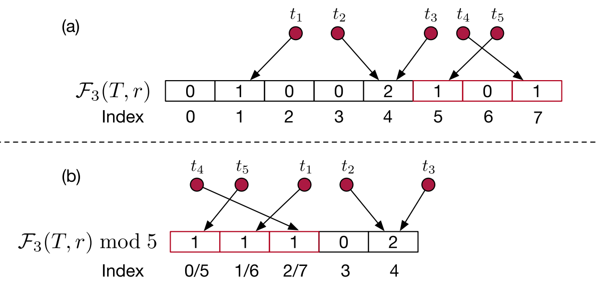


Fig. 7. Illustration of tash modulo. (a) shows the tash table of $\mathcal{F}_l(T,r)$; (b) shows the tash table of $\mathcal{F}_l(T,r) \bmod 5$ where the modulo operation moves tags in the entries which indexes (i.e., i^{th}) are greater than 4 to $(i \bmod 5)^{th}$ entries.

D. Design of Tash Modulo

From the above design, the size of a tash table must be the power of 2 (e.g., $2^l = 2, 4, 8, \cdots$), determined by the dimension of the table. In practice, the table size is a crucial performance parameter that we wish to optimize, as will be shown in Sec. VII: using larger table gives us more chances to reduce the false positives but is time-consuming. The optimal size of tash table should be carefully calculated. The current dimension-determined table size is not reasonable. For example, if the optimal size is 256, the real size must be set to $1024 = 2^{10} \leq 256$ for meeting the constraint of 2^l.

To address the above issue, we design the tash modulo operation to allow arbitrary table size, denoted by

$$\mathcal{F}_l(T,r) \bmod L$$

As illustrated in Fig. 7, the modulo operation moves the tags tashed at i^{th} entry (i.e., $i \geq L$) to the $(i \bmod L)^{th}$ entry. As a result, the i^{th} entry in the new tash table contains all tags whose tash values equal $i + k \times L$ where $k = 0, 1, 2, \ldots$. This is equivalent to performing the Tash OR. Thus, when constructing the i^{th} entry, the reader performs the selective reading with several commands as follows:

$$S_1(0,3,r,l,i,0), S_2(5,3,r,l,i+L,0), \cdots, \\ S_K(5,3,r,l,KL,0), S_\square$$

where $K = \lceil 2^l/L \rceil$. The action code 5 is utilized for the tash OR. For example, the selective reading commens broadcasted for the 0^{th} entry shown in Fig. 7 are sketched:

$$S_1(0,3,r,l,\underline{0},0), S_2(5,3,r,l,\underline{5},0), S_\square$$

E. Further Discussion

At first glance, obtaining a tash table takes a relatively longer time than obtaining a bitmap because a bitmap requires only one round of inventory, whereas a tash table requires multiple rounds. The additional time consumption is a trade-off for practicality because the reply of a COTS tag at the slot level is beyond control. Nevertheless, this additional cost brings an additional benefit, i.e., a tash table has the exact number of tags mapped onto its each entry, which cannot be suggested by a bitmap. Moreover, a one-stop operator service can save more time.

Qian *et al.* [25] and Shahzad and Liu [28] proposed a similar concept of utilizing a pre-stored random bit-string to construct a lightweight pseudo-random function. These studies have inspired our work. However, their main objective of these previous researchers is to accelerate the calculation of a random number, which still requires the engagement with the chip of a tag, and thus, has never been implemented in practice. In the present work, we do not require additional efforts on changing the logics of a tag chip and we associate this concept with the function of selective reading, moving the main task from a tag to a reader. Our design not only preserves the good features of the hash function but also gives a practical solution. This process has never been performed before.

Channel error is one of the most notorious problems of HEPs because pure one-bit signal transmission is vulnerable to ambient interference. Thus, an additional error control mechanism is expected to be applied to HEPs. In the Gen2 protocol, the CRC8 code is automatically appended to the data transmitted between a reader and a tag for error detection, even when one bit of EPC is transmitted. The corrupt data will be retransmitted. Therefore, we should not be concerned with channel error.

V. Tash Implementation

Our implementation involves two kinds of protocols: UHF Gen2 air interface protocol (Gen2) and Low Level Reader Protocol (LLRP). As shown in Fig. 8, Gen2 protocol defines the physical and logical interaction between readers and passive tags, while LLRP allows a client computer to control a reader. Each client computer connects one ore more RFID readers via Ethernet cables. LLRP is the *driver program (or driver protocol)* for Gen2 readers. We leverage LLRP to manipulate a reader to broadcast Gen2 commands that we need. Notice that we do not need particularly implement Gen2 protocol,

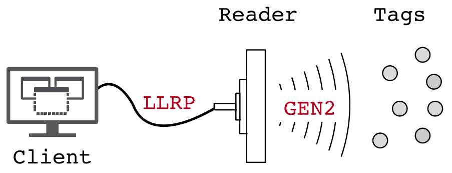
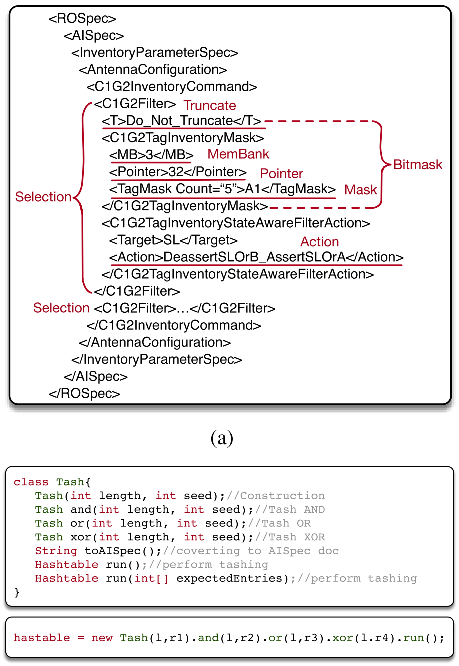


Fig. 8. Gen2 vs. LLRP. Gen2 is the air protocol between a reader and tags while LLRP is the driver protocol between a client computer and a reader. Our framework leverages LLRP to manipulate a reader to broadcast Gen2 commands in need.


Truncate
MemBank
Pointer
Mask
Bitmask
Selection
Action
Selection

```
<ROSpec>
  <AISpec>
    <InventoryParameterSpec>
      <AntennaConfiguration>
        <C1G2InventoryCommand>
          <C1G2Filter>
            <T>Do_Not_Truncate</T>
            <C1G2TagInventoryMask>
              <MB>3</MB>
              <Pointer>32</Pointer>
              <TagMask Count="5">A1</TagMask>
            </C1G2TagInventoryMask>
            <C1G2TagInventoryStateAwareFilterAction>
              <Target>SL</Target>
              <Action>DeassertSLOrB_AssertSLOrA</Action>
            </C1G2TagInventoryStateAwareFilterAction>
          </C1G2Filter>
          <C1G2Filter>...</C1G2Filter>
        </C1G2InventoryCommand>
      </AntennaConfiguration>
    </InventoryParameterSpec>
  </AISpec>
</ROSpec>
```

(a)

```
class Tash{
   Tash(int length, int seed);//Construction
   Tash and(int length, int seed);//Tash AND
   Tash or(int length, int seed);//Tash OR
   Tash xor(int length, int seed);//Tash XOR
   String toAISpec();//coverting to AISpec doc
   Hashtable run();//perform tashing
   Hashtable run(int[] expectedEntries);//perform tashing
}
```

```
hastable = new Tash(l,r1).and(l,r2).or(l,r3).xor(l.r4).run();
```

(b)

Fig. 9. Tash implementation. (a) The specification of the XML file defines various parameters that are required for selection commands. (b) The primary interfaces provides by tash framework, which is developed by using Java language and LLRP Toolkit.

which has been implemented in the COTS RFID devices that we are using. Specifically, LLRP specifies two types of operations: reader operation (RO) and access operation (AO). Both operations are represented in XML document form and transported to a reader through TCP/IP. Note that the function of LLRP clients is to transmit an XML-Formatted configuration document from the reader to the reader. It is no different in using Octane LLRP client [66] (for ImpinJ reader) or the open-source LLRP toolkit [67] to transmit the document.

Reader Operation: RO defines the inventory parameters specified in the Gen2 protocol, such as bitmask, antenna power, and frequency. Fig. 9(a) shows a simplified instance of an `ROSpec`. An `ROSpec` is composed of at least one `AISpec`. Each `AISpec` is used for an antenna setting. An `AISpec` consists of more than one `C1G2Filters`. The filter functions as a bitmask. We can set multiple selection commands by adding multiple `C1G2Filters`.

Access Operation: AO defines the access parameters for writing or reading data to and from a tag. We leverage the `C1G2Write` inside an `AOSpec` to write the hash value of the

TABLE III

SUMMARY OF GEN2-COMPATIBILITY ON TAG

	ImpinJ Monza									**Alien ALN**								
`Tag models`	5	D	E	QT	X-2K	X-8K	R6	R6-P	R6-C	9840	9830	9662	9610	9726	9820	9715	9716	9629
`MemBank1` (bits)	128	128	496	128	128	128	96	128/96	96	128	128	480	96-480	128	128	128	128	96
`MemBank3` (bits)	32	32	128	512	2176	8192	×	32/64	32	128	128	512	512	128	128	128	128	512
`Write` cmd	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`Select` cmd	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`Truncate` cmd	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

`EPC` into a user-defined memory bank. As the `EPC`s are highly related to the products the tags attached, the writing of hash values should be accomplished by the product manufactures or administrators. There is almost no overhead to write data into `MemBank-3` since it is allowed to write a batch of tags simultaneously using `Write` commands specified in one `AOSpec`, without physically changing tags' positions.

Tash Framework: Our framework is developed by using Java language and the LLRP Toolkit [67], which is an open-source library for handling `ROSpec` and `AOSpec`. Fig. 9(b) shows the primary interfaces provided by the tash framework. The class `Tash` makes the first selection through its construction method and allows the calls of three operators to be chained together in a single statement. The method `toAISpec` converts a `Tash` object or a chain of `Tash` objects into an `AISpec`. The entry-inventories are physically executed in the connected reader when the method `run` is invoked. This method allows users to make selective entry-inventories by passing an index array. For example, the operation $\mathcal{F}_l(T, r_1) \oplus \mathcal{F}_l(T, r_2) || \mathcal{F}_l(T, r_3) \otimes \mathcal{F}_l(T, r_4)$ can be coded in a manner similar to that shown at the bottom of Fig. 9(b).

VI. MICROBENCHMARK

We start with a few experiments that provide insight to our hash primitives.

A. Experimental Setup

We evaluate the framework using COTS UHF readers and tags. We use a total of 3 models of ImpinJ readers (R220, R420 and R680), each of which is connected to a 900MHz and 8dB gain directional antenna. In order to better understand the feasibility and effectiveness of Tash in practice, we test a total of $3,000$ COTS tags with different models. We divide these tags into 10 groups of 300 tags each. The tags of each group are densely attached to a plastic board which is placed in front of a reader antenna. As shown in Fig. 10, three hundreds is the maximum number of tags that can be covered by one directional antenna in our laboratory. We store the $3,000$ `EPC` numbers in our database as the ground truth. The 128-bit MD5 is employed as the common hash function to generate the hash values of `EPC`s. The experiments with the same settings are repeated across the 10 groups, and the average result is reported.

B. Compatibility Investigation

First, we investigate the compatibility of Gen2 across 10 different types of readers and 18 different types of tags in

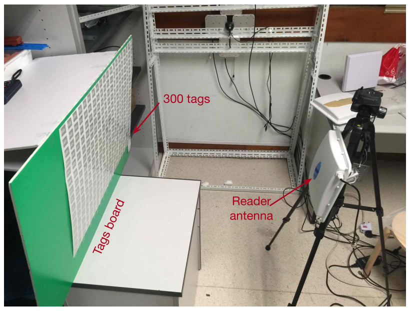


Fig. 10. Testbed in our laboratory. Total 300 tags are attached on a board and covered by a directional reader antenna.

terms of the functions or commands that Tash requires. The readers and tags may come from different manufacturers but work together in practice. These investigated products are all publicly claimed to be completely Gen2-compatible.

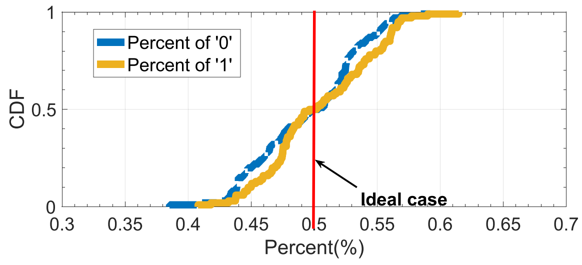
Reader Compatibility: We investigate the R220, R420 and R680 models from ImpinJ [68],[3] the Mercury6, Sargas and M6e models from ThingMagic [69], as well as the ALR-F800, 9900+, 9680 and 9650 models from Alien [70]. We perform the investigation through real tests for the first three models of readers (i.e., the ImpinJ series), and investigate the other readers through their data sheets or manuals (because we are limited by the lack of hardware). The Gen2-compatibility of readers is briefly summarized in Table. IV. Consequently, we have the subsequent findings. (1) All the readers do support `Write/Read` command, which Tash uses for writing or reading hash values of `EPC` numbers. (2) All the readers do support the `Select` command, which Tash uses for the selective reading. (3) The development manual (Page 15, Table 3-1) released by ImpinJ Corp. [66] states that the `Truncate` flag of all its series must be `always` be set to 0, namely, the flag is unspecified and allows the reader to decide what truncate action to take. However, our practical tests suggest that none model of the ImpinJ series supports the `Truncate` command, which Tash uses to hear the one-bit presence signal. The serviceability of other readers is not clearly indicated in the manuals of those readers. (4) The Gen2 protocol does not specify how many `C1G2Filters` and `AISpecs` that a reader should support. Our practical tests suggest that the ImpinJ series supports 4 `C1G2Filters` and 16 `AISpecs`, which means that we can only use a maximum of four tash operators each time.

Tag Compatibility: We investigate 9 chip models from ImpinJ Monza series and 9 additional models from Alien ALN series. The majority of tags on the market contain

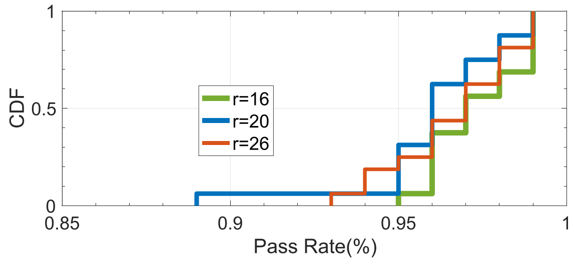
[3]Since the R1000 is outdated, its compatibility remains unknown.

TABLE IV
SUMMARY OF GEN2-COMPATIBILITY ON READER

Commands or functions	ImpinJ	ThingMagic	Alien
`Write/Read`	✓	✓	✓
`Select`	✓	✓	✓
`Truncate`	×	–	–
Max No. of `C1G2Filters`	4	–	–
Max No. of `AISpecs`	16	–	–



(a)



(b)

Fig. 11. Evaluation of tash function. (a) shows the CDF of percents of '0' and '1' appearing in the tash values. (b) shows the CDF of pass rates of randomness tests.

these 18 models of chips and customized antennas. Table III summarizes the result of our investigation, from which we have the subsequent findings. (1) Tags reserve 96 $\sim$ 480 bits of memory for storing `EPC` numbers, among which the size of 96 bits has become the de facto standard. (2) Tash requires `MemBank-3` to store the hash values. The results of the investigation show that almost all tags allow to write to and read from the third memory bank, with an exception of ImpinJ Monza R6, which does not have the user-defined memory. The size of the third memory bank fluctuates around 32 $\sim$ 512 bits. The de facto standard has become 128 bits.

Summary: Despite positive and public claims, our investigation shows that current COTS RFID devices, regardless of readers or tags and models, have some defects in their compatibility with Gen2, especially with regard to `Truncate`. The reason, we may infer, is that these commands are seldom used in practice and therefore never receive attention from manufacturers. The partial compatibility of such devices cannot fully achieve the performance Tash brings. Even so, we are obliged to make the claim, again, that our design strictly follows the Gen2 protocol. We hope this work can encourage manufacturers to upgrade their products (e.g., reader firmware) to achieve full compatibility.

C. Evaluation of Tash Function

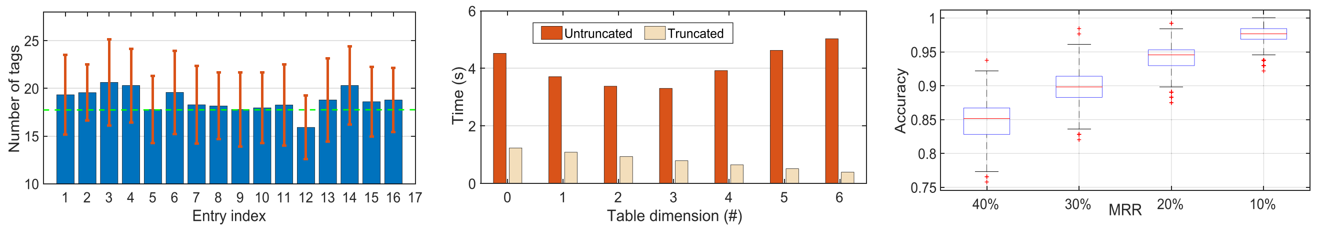
Second, we evaluate the tash function with respect to the randomness and the accessibility.

Randomness: Randomness is the most important metric for a hash function. It requires that the outputs of a hash function must be uniformly distributed. To validate the randomness of the tash function, we collect $99,886$ real `EPC` numbers from our partner (i.e., an international logistics company), which introduced RFID technology for sorting tasks five years ago. Each `EPC` number has a length of 96 bits and encodes the basic information about the package, such as sources, destinations, serial numbers, and so on. We employ the 128-bit MD5 to create the hash values of these `EPC`s. As the minimum size of the `MemBank-3` is 32 bits (see Table IV), we choose to use only the first 32 bits for our tests. We traverse r and l from $0 \sim 31$ and $1 \sim 32 - r$ respectively. For each pair of r and l, we obtain $99,886$ tash values over all the `EPC`s. Across these tash values, we further conduct the following two analysis: (1) We merge 100 tash values, which are randomly selected from the above results, into a long bit string. We then calculate the percents of '0' and '1' emerged in that bit string. This operation is repeated for 100 times. Finally, totally 100 pairs of percents are obtained. Their CDFs are plotted in Fig. 11(a). Ideally, each bit has a equal probability of 0.5 to be zero or one if a hash function makes a good randomicity. From the figure, we can figure out that the percents distributed between 0.4 and 0.6. In particular, percents of '0' and '1' have means of 0.49 and 0.50 with standard deviations of 0.043 and 0.044 respectively. (2) We shuffle these values into 100 groups, and employ the χ^2-test with a significance level of 0.05 to test each group's goodness-of-fits of the uniform distribution (i.e., passed or failed). Then, we finally calculate the pass rate for a pair of setting. In this manner, we totally obtain 496 pass rates. More than 60% of the pass rates are over than 0.95. In particular, three sets of the results with $r = 16$, 20 and 26 and a variable l, are selected to show in Fig. 11(b). We find that 90% of the pass rates exceed 0.95 for the three cases, and their median pass rates are around 0.97. Thus, the two above statistical results suggest that our tash function has a very good quality of randomness.

Accessibility: Accessibility refers to the ability to get access to a tash value from a tag. As aforementioned, we have two ways to acquire the tash values. The first way is to use the `Read` command. The second way is to indirectly access a tash value through a selective reading. We choose the second method since it is the basis of our design. Specifically, we perform a selective reading to determine whether the tags are collected as expected, when given random inputs and a possible tash value. We intensively and continuously perform such readings across the 10×300 tags using three 4-port ImpinJ readers for three rounds of 24 hours in a relatively isolated environment (e.g., an empty room without disturbance). Surprisingly, we find all the reading results faithfully conform to our benchmarks without any exceptions. This shows that the selective reading is well supported by the manufactures and is both stable and reliable.

D. Evaluation of Tash Table Function

Third, we evaluate the performance of the tash table function in terms of its balance and gathering speed.



(a) Balance (b) Gathering time (c) Accuracy

Fig. 12. Evaluation of tash table function. (a) shows the balance of a 4-bit tash table across 300 tags using 100 different random seed; (b) shows the time consumption on gathering 6 hash tables with different dimensions; (c) shows the accuracy of hash bitmap as a function of the channel state.

Balance: A good hash table function will equally assign each key to a bucket. We expect the output tash table to be as balanced as possible. To show this feature, we generate 100 different 4-bit tash tables (i.e., each includes 16 entries) across 300 tags using 100 different random seeds. If the tash table is well balanced, the expected number of each entry should be very close to $300/16 = 18.7$. Fig. 12(a) shows the mean number of tags in each entry as well as their standard deviations. The average number across 16 entries equals 18.75, which is very close to the expected theoretical value. The average standard deviation equals 0.44. Thus, the good randomness quality of tash functions results in output tash tables being well balanced.

Gathering Overhead: We then consider the time consumption of gathering a tash table. Fixing the random seed, we vary the table dimension l from 0 to 6. We then measure the time taken on gathering a tash table with the deployed 300 tags. Fig. 12(b) shows the resulting time as a function of the table dimension. From the immediately above-mentioned figure, we can observe the subsequent findings.

- When $l = 0$ without truncating reply, the result is equivalent to collecting 300 complete `EPC`s of all the tags one by one. Such time consumption (i.e., $4,524ms$) is viewed as the baseline.
- By contrast, when $l > 0$ without truncating a reply, the collection amounts to dividing all the tags into 2^l groups "equally" and then collecting each group independently. In this manner, when $l \leq 4$, such "divide and conquer" approach is better than "one time deal", i.e., a drop in overhead of about 10%. The Gen2 reader uses a Q-adaptive algorithm for the anti-collision. This algorithm is able to adaptively learn the best frame length from the collision history. Due to the division, a smaller number of tags can make reader's learning relatively quicker and improve the overall performance. However, when $l > 4$, the performance of "divide and conquer" approach starts to deteriorate. This is because the ImpinJ reader supports 16 `AISpecs` at most (see Table. IV). We have to re-send another `ROSpec` for the remaining selective readings when the number of entry-inventory is above 16 (i.e., $l > 4$), which introduces additional time consumption.
- We then consider the case where the reply is truncated to a one-bit presence signal as assumed by HEPs. Due to the defects of ImpinJ readers in the implementation of the `Truncate` command, we use a USRP N210 to implement a software defined reader, which can truncate any length of the `EPC` reply. The source code of the implementation can be found in [71]. The results show that truncated reply would introduce about $72.7\%, 70.75\%, 72.26\%, 76.06\%, 83.46\%, 88.89\%$ and 92.18% drop of the overhead in the six dimensions respectively. Unlike the untruncated results with ImpinJ reader, our USRP reader does not require to receive the additional `ROSpecs`. Thus, we can gain more time reduction as the dimension increases.

Gathering accuracy: We evaluate the accuracy of gathering a hash bitmap as a function of the Missing Reading Rate (MRR). In practice, the reader would miss a tag's reply due to the ambient interference or temporal obstacle. The MRR is the rate of the unsuccessful reading attempts in which a tag's `EPC` is not received correctly. For example, we attempt to read a tag's `EPC` 100 times but it is only correctly identified in 70 attempts. Then the MRR equals 30%. Here, the gathering accuracy is expressed as the percentage of the entries which are correctly determined. To evaluate the accuracy, we distribute 300 tags around our room instead of in a regular grid, such that their channel states are different. Meanwhile, we runs a robot carrying a metal plate to move forward and backward in the rooms. In this way, we can emulate a wanted MRR by adjusting the moving speed of the robot. We totally test four cases where the average MRR is controlled at $40\%, 30\%, 20\%$ and 10% respectively. In each case, we repeat to gather a 7-bit tash table with a length of 128 entries for 50 times. Fig. 12(c) shows the accuracy distribution in the four cases. It can be seen from the figure that the accuracy is above 90% even when the MRR is up 30%. In other words, 30% tags would be missed to read if reading them all, but 90% entries in the tash table acquired across these tags are correct. Clearly, the accuracy of a tash table is higher than reading tags all. This is because tash only requires each tag to reply a truncated `EPC` rather than a whole one. It is more reliable to transmit a fewer bits.

E. Evaluation of Tash Operators

Finally, we investigate the performance of tash operators. Superior to existing HEPs, these operators allow us to perform set operations on-tag and conduct a one-stop inventory. In particular, we show the performance of OR as a representative across 300 tags. The tests for other operators are similar and omitted due to the space limitation. In the experiments,

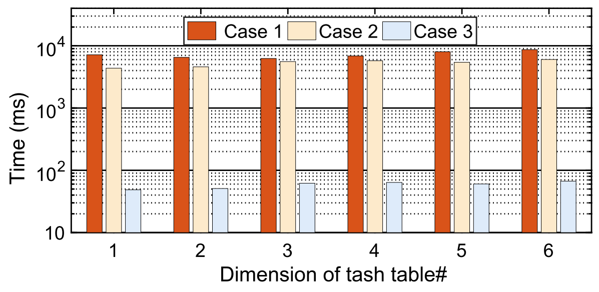


Fig. 13. Performance of tash OR. In Case 1, two tash tables are conducted OR operation on application layer without truncating replies. In Case 2 and Case 3 two tables are conducted Tash OR on tags without/with truncating replies.

we fix the two random seeds but change the dimension of tash table. Fig. 13 shows the results of three cases. In Case 1, we independently produce 2 tash tables without truncating a reply and conduct the OR in the application layer. In Case 2 and Case 3, we conduct on-tag OR function as Tash provides without and with truncating a reply respectively. Consequently, when the dimension equals 2, Case 1 takes $6,511ms$ on collecting two tables. On the contrary, the amount of time taken is reduced to $4,578ms$ (i.e., 29.7% drop) if we perform an on-tag OR function even without truncation (Case 2). Ideally, the amount of time taken could be further reduced to $50.97ms$ by using a truncating reply (Case 3), which offers a staggering drop in time usage by 99.22%. Our experiments relate only to the amount of time spent on ORing two tables. It may be predicted that much more outperformance will be gained if multiple tables are involved. The tash operators that we design in this work have never been proposed before.

VII. Tash Applications

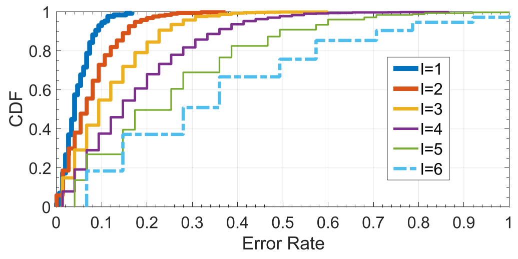
We then use our prototype to demonstrate the benefits and potentials of Tash in two typical applications.

A. Application I: Cardinality Estimation

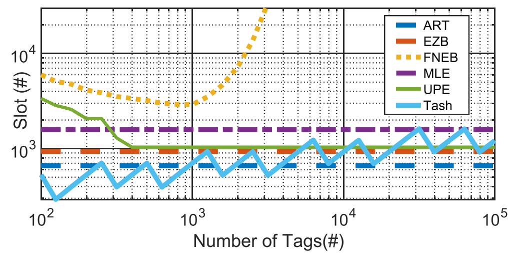
We evaluate our estimation scheme through the testbed as well as large-scale simulations.

Testbed Based: Our scheme only uses the first entry of the tash table for the estimation, thereby we only need one entry-inventory. Fig. 14(a) shows the CDF of estimation results across 300 tags. We define the error rate as $|n-\widehat{n}|/n$ where $\widehat{n}$ is the estimated number. As a result, 90% of the estimations have an error rate less than 0.1 and a median of 0.04 when setting the dimension $l = 1$. In this case, almost half tags follow into the first entry so the rate could be pretty high, at the price of longer inventory time. As l increases, the error rate also increases because less samples are acquired for the estimation. These experiments show the feasibility of using tash table for cardinality estimation.

Simulation Based: We then perform the evaluation through large-scale simulations for two reasons: (1) ensuring its scalability when meeting a huge number of tags. (2) making comparisons with prior work, which are all simulation-based. For the fairness, we numerically simulate in Matlab



(a) Testbed



(b) Large-scale simulation

Fig. 14. Cardinality estimation. (a) shows the CDF of error rates for estimating 300 tags with our testbed. (2) shows the estimation comparisons with other theoretical algorithms with simulation. (a) Testbed. (b) Large-scale simulation.

using tash scheme as well as other five prior RFID estimation schemes: UPE [21], EZB [22], FNEB [72], MLE [73], ART [28]. We implement these schemes by referring to the RFID estimation tool developed by Shahzad [74]. Fig. 14(b) shows the time cost with a varying n given $\alpha = 0.9$ and $\beta = 0.08$. We observe that our scheme is $5\times$ faster than the others on average when $n < 1000$. Thus our scheme is suitable for the estimation with a small number of tags. When $n > 1000$, the performance of our scheme starts to vibrate between ART and MLE, due to two reasons. First, our scheme is not collision-free so that more efforts are required to deal with the collisions incurred by more tags. Second, the size of a tash table can only increase in the power of two, making the size always vibrate around the optimal one. Even so, the advantage of our scheme is still clear: it is the first RFID estimation scheme that can work in real life. Notice that ART claimed to work with RFID systems because they are theoretically compatible with ALOHA protocols. Actually, the current COTS RFID systems do not allow user to control the low-level access, like fined-grained adjustment of frame length and obtaining slot-level feedback, which are necessary to implement ART. Thus, there is no way for ART to implement their algorithms over COTS RFID systems without any hardware modification and fabrication.

B. Application II: Missing Detection

Consider a major warehouse that stores thousands of apparel, shoes, pallets, and cases. How can a staff *immediately* determine if anything is missing? This section demonstrates the Tash usage for a typical application: missing detection.

1) Problem Formulation: The purpose of missing tag detection is to quickly find out the missing tags without collecting all the tags in the scene. Such detection is very useful,

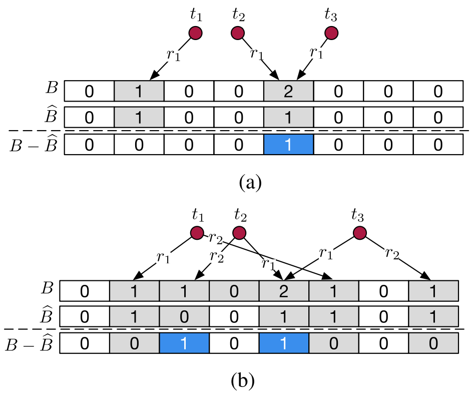


Fig. 15. An example of missing detection. B is the intact tash table generated using the known EPCs while $\widehat{B}$ is an instance over the tags in the current scene. (a) Tashing once. (b) Tashing twice.

especially when thousands of tags are present. We formally define the problem of detecting missing tags in Problem 1. We assume that the EPCs of all the tags in a closed system are stored in a database and known in advance. This assumption is reasonable and necessary, because it is impossible for us to tell that a tag is missing without any prior knowledge of its existence.

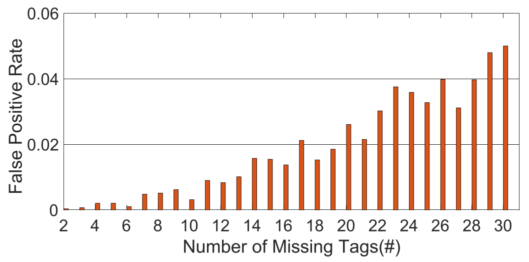
Problem 1: How to quickly identify m missing out of n tags with a false positive rate of γ at most?

2) Proposed Detector: The underlying idea is to compare two tash tables B and $\widehat{B}$. B is an *intact tash table* created by tashing all the known EPCs which are stored in the database, while $\widehat{B}$ is an *instance tash table* obtained from the tags in the scene. We can detect the missing tags through comparing the difference between B and $\widehat{B}$. If the residual table $B-\widehat{B}$ (i.e., entry-wise subtraction) equals 0, no missing tag event happens. Otherwise, the tags mapped into the non-zero entries of the residual table are missing. Fig. 15(a) illustrates an example in which three tags, t_1, t_2 and t_3, are mapped into the intact tash table B. $\widehat{B}$ is an instance table where tag t_2 is missing, and thus $\widehat{B}[4]=1$. Consequently, $(B-\widehat{B})[4]=1$, we can definitely infer that one tag is missing. However, it is impossible for us to tell which tag is missing because t_2 and t_3 are simultaneously mapped into the fourth entry.

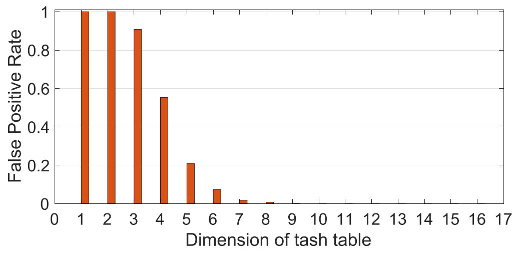
Inspired by the Bloom filter [75], we perform k tashings to identify the missing tags as follows:

$$B=\mathcal{F}_l(T,r_1)||\ldots||\mathcal{F}_l(T,r_k) \tag{3}$$

The final B after tash ORs is considered to use k independent hash functions (i.e., induced by k random seeds) to map each tag into B for k times, as shown in Fig. 15(b). The residual table of $B-\widehat{B}$ is therefore viewed as a Bloom filter which represents the missing tags. Thereafter, to answer a query of whether a tag t is missing, we check whether all entries set by $f_l(t,r_1),\cdots$ and $f_l(t,r_k)$ in the residual table have a value of non-zero. If the answer is yes, then tag t is the missing one. Otherwise, it is not the missing tag. Fig. 15(b) illustrates an example in which each tag is tashed twice. The missing tag t_2 can be identified because both the 2^{rd} and the 4^{th} entry in the residual table have value of non-zero. Despite multiple



(a) $k=2$ and $l=8$



(b) $k=2$ and $m=10$

Fig. 16. Missing detection with 300 tags. (a) The resulted FPRs as function of the missing number (b) as function of the dimension of tash tables.

tashings, the query may yield a *false positive*, where it suggests a tag is missing even though it is not.

Analysis: To lower the rate of false positive rate, it is necessary to answer two questions.

(1) *How many tash functions do we need?* Given the table dimension l, we expect to optimize the number of tash functions. There are two competing forces: using more tash functions gives us more chance to find a zero bit for a missing tag, but using fewer tash functions increases the fraction of zero bits in the table. After m missing tags are tashed into the table, the probability that a specific bit is still 0 is $(1-\frac{1}{L})^{km}\approx e^{-km/L}$ where $L=2^l$. Correspondingly, the probability of a false positive p is given by

$$p=(1-e^{-km/L})^k \tag{4}$$

Namely, a missing tag falls into k non-zero entries. Lemma. 1 suggests that the optimal number of tash functions is achieved when $k=\ln 2\cdot(L/m)$.

Lemma 1: The false positive rate is minimized when $p=(1/2)^k$ or equivalently $k=\ln 2\cdot(L/m)$.

Proof: Please refer to [75] for the proof. □

(2) *How large tash table is necessary to represent all m missing tags?*

Recall that the false positive rate achieves minimum when $p=(1/2)^k$. Let $p\le\gamma$. After some algebraic manipulation, we find

$$L\ge\frac{m\log_2(1/\gamma)}{\ln 2}=m\log_2 e\cdot\log_2(1/\gamma)=1.44m\log_2(1/\gamma) \tag{5}$$

Finally, putting the above conclusions together, we have the subsequent theorem.

Theorem 1: Setting the table dimension to $\lceil\log_2(1.44m\log_2(1/\gamma))\rceil$ and using $\lceil\ln 2\cdot(2^l/m)\rceil$ random seeds allow the *false positive rate of identifying m missing tags lower than a given tolerance γ.*

3) Evaluation: We evaluate the effectiveness of missing detection in real case. We randomly remove m tags from our testbed as shown in §VI. Since we only have 300 tags in total, we fix the number of random seeds to 2, i.e., $k = 2$. The performance is evaluated in term of the false positive rate (FPR), which is the ratio of number of mistakenly detected as missing tags to the total number of really missing tags. Our scheme is able to successfully find out all the missing tags because the residual table always contains the entries that missing tags are tashed into. Fig. 16(a) shows the results of the first case in which we use an 8-bit hash table (i.e., $l = 8$) to detect the missing tags. Consequently, the FPR is maintained around 0.01 when $m < 14$ (i.e., 5% of the tags are missing). Fig. 16(b) shows the second case in which we remove 10 tags and detect the missing tags by changing the dimension of tash table. As Theorem. 1 suggests, we should set $l = 5, 6, 7$ to guarantee the FPR $\gamma < 0.2, 0.1, 0.01$. From the figure, we can find that the results of our experiments completely conform to this theorem. The real FPRs equal $0.21, 0.07$ and 0.008 in the three cases. Tash enabled missing detection works well in practice.

VIII. Related Work

Two categories of work are related to ours: the design of the hash function and the applications of HEP. As the common HEP applications have been reviewed to motivate our design in §VIII. Here, we pay more attention on the popular designs of hash functions. Feldhofer and Rechberger [7] firstly point that current common hash functions (e.g., `MD5`, `SHA-1`, etc.), are not hardware friendly and unsuitable at all for RFID tags, which have very constrained computing ability. Such difficulty has spurred considerable research [7]–[15]. The primary designs and their features are sketched in Table. I. For example, Bogdanov *et al.* [10] propose a hardware-optimized block cipher, PRESENT, designed with area and power constraints. The follow-up work [11] presents three different architectures of PRESENT and highlights their availability for both active and passive smart devices. Their implementations reduce the number of GEs to $1,000$ around. Another follow-up work [20] extends the design of PRESENT and gives 8 variants to fulfill different requirements, e.g., DM-PRESENT-80, DM-PRESENT-128, H-PRESENT-128, etc. The work [8] suggests to choose DES as hash function for RFID tags due to relatively low complexity, and presents a variant of DES, called asi.e., DESXL. Lim and Korkishko [12] present a 64-bit hash function with three key size options (64 bits, 96 bits and 128 bits), which requires about $3,500$ and $4,100$ GEs. In summary, despite these optimized designs, majority are still presented in theory and none of them are available for the COTS RFID tags. On contrary, our work explores hash function from another different aspect, that is, leveraging selective reading to mimic equivalent hash primitives.

IX. Conclusion

This work discusses a fundamental issue that how to supplement hash functionality to existing COTS RFID systems, which is dispensable for prior HEPs. A key innovation of this work is our design of hash primitives, which is implemented using selective reading. Tash not only makes a big step forward in boosting prior HEPs, but also opens up a wide range of exciting opportunities.

References

[1] L. Yang *et al.*, "Tagoram: Realtimetracking of mobile RFID tags to high precision using COTS devices," in *Proc. ACM MobiCom*, 2014, pp. 237–248.

[2] L. Yang, Y. Li, Q. Lin, X.-Y. Li, and Y. Liu, "Making sense of mechanical vibration period with sub-millisecond accuracy using backscatter signals," in *Proc. ACM MobiCom*, 2016, pp. 16–28.

[3] *Global RFID Healthcare and Pharmaceutical Marke*, Frost and Sullivan, San Antonio, TX, USA, 2011.

[4] (2004). *EPCglobal Gen2 Specification*. [Online]. Available: www.gs1.org/epcglobal

[5] J. Wang, H. Hassanieh, D. Katabi, and P. Indyk, "Efficient and reliable low-power backscatter networks," in *Proc. ACM SIGCOM*, 2012, pp. 61–72.

[6] D. M. Dobkin, *The RF in RFID: UHF RFID in Practice*. Newnes, NSW, Australia: Elsevier, 2012.

[7] M. Feldhofer and C. Rechberger, "A case against currently used hash functions in RFID protocols," in *Proc. Workshops Move Meaningful Internet Syst. (OTM)*. Berlin, Germany: Springer, 2006, pp. 372–381.

[8] A. Poschmann, G. Leander, K. Schramm, and C. Paar, "New lightweight des variants suited for RFID applications," in *Proc. FSE*, vol. 4593, 2007, pp. 196–210.

[9] H. Yoshida *et al.*, "MAME: A compression function with reduced hardware requirements," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2007, pp. 148–165.

[10] A. Bogdanov *et al.*, "PRESENT: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2007, pp. 450–466.

[11] C. Rolfes, A. Poschmann, G. Leander, and C. Paar, "Ultra-lightweight implementations for smart devices—Security for 1000 gate equivalents," in *Smart Card Research and Advanced Applications*, vol. 5189. Berlin, Germany: Springer, 2008, pp. 89–103.

[12] C. H. Lim and T. Korkishko, "mCrypton—A lightweight block cipher for security of low-cost RFID tags and sensors," in *Proc. WISA*, vol. 3786. Berlin, Germany: Springer, 2005, pp. 243–258.

[13] Y. Yu, Y. Yang, Y. Fan, and H. Min. *Security Scheme for RFID Tag: Auto-ID Labs White Paper WP-HARDWARE-022*. Accessed: Jan. 1, 2019. [Online]. Available: http://www.autoidlabs.org/

[14] D. Hong *et al.*, "HIGHT: A new block cipher suitable for low-resource device," in *Proc. CHES*, vol. 4249. Berlin, Germany: Springer, 2006, pp. 46–59.

[15] T. Good and M. Benaissa, "Hardware results for selected stream cipher candidates," *State Art Stream Ciphers*, vol. 7, pp. 191–204, Feb. 2007.

[16] M. Philipose, J. R. Smith, B. Jiang, A. Mamishev, S. Roy, and K. Sundara-Rajan, "Battery-free wireless identification and sensing," *IEEE Pervasive Comput.*, vol. 4, no. 1, pp. 37–45, Jan. 2005.

[17] H. Zhang, J. Gummeson, B. Ransford, and K. Fu, "Moo: A batteryless computational RFID and sensing platform," Dept. Comput. Sci., Univ. Massachusetts, Boston, MA, USA, Tech. Rep. UM-CS-2011-020, 2011.

[18] C. Pendl, M. Pelnar, and M. Hutter, "Elliptic curve cryptography on the WISP UHF RFID tag," in *RFID. Security and Privacy* vol. 7055, no. 1. Jun. 2011, pp. 32–47.

[19] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Proc. CHES*, vol. 4. Berlin, Germany: Springer, 2004, pp. 357–370.

[20] A. Bogdanov *et al.*, "Hash functions and RFID tags: Mind the gap," in *Proc. IACR CHES*, 2008, pp. 283–299.

[21] M. Kodialam and T. Nandagopal, "Fast and reliable estimation schemes in RFID systems," in *Proc. ACM MobiCom*, 2006, pp. 322–333.

[22] M. Kodialam, T. Nandagopal, and W. C. Lau, "Anonymous tracking using RFID tags," in *Proc. IEEE INFOCOM*, May 2007, pp. 1217–1225.

[23] B. Sheng, C. C. Tan, Q. Li, and W. Mao, "Finding popular categories for RFID tags," in *Proc. ACM MobiHoc*, 2008, pp. 159–168.

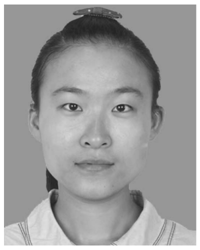
[24] W.-K. Sze, W.-C. Lau, and O.-C. Yue, "Fast RFID counting under unreliable radio channels," in *Proc. IEEE ICC*, Jun. 2009, pp. 1–5.

[25] C. Qian, Y. Liu, H. Ngan, and L. M. Ni, "ASAP: Scalable identification and counting for contactless RFID systems," in *Proc. IEEE ICDCS*, 2010, pp. 52–61.

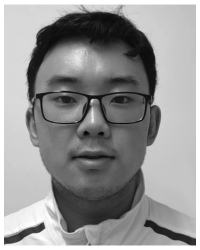
[26] C. Qian, H. Ngan, Y. Liu, and L. M. Ni, "Cardinality estimation for large-scale RFID systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 9, pp. 1441–1454, Sep. 2011.
[27] V. Shah-Mansouri and V. W. S. Wong, "Cardinality estimation in RFID systems with multiple readers," *IEEE Trans. Wireless Commun.*, vol. 10, no. 5, pp. 1458–1469, May 2011.
[28] M. Shahzad and A. X. Liu, "Every bit counts: Fast and scalable RFID estimation," in *Proc. ACM MobiCom*, 2012, pp. 365–376
[29] Y. Zheng and M. Li, "ZOE: Fast cardinality estimation for large-scale RFID systems," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 908–916.
[30] B. Chen, Z. Zhou, and H. Yu, "Understanding RFID counting protocols," in *Proc. ACM MobiCom*, 2013, pp. 291–302.
[31] Q. Xiao, B. Xiao, and S. Chen, "Differential estimation in dynamic RFID systems," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 295–299.
[32] W. Gong, K. Liu, X. Miao, and H. Liu, "Arbitrarily accurate approximation scheme for large-scale RFID cardinality estimation," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 477–485.
[33] X. Liu, K. Li, H. Qi, B. Xiao, and X. Xie, "Fast counting the key tags in anonymous RFID systems," in *Proc. IEEE ICNP*, Oct. 2014, pp. 59–70.
[34] Y. Zheng and M. Li, "Towards more efficient cardinality estimation for large-scale RFID systems," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 1886–1896, Dec. 2014.
[35] X. Liu *et al.*, "RFID cardinality estimation with blocker tags," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1679–1687.
[36] Y. Hou, J. Ou, Y. Zheng, and M. Li, "PLACE: Physical layer cardinality estimation for large-scale RFID systems," in *Proc. IEEE INFOCOM*, Oct. 2015, pp. 1957–1965.
[37] C. C. Tan, B. Sheng, and Q. Li, "How to monitor for missing RFID tags," in *Proc. IEEE ICDCS*, Jun. 2008, pp. 295–302.
[38] T. Li, S. Chen, and Y. Ling, "Identifying the missing tags in a large RFID system," in *Proc. ACM MobiHoc*, 2010, pp. 1–10.
[39] W. Luo, S. Chen, T. Li, and S. Chen, "Efficient missing tag detection in RFID systems," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 356–360.
[40] R. Zhang, Y. Liu, Y. Zhang, and J. Sun, "Fast identification of the missing tags in a large RFID system," in *Proc. IEEE SECON*, Jun. 2011, pp. 278–286.
[41] W. Luo, S. Chen, T. Li, and Y. Qiao, "Probabilistic missing-tag detection and energy-time tradeoff in large-scale RFID systems," in *Proc. ACM MobiHoc*, 2012, pp. 95–104.
[42] Y. Zheng and M. Li, "P-MTI: Physical-layer missing tag identification via compressive sensing," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1356–1366, Aug. 2015.
[43] C. C. Tan, B. Sheng, and Q. Li, "Efficient techniques for monitoring missing RFID tags," *IEEE Trans. Wireless Commun.*, vol. 9, no. 6, pp. 1882–1889, Jun. 2010.
[44] T. Li, S. Chen, and Y. Ling, "Efficient protocols for identifying the missing tags in a large RFID system," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1974–1987, Dec. 2013.
[45] X. Liu, K. Li, G. Min, Y. Shen, A. X. Liu, and W. Qu, "A multiple hashing approach to complete identification of missing RFID tags," *IEEE Trans. Commun.*, vol. 62, no. 3, pp. 1046–1057, Mar. 2014.
[46] W. Luo, S. Chen, Y. Qiao, and T. Li, "Missing-tag detection and energy–time tradeoff in large-scale RFID systems with unreliable channels," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1079–1091, Aug. 2014.
[47] C. Ma, J. Lin, and Y. Wang, "Efficient missing tag detection in a large RFID system," in *Proc. IEEE TrustCom*, Jun. 2012, pp. 185–192.
[48] W. Xie *et al.*, "RFID seeking: Finding a lost tag rather than only detecting its missing," *J. Netw. Comput. Appl.*, vol. 42, pp. 135–142, Jun. 2014.
[49] M. Shahzad and A. X. Liu, "Expecting the unexpected: Fast and reliable detection of missing RFID tags in the wild," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1939–1947.
[50] M. Shahzad and A. X. Liu, "Fast and reliable detection and identification of missing RFID tags in the wild," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3770–3784, Dec. 2016.
[51] J. Yu, L. Chen, R. Zhang, and K. Wang, "On missing tag detection in multiple-group multiple-region RFID systems," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1371–1381, May 2016.
[52] J. Yu, L. Chen, and K. Wang. (2015). "Finding needles in a haystack: Missing tag detection in large RFID systems." [Online]. Available: https://arxiv.org/abs/1512.05228
[53] B. Sheng, Q. Li, and W. Mao, "Efficient continuous scanning in RFID systems," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[54] L. Xie, Q. Li, X. Chen, S. Lu, and D. Chen, "Continuous scanning with mobile reader in RFID systems: An experimental study," in *Proc. ACM MobiHoc*, 2013, pp. 11–20.
[55] H. Liu, W. Gong, X. Miao, K. Liu, and W. He, "Towards adaptive continuous scanning in large-scale RFID systems," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 486–494.
[56] L. Xie, H. Han, Q. Li, J. Wu, and S. Lu, "Efficiently collecting histograms over RFID tags," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 145–153.
[57] W. Luo, Y. Qiao, and S. Chen, "An efficient protocol for RFID multigroup threshold-based classification," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 890–898.
[58] W. Luo, Y. Qiao, S. Chen, and M. Chen, "An efficient protocol for RFID multigroup threshold-based classification based on sampling and logical bitmap," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 397–407, Feb. 2016.
[59] J. Liu, B. Xiao, S. Chen, F. Zhu, and L. Chen, "Fast RFID grouping protocols," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1948–1956.
[60] X. Liu, B. Xiao, S. Zhang, K. Bu, and A. Chan, "STEP: A time-efficient tag searching protocol in large RFID systems," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3265–3277, Nov. 2015.
[61] Y. Zheng and M. Li, "Fast tag searching protocol for large-scale RFID systems," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 924–934, Jun. 2013.
[62] Y. Qiao, S. Chen, T. Li, and S. Chen, "Energy-efficient polling protocols in RFID systems," in *Proc. ACM MobiHoc*, 2011, p. 25.
[63] Y. Qiao, S. Chen, and T. Li, "Tag-ordering polling protocols in RFID systems," in *RFID as Infrastructure*. Berlin, Germany: Springer, 2013, pp. 59–82.
[64] B. Li, Y. He, W. Liu, L. Wang, and H. Wang, "LocP: An efficient localized polling protocol for large-scale RFID systems," in *Proc. IEEE ICNP*, Nov. 2016, pp. 1–10.
[65] Y. Zhang, L. T. Yang, and J. Chen, *RFID and Sensor Networks: Architectures, Protocols, Security, and Integrations*. Boca Raton, FL, USA: CRC Press, 2009.
[66] *Octane LLRP Version 4.8.0*, ImpinJ, Seattle, WA, USA, 2017.
[67] (2010). *LLRP Toolkit*. [Online]. Available: http://llrp.org
[68] (2017). *ImpinJ*. [Online]. Available: http://www.impinj.com/
[69] (2017). *ThingMagic*. [Online]. Available: http://www.thingmagic.com
[70] (2017). *Alien*. [Online]. Available: http://www.alientechnology.com
[71] (2018). *USRP-Enabled Tash Reader*. [Online]. Available: https://github.com/tagsys/tash2
[72] H. Han *et al.*, "Counting RFID tags efficiently and anonymously," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[73] T. Li, S. Wu, S. Chen, and M. Yang, "Energy efficient algorithms for the RFID estimation problem," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[74] M. Shahzad. *RFID Estimation Tool*. Accessed: Jan. 1, 2019. [Online]. Available: http://www4.ncsu.edu/~mshahza/publications.html
[75] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2004.

Qiongzheng Lin (M'18) received the B.S. degree and Ph.D. degrees from the School of Software, Tsinghua University, China. He is currently a Post-Doctoral Fellow of the Department of Computing, The Hong Kong Polytechnic University. His research interests include radio frequency identification and sensor network, mobile sensing, and pervasive computing. He is a member of the ACM.

Lei Yang received the B.S. and Ph.D. degrees from the Department of Computer Science and Technology, School of Software, XiâŁ™an Jiaotong University. He was a Post-Doctoral Fellow with the School of Software, Tsinghua University. He is currently a Research Assistant Professor with the Department of Computing, The Hong Kong Polytechnic University. His research interests include the Internet of Things, radio frequency identification and backscatters, and wireless and mobile computing. He was a recipient of the Best Paper Award from MobiCom 2014 and MobiHoc 2014, the Best Video Award (Runner-Up) from MobiCom 2016, the Best-In-Session Presentation Award from INFOCOM 2017, the Best Demo Award (Runner-Up) from MobiCom 2018, and the ACM China Doctoral Dissertation Award.



Chunhui Duan (S'16) received the B.S. degree from the School of Software, Tsinghua University, where she is currently pursuing the Ph.D. degree. Her research interests include radio frequency identification, wireless network, and mobile sensing and pervasive computing.



Zhenlin An received the B.S. degree from the School of Information Communicating Engineering, Dalian University of Technology, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. His research interests include the wireless and backscatter communication and mobile computing.