Analog On-Tag Hashing: Towards Selective Reading as Hash Primitives in Gen2 RFID Systems

Lei Yang*, Qiongzheng Lin*, Chunhui Duan*†, Zhenlin An*
*Department of Computing, The Hong Kong Polytechnic University
† School of Software, Tsinghua University
{young,lin,hui,an}@tagsys.org

ABSTRACT

Deployment of billions of Commercial off-the-shelf (COTS) 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 *predicable* 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 10-year untouchable mirage. This work 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 *selective reading*, which is a fundamental and mandatory functionality specified in Gen2 protocol, without any hardware modification and fabrication. We further apply our hash primitives in two typical HEP applications (i.e., cardinality estimation and missing detection) to show the feasibility and effectiveness of Tash. Results from our prototype, which is composed of one ImpinJ reader and 3,000 Alien tags, demonstrate that the new design lowers 60% of the communication overhead in the air. The tash operator can additionally introduce an overhead drop of 29.7%.

CCS CONCEPTS

• **Networks → Cyber-physical networks**; • **Computer systems organization → Embedded and cyber-physical systems**;

KEYWORDS

RFID; Hash Function; Hash Table Function; EPCGlobal Gen2

MobiCom '17, October 16–20, 2017, Snowbird, UT, USA

ACM ISBN 978-1-4503-4916-1/17/10…$15.00
https://doi.org/10.1145/3117811.3117835

1 INTRODUCTION

RFID systems are increasingly used in everyday scenarios, which range from object tracking, indoor localization [60], vibration sensing [61], 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 [12].

1.1 The State-of-the-Art

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 [1] 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 [55]. First, the RFID system utilizes simple modulations (e.g., ON-OFF keying or BPSK) due to the lack of traditional transceiver [9], 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.

1.2 Ten-Year Mirage of HEP

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 HEPs 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 `EPC`s and seeds are known.

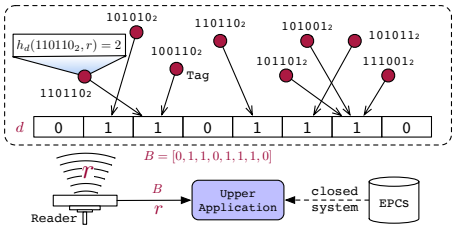


Fig. 1: Hash enabled protocol illustration. In the figure, 8 tags emit one-bit signals in the $h_d(\texttt{EPC}, r)^{th}$ time slots respectively, which are challenged by the random seed r and the frame length d. Finally, the reader abstracts tags' responses as a presence bitmap.

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 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*[1]. 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. We show the following two typical applications as examples to drive the key point:

• *Cardinality estimation.* Estimating the size of a given tag population is required in many applications, such as privacy sensitive systems and warehouse monitoring. Kodialam et al. [18] presented a pioneer estimator. Given that tags select the time slots uniformly because of hashing, the expected number of '0's equals $n_0 = d(1-1/d)^n \approx de^{-n/d}$. 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.

• *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? Sheng and Li [53] 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.

HEPs are advantageous in terms of speed and privacy. HEPs are 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 approximately 96× of the time (i.e., the EPC length equals 96 bits in theory [2]). 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).

HEPs are very promising. 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.

1.3 Why Not Support Hashing?

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 [11, 39], modifies the common hash functions to accommodate resource-constrained RFID tags. The second group [5, 5, 14, 16, 24, 39, 44, 63, 66] designs new lightweight and efficient hash functions dedicatedly for RFID tags. The third group seeks new design of RFID tags like WISP[38] and Moo [67], which gives tags more powerful computing capabilities (e.g., hashing [37]). Unfortunately, as far as we know, none of these work has been really applied in COTS RFID systems yet.

Why is the hash function unfavored? 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. 1 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.

1.4 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[3], while we actually implement

[1] Some work assume the reader can recognize the signal collision, obtaining three results: empty, single and collision.

[2] Actual case in practice would be less than this estimate due to other extra jobs, such as setup time, query time, etc.

[3] 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.

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 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 [6]. 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.

2 RELATED WORK

We review the related work from two aspects: the designs of hash functions and hash enabled protocols.

Design of hash function. Feldhofer and Rechberger [11] 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 [5, 5, 11, 14, 16, 24, 39, 39, 44, 63, 66]. We sketch the primary designs and their features in Table. 1. For example, Bogdanov et al. [5] propose a hardware-optimized block cipher, PRESENT, designed with area and power constraints. The follow-up work [44] 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. Lim and Korkishko [24] 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.

Design of hash enabled protocol. To drive our key point, we conduct a brief survey of previous related works. 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.* Dozens of estimators [8, 13, 17, 19, 28, 29, 40, 41, 45, 47, 51, 52, 56, 71, 72] have been proposed in the past decade. For example, Qian et al. [41] proposed an estimation scheme called lottery frame. Shahzad and Liu [47] estimated the number based on the average run-length of ones in a bit string received using the FSA. In particular, they claimed that their protocol is compatible with Gen2 systems. However, their scheme still requires modifying the communication protocol, and thus, it fails to work with COTS Gen2 systems. By contrast, our prototype can operate in COTS Gen2 systems as demonstrated in this study. (2) *Missing detection.* The missing detection problem was firstly mentioned in [53]. Thereafter, many follow-up works [21, 22, 27, 31–33, 36, 48, 49, 54, 59, 64, 65, 68, 73] have started to study the issue of false positives resulting from the collided slots by using multiple bitmaps. (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 [25, 50, 58] have proposed continuous reading protocols that can incrementally collect tags in each step using the bitmap. For example, Sheng et al. [50] aimed to preserve the tags collected in the previous round and collect only unknown tags. Xie et al. [58] conducted an experimental study on mobile reader scanning. Liu et al. [25] initially estimated the number of overlapping tags in two adjacent inventories and then performed an effective incremental inventory. (4) *Data mining.* These works [26, 34, 35, 51, 57] discuss how to discover potential information online through bitmaps. For example, Sheng et al. [51] proposed to identify the popular RFID categories using the group testing technique. Xie et al. found histograms over tags through a small number of bitmaps[57]. Luo et al. [34, 35] determined whether the number of objects in each group was above or below a threshold. Liu et al. [26] proposed a new online classification protocol for a large number of groups. (5) *Tag searching.* These works[30, 70] 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 et al. [70] utilized bitmaps to aggregate a large volume of RFID tag information and to search the tags quickly. Liu et al. [30] 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.* [20, 42, 43] 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.

Table 1: Performance overview of current hash functions [4]

Hash functions	Key size	GE	Power	Clock cycles
SHA-256[11]	256	10,868	$15.87\mu A$	1,128
SHA-1 [11]	160	8,120	$10.68\mu A$	1,274
AES [10]	128	3,400	$8.15\mu A$	1,032
MAME[63]	256	8,100	$5.16\mu A$	96
MD5 [11]	128	8,400	-	612
MD4 [11]	128	7,350	-	456
PRESENT-80 [5]	80	1,570	-	32
PRESENT-80 [44]	80	1,075	-	563
PRESENT-128 [6]	128	1,886	-	32
DES [39]	56	2,309	-	144
mCrypton [24]	96	2,608	-	13
TEA [66]	128	2,355	-	64
HIGHT [16]	128	3,048	-	34
DESXL [39]	184	2,168	-	144
Grain & Trivium[14]	80	2,599	-	1

3 OVERVIEW

Tash provides a group of hash primitives for HEPs. This section presents the scope and formally defines our hash primitives.

3.1 Scope

Despite clear and certain specifications, the implementation of the Gen2 protocol still varies with readers and manufacturers because of firmware bugs or compromises, especially in early released reader devices, according to our compatibility report presented in §7. Here, we firmly claim that our design and implementation strictly follow the specifications of the Gen2 and LLRP protocols (refer to §6). The framework works with any Gen2-compatible readers and tags. The performance losses caused by defects in devices are outside the scope of our discussion.

3.2 Definitions of Hash Primitives

Before delving into details, we formally define the hash primitives that the HEPs require, from a high-level.

Definition 1 (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* EPCs *of the tags and random seeds.*

Tash function and tash value. As the above definition specifies, an l-bit tash function takes an EPC t and a random seed r as input and outputs an l-bit integer i, denoted by:

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

We call l the *dimension* of tash function (i.e., $l = 0, 1, 2, \dots$). The tash value i is an integer $\in [0, 2^l - 1]$. Similar to other common hash function, the tash function has three basic characteristics. First, the output changes significantly when the two parameters are altered. Second, its output is uniformly distributed within the given range, and predicable if all inputs are known. Third, the hash values are accessible.

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.*

Tash table function and tash table. 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., $T = \{t_1, t_2, \dots, t_n\}$) 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 black box equipping with tash table function. When inputing a random seed, the box 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 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.

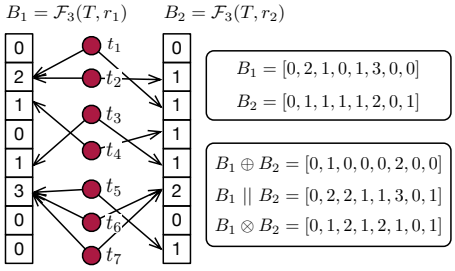


Fig. 2: Illustration of tash operators. The left shows two independent tash tables, while the right shows the results of the two tash tables with tash AND, OR and XOR.

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.

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 $B[i] = |\{t | f_l(t, r_1) = i \& f_l(t, r_2) = i\}|$.*

The tash AND is aimed at obtaining the common intersection of corresponding entries from two tash tables. For example, as shown in Fig. 2, $B_1[1]$ 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.

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 $B[i] = |\{t | f_l(t, r_1) = i || f_l(t, r_2) = i\}|$.*

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. 2, $(B_1 || B_2)[5] = |\{t_5, t_6, t_7\}| = 3$ although $B_1[5] + B_2[5] = 5$ because t_6 and t_7 appear twice in the two tash tables.

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 $B[i] = |\{t | f_l(t, r_1) = i \,\&\, f_l(t, r_2) \neq i\}|$.*

The tash XOR is aimed at obtaining the total number of the set difference. As Fig. 2 shows, $B_1[5] = |\{t_5, t_6, t_7\}|$ and $B_2[5] = |\{t_6, t_7\}|$. Then $(B_1 \otimes B_2)[5] = |\{t_5\}| = 1$.

The above operators can be applied in a series of tash tables with the same dimension for a hybrid operation, e.g., $B_1 \oplus B_2 || B_3$. Tash AND and OR satisfy operational laws such as associative law and commutative law, e.g., $B_1 \oplus B_2 = B_2 \oplus B_1$. The design of tash operators is one of the attractive features of the tash framework, and it has never been proposed before. More importantly, we design and implement these operators in the physical layer to provide one-stop acquisition solution.

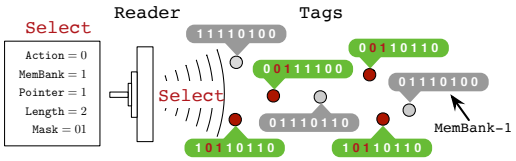


Fig. 3: Illustration of selective reading in Gen2. There are total 7 tags covered by a reader. The reader initiates a selective reading using a `Select` command, which let these tags (highlighted with dark red) whose data starting at the first bit with a length of 2 bits in the `MemBank-1` equals 01_2, participate in the incoming inventory, while other tags (with gray color) that do not meet the condition remain silent.

3.3 Solution Sketch

Tash is designed to reduce the overhead for air communications. It runs in the middle of the reader and upper application. The upper application passes a pair of arguments (i.e., r and l), or pairs of arguments (as well as operators) to Tash. On the basis of the arguments, Tash generates one or more configuration files to manipulate the reader's reading. Finally, Tash abstracts the reading results to a tash table, which is returned to the upper application.

The rest of the paper is structured as follows. We firstly present the tash design in §4. We next demonstrate the usage of our hash primitives in two classic applications in §5. We then introduce the tash implementation using LLRP interfaces in §6. In §7 and §8, we present the microbenchmark and the usage evaluation. Finally, we conclude in §9 and present future directions.

4 TASH DESIGN

In this section, we introduce the background of Gen2 protocol, and then present the technical details of our designs.

4.1 Background of Gen2 Protocol

The Gen2 standard defines air communication between readers and tags. On the basis of [1, 69], we introduce its four central functions we will employ:

F1: Memory Model. Gen2 specifies a simple tag memory model (pages 44 ~ 46 of [1]). 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 specific information. (4) `MemBank-3` is a customized storage that contains user-defined data. We can use `Read` or `Write` commands to read or write data into these banks.

F2: Selective Reading. Gen2 specifies that each inventory must be started with `Select` command (pages 72~73 of [1]). 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). The following fields are presented for this study.

• `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 of tags.

• `Action`. This field specifies an action that will be will performed by the tags. Table. 2 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 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 [1].

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 (see page 76~80 of [1].) starts a new *inventory round* over a subset of tags, chosen by the previous `Select` commands.

4.2 Design of the Tash Function

An l-bit tash function is essentially a hash function that is indispensable to HEPs. We design the tash function while following the three principles outlined as follows. The first principle requires that

Table 2: 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	deassert`SL`
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`

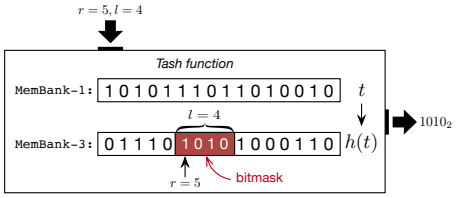


Fig. 4: Illustration of a tash function. The result of a tash function of $f_4(t, 5)$ is equal to 1010_2.

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 6 (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.*

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 an 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). Due to space limit, more discussions about the design are presented in our technical report [62].

4.3 Design of the Tash Table Function

The tash table function treats a reader and multiple tags as if they were a single virtual node, outputting a tash table. 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 selection command (i.e., `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. 2. In particular, if $u = 1$, then each tag will reply with a truncated `EPC` number.

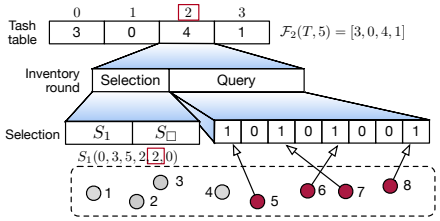


Fig. 5: Illustration of creating a tash table. Given that $r = 5$ and $l = 2$, $\mathcal{F}_2(T, 5) = [3, 0, 4, 1]$. Zooming into the 3^{rd} entry-inventory, tags t_5, t_6, t_7 and t_8 are selected to join the inventory. $S_\square$ means this is the end command.

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 selective reading with two selection commands as follows:

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

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. 6. 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 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. Four 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.

4.4 Design of Tash Operators

A tash operator is connected to two tash tables, which have the same dimensions but are constructed using two different seeds. When two seeds, r_1 and r_2, are given, we can obtain two l-bit tash tables: $B_1 = \mathcal{F}_l(T, r_1)$ and $B_2 = \mathcal{F}_l(T, r_2)$. Our objective is to obtain a final tash table B by performing one of the subsequent tash operators on B_1 and B_2.

Tash AND. If $B = B_1 \oplus B_2$, 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. 2, 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, $S_{\square}$ is reserved for the one-bit presence signal.

Tash OR. If $B = B_1 || B_2$, 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:

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

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. 2) 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.

Tash XOR. If $B = B_1 \otimes B_2$, 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:

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

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:

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

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:

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

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.

4.5 Discussion

Comparison with bitmap. A tash table evidently takes a considerably longer time to obtain than a bitmap because a bitmap requires only one round of inventory, whereas a tash table requires multiple rounds. The additional time consumption is the trade-off for practicality because the reply of a COTS tag at the slot level is out of 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.

Embedded pseudo-random function. Qian et al. [40] and Shahzad et al. [47] 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.

5 TASH USAGE

This section revisits two classic problems of HEPs for usage study. We propose two practical solutions that use tash primitives for these problems. Note that in spite of two demonstration presented in this section, our tash primitives especially the tash table can serve any kind of HEPs.

5.1 Usage I: Cardinality Estimation

Cardinality estimation aims to estimate the total number of tags by using one-bit presence signals that are received without collecting each individual tag. The problem is formally defined as follows:

Problem 1. *When a tag population of an unknown size n, a tolerance of $\beta \in (0, 1)$, and a required confidence level of $\alpha \in (0, 1)$ is given, how can the number of tags $\widehat{n}$ be estimated such that* $\Pr(|\widehat{n} - n| \leq \beta n) \geq \alpha$*?*

A naive method would be to add all the entries of a tash table together or let all tags reply at the first entry. Since each tag participates in one and only one entry-inventory, the final number is

exactly equal to n. Keep in mind that our each entry corresponds to a complete round inventory. The naive method is equivalent to collecting them all, which is extremely time-consuming. We subsequently provide a reliable solution in a probabilistic way.

Proposed Estimator. We leverage the number of tags mapped into the first entry of a tash table to estimate n. Let X be the random variable to indicate the value of the first entry of a tash table. Since n tags are randomly and uniformly assigned into 2^l entires, we have

$$\Pr(X = m) = \binom{n}{m} p^m (1-p)^{n-m} \tag{3}$$

where $p = 1/2^l$. Evidently, variable X follows a standard Binomial distribution with the parameters n and p, i.e., $X \sim B(n, p)$. Therefore the expected value $\mu = np$ and variance $\delta = np(1-p)$. By equating the expected value and an instanced value m, our estimator $\widehat{n}$ is given by:

$$\widehat{n} = m/p = m2^l \tag{4}$$

The estimator only requires the first entry of the hash table, so it skips inventories of other entries. We can choose an appropriate l to ensure the estimation error within the given tolerance level β with a confidence of greater than α according to Theorem 1.

Theorem 1. *The optimal dimension of the tash table is equal to* $\lceil \log_2 \frac{\sqrt{2}\cdot erf^{-1}(\alpha)}{\sqrt{2}\cdot erf^{-1}(\alpha)-\beta} \rceil$, *which results in an estimation error* $\leq \beta$ *with a probability of at least* α.

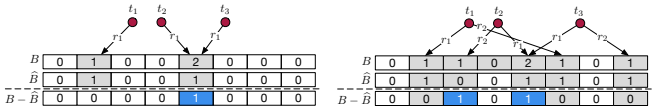
Proof. Please refer to [62] for the proof. □

5.2 Usage II: Missing Tag Detection

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, especially when thousands of tags are present. We formally define the problem of detecting missing tags in Problem 2. 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 2. *How to quickly identify m missing out of n tags with a false positive rate of* γ *at most?*

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. 6(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.



(a) Tashing once (b) Tashing twice

Fig. 6: 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.

Inspired by the Bloom filter[7], we perform k tashings to identify the missing tags as follows:

$$B = \mathcal{F}_l(T, r_1) || \ldots || \mathcal{F}_l(T, r_k) \tag{5}$$

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. 6(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. 6(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 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

$$p = (1 - e^{-km/L})^k \tag{6}$$

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 [7] 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 $p \leq \gamma$. After some algebraic manipulation, we find

$$L \geq \frac{m \log_2(1/\gamma)}{\ln 2} = m \log_2 e \cdot \log_2(1/\gamma) = 1.44 m \log_2(1/\gamma) \tag{7}$$

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

Theorem 2. *Setting the table dimension to* $\lceil \log_2(1.44 m \log_2(1/\gamma)) \rceil$ *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.

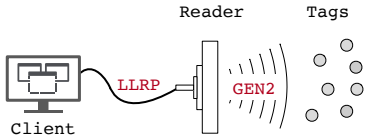


Fig. 7: Gen2 vs. LLRP. Gen2 is the air protocol between a reader and tags while LLRP is the driver protocol between a client computer and a reader. Our framework leverages LLRP to manipulate a reader to broadcast Gen2 commands that we need.

6 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. 7, 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, 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.

Reader operation. RO defines the inventory parameters specified in the Gen2 protocol, such as bitmask, antenna power, and frequency. Fig. 8 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 `C1G2Filter`s. The filter functions as a bitmask. We can set multiple selection commands by adding multiple `C1G2Filter`s.

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 `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.

7 MICROBENCHMARK

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

7.1 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

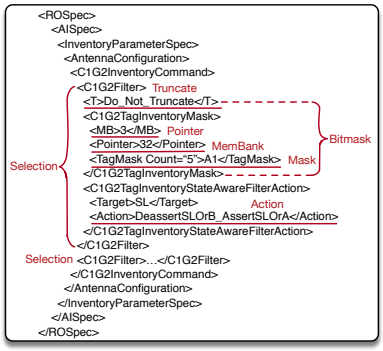


Fig. 8: LLRP RO specification. The XML document defines various parameters that are required for `Select` command.

placed in front of a reader antenna. 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.

7.2 Compatibility Investigation

First, we investigate the compatibility of Gen2 across 10 different types of readers and 18 different types of tags in 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 [3], the Mercury6, Sargas and M6e models from ThingMagic [4], as well as the ALR-F800, 9900+, 9680 and 9650 models from Alien [2]. 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. 3. 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) 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`

Table 3: 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	–	–

Table 4: Summary of Gen2-compatibility on tag.

	ImpinJ Monza									Alien ALN								
Commands	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	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

and `AISpec`s that a reader should support. Our practical tests suggest that the ImpinJ series supports 4 `C1G2Filter`s and 16 `AISpec`s, which means that we can only use a maximum of four tash operators each time.

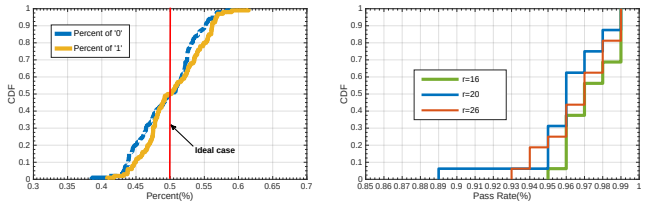
Tag compatibility. We investigate 9 chip models from ImpinJ Monza series and 9 additional models from Alien ALN series. The majority of tags on the market contain these 18 models of chips and customized antennas. Table. 4 summarizes the result of our investigation, from which we have the subsequent findings. (1) Tags reserve 96 ∼ 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 ∼ 512 bits. The de facto standard has become 128 bits. (4) All tags are claimed to support the `Truncate` command according to their public data sheets. However, we have no idea about their real serviceability due to the lack of `Truncate`-supportable reader available for practical tests. In our future work, we plan to utilize USRP for further tests.

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.

7.3 Tash Function

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

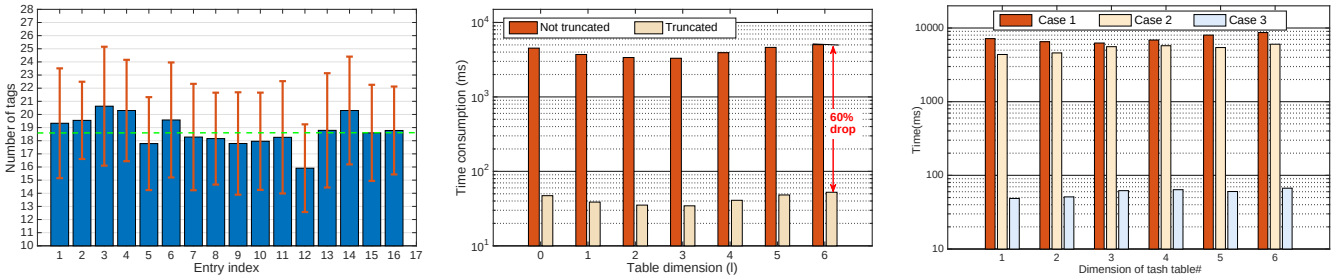
Randomness. Randomness is the most important metric for a hash function. It requires that the outputs of a hash function must be uniformly distributed. To validate the randomness of the tash function, we collect 99, 886 real `EPC` numbers from our partner (i.e., an international logistics company), which 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. 3), we choose to use only the first 32 bits for our tests. We traverse r and l from 0 ∼ 31 and 1 ∼ $32 - r$ 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. 9(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. 9(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.



(a) Distributions of percents of '0' and '1' (b) Results of random test

Fig. 9: 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 random ness tests.

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



(a) Balance of tash table

(b) Performance of tash table function

(c) Performance of tash operator

Fig. 10: Evaluation of tash table function and tash operator. (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 tash tables with different dimensions. **(c) shows the time consumption on performing OR on two tash tables.**

our benchmarks without any exceptions. This shows that the selective reading is well supported by the manufactures and is both stable and reliable.

7.4 Tash Table Function

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

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. 10(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.

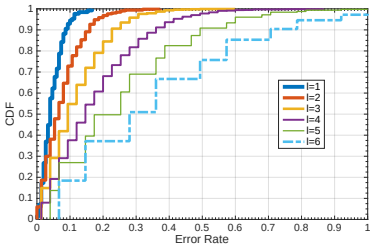
Gathering speed. We then consider the time consumption of gathering a tash table. Fixing the random seed, we vary the table dimension l from 0 to 6. We then measure the time taken on gathering a tash table with the deployed 300 tags. Fig. 10(b) shows the resulting time as a function of the table dimension. From the immediately above-mentioned figure, we can observe the subsequent findings. (1) When $l = 0$ without truncating reply, the result is equivalent to collecting 300 complete `EPC`s of all the tags. Such time consumption (i.e., $4,524ms$) is viewed as our baseline. (2) 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. (3) However, when $l > 4$, the performance of "divide and conquer" approach starts to deteriorate. The ImpinJ reader supports 16 `AISpec`s at most (see Table. 3). 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. (4) 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 cannot measure the actual time spent on collecting truncated `EPC`s. We can only utilize the least-square algorithm to estimate the transmission time for a one-bit presence signal. Our fitting results show that truncating reply would introduce about 60% drop of the overhead at least.

7.5 Tash Operators

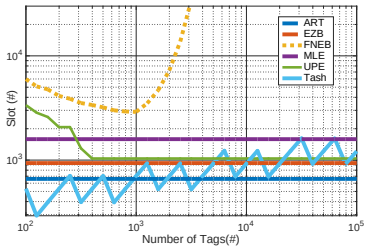
Finally, we investigate the performance of tash operators. Superior to existing HEPs, these operators allow us to perform set operations on-tag and conduct a one-stop inventory. In particular, we show the performance of OR as a representative across 300 tags. The tests for other operators are similar and omitted due to the space limitation. In the experiments, we fix the two random seeds but change the dimension of tash table. Fig. 10(c) 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.

8 USAGE EVALUATION

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



(a) Testbed



(b) Large-scale simulation

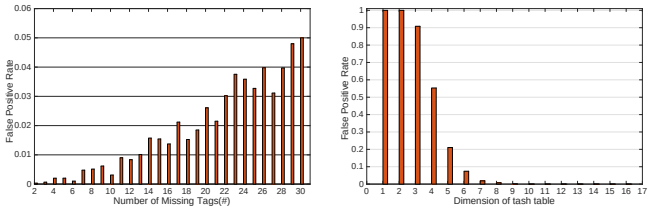
Fig. 11: Cardinality estimation. (a) shows the CDF of error rates for estimating 300 tags with our testbed. (b) shows the estimation comparisons with other theoretical algorithms with simulation.

8.1 Usage 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. 11(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. We numerically simulate in Matlab using tash scheme as well as other five prior RFID estimation schemes: UPE[18], EZB[19], FNEB[15], MLE[23], ART[47]. We implement these schemes by referring to the RFID estimation tool developed by Shahzad[46]. Fig. 11(b) shows the time cost with a varying n given $\alpha = 0.9$ and $\beta = 0.08$. We observe that our scheme is 5× 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.



(a) $k = 2$ and $l = 8$ (b) $k = 2$ and $m = 10$

Fig. 12: Missing detection with 300 tags. (a) The resulted FPRs as function of the missing number (b) as function of the dimension of tash tables.

8.2 Usage II: Missing Detection

Finally, we evaluate the effectiveness of missing detection in real case. We randomly remove m tags from the testbed. 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. 12(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. 12(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. 2 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.

9 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.

Acknowledgments. The research is supported by GRF/ECS (NO. 25222917), NSFC General Program (NO. 61572282) and Hong Kong Polytechnic University (NO. 1-ZVJ3). We thank all the reviewers for their valuable comments and helpful suggestions, and particularly thank Eric Rozner for the shepherding.

REFERENCES

[1] 2004. EPCglobal Gen2 Specification. www.gs1.org/epcglobal. (2004).
[2] 2017. Alien. http://www.alientechnology.com. (2017).
[3] 2017. ImpinJ, Inc. http://www.impinj.com/. (2017).
[4] 2017. ThingMagic. http://www.thingmagic.com. (2017).
[5] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. 2007. PRESENT: An ultra-lightweight block cipher. In *CHES*, Vol. 4727. Springer, 450–466.
[6] Andrey Bogdanov, Gregor Leander, Christof Paar, Axel Poschmann, Matt JB Robshaw, and Yannick Seurin. 2008. Hash functions and RFID tags: Mind the gap. In *Proc. of IACR CHES*.
[7] Andrei Broder and Michael Mitzenmacher. 2004. Network applications of bloom filters: A survey. *Internet Mathematics* 1, 4 (2004), 485–509.
[8] Binbin Chen, Ziling Zhou, and Haifeng Yu. 2013. Understanding RFID counting protocols. In *Proc. of ACM MobiCom*.
[9] Daniel M Dobkin. 2012. *The RF in RFID: UHF RFID in Practice*. Newnes.
[10] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. 2004. Strong authentication for RFID systems using the AES algorithm. In *CHES*, Vol. 4. Springer, 357–370.
[11] Martin Feldhofer and Christian Rechberger. 2006. A case against currently used hash functions in RFID protocols. In *On the move to meaningful internet systems 2006: OTM 2006 workshops*. Springer, 372–381.
[12] Frost and Sullivan. 2011. Global RFID healthcare and pharmaceutical marke. *Industry Report* (2011).
[13] Wei Gong, Kebin Liu, Xin Miao, and Haoxiang Liu. 2014. Arbitrarily accurate approximation scheme for large-scale rfid cardinality estimation. In *Proc. of IEEE INFOCOM*.
[14] Tim Good and Mohammed Benaissa. 2007. Hardware results for selected stream cipher candidates. *State of the Art of Stream Ciphers* 7 (2007), 191–204.
[15] Hao Han, Bo Sheng, Chiu C Tan, Qun Li, Weizhen Mao, and Sanglu Lu. 2010. Counting RFID tags efficiently and anonymously. In *Proc. of IEEE INFOCOM*.
[16] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, et al. 2006. HIGHT: A new block cipher suitable for low-resource device. In *CHES*, Vol. 4249. Springer, 46–59.
[17] Yuxiao Hou, Jiajue Ou, Yuanqing Zheng, and Mo Li. 2015. PLACE: Physical layer cardinality estimation for large-scale RFID systems. In *Proc. of IEEE INFOCOM*.
[18] Murali Kodialam and Thyaga Nandagopal. 2006. Fast and reliable estimation schemes in RFID systems. In *Proc. of ACM MobiCom*.
[19] Murali Kodialam, Thyaga Nandagopal, and Wing Cheong Lau. 2007. Anonymous tracking using RFID tags. In *Proc. of IEEE INFOCOM*.
[20] Binbin Li, Yuan He, Wenyuan Liu, Lin Wang, and Hongyan Wang. 2016. LocP: An efficient Localized Polling Protocol for large-scale RFID systems. In *Proc. of IEEE ICNP*.
[21] Tao Li, Shigang Chen, and Yibei Ling. 2010. Identifying the missing tags in a large RFID system. In *Proc. of ACM MobiHoc*.
[22] Tao Li, Shigang Chen, and Yibei Ling. 2013. Efficient protocols for identifying the missing tags in a large RFID system. *IEEE/ACM Transactions on Networking* 21, 6 (2013), 1974–1987.
[23] Tao Li, Samuel Wu, Shigang Chen, and Mark Yang. 2010. Energy efficient algorithms for the RFID estimation problem. In *Proc. of IEEE INFOCOM*.
[24] Chae Hoon Lim and Tymur Korkishko. 2005. mCrypton-a lightweight block cipher for security of low-cost RFID tags and sensors. In *WISA*, Vol. 3786. Springer, 243–258.
[25] Haoxiang Liu, Wei Gong, Xin Miao, Kebin Liu, and Wenbo He. 2014. Towards adaptive continuous scanning in large-scale rfid systems. In *Proc. of IEEE INFOCOM*.
[26] Jia Liu, Bin Xiao, Shigang Chen, Feng Zhu, and Lijun Chen. 2015. Fast RFID grouping protocols. In *Proc. of IEEE INFOCOM*. 1948–1956.
[27] Xiulong Liu, Keqiu Li, Geyong Min, Yanming Shen, Alex X Liu, and Wenyu Qu. 2014. A multiple hashing approach to complete identification of missing RFID tags. *IEEE Transactions on Communications* 62, 3 (2014), 1046–1057.
[28] Xiulong Liu, Keqiu Li, Heng Qi, Bin Xiao, and Xin Xie. 2014. Fast counting the key tags in anonymous RFID systems. In *Proc.of IEEE ICNP*.
[29] Xiulong Liu, Bin Xiao, Keqiu Li, Jie Wu, Alex X Liu, Heng Qi, and Xin Xie. 2015. RFID cardinality estimation with blocker tags. In *Proc. of IEEE INFOCOM*.
[30] Xuan Liu, Bin Xiao, Shigeng Zhang, Kai Bu, and Alvin Chan. 2015. STEP: A time-efficient tag searching protocol in large RFID systems. *IEEE Trans. Comput.* 64, 11 (2015), 3265–3277.
[31] Wen Luo, Shigang Chen, Tao Li, and Shiping Chen. 2011. Efficient missing tag detection in RFID systems. In *Proc. of IEEE INFOCOM*.
[32] Wen Luo, Shigang Chen, Tao Li, and Yan Qiao. 2012. Probabilistic missing-tag detection and energy-time tradeoff in large-scale RFID systems. In *Proc. of ACM MobiHoc*.
[33] Wen Luo, Shigang Chen, Yan Qiao, and Tao Li. 2014. Missing-tag detection and energy–time tradeoff in large-scale RFID systems with unreliable channels. *IEEE/ACM Transactions on Networking* 22, 4 (2014), 1079–1091.
[34] Wen Luo, Yan Qiao, and Shigang Chen. 2013. An efficient protocol for RFID multigroup threshold-based classification. In *Proc. of IEEE INFOCOM*. 890–898.
[35] Wen Luo, Yan Qiao, Shigang Chen, and Min Chen. 2016. An efficient protocol for RFID multigroup threshold-based classification based on sampling and logical bitmap. *IEEE/ACM Transactions on Networking* 24, 1 (2016), 397–407.
[36] Cunqing Ma, Jingqiang Lin, and Yuewu Wang. 2012. Efficient missing tag detection in a large RFID system. In *Proc. of IEEE TrustCom*.
[37] Christian Pendl, Markus Pelnar, and Michael Hutter. 2012. Elliptic curve cryptography on the WISP UHF RFID tag. *RFID. Security and Privacy* (2012), 32–47.
[38] Matthai Philipose, Joshua R Smith, Bing Jiang, Alexander Mamishev, Sumit Roy, and Kishore Sundara-Rajan. 2005. Battery-free wireless identification and sensing. *IEEE Pervasive computing* 4, 1 (2005), 37–45.
[39] Axel Poschmann, Gregor Leander, Kai Schramm, and Christof Paar. 2007. New Light-Weight DES Variants Suited for RFID Applications, proceedings of Fast Software Encryption 14. *Lecture Notes in Computer Science, Springer (to appear)* (2007).
[40] Chen Qian, Yunhuai Liu, Hoilun Ngan, and Lionel M Ni. 2010. Asap: Scalable identification and counting for contactless rfid systems. In *Proc. of IEEE ICDCS*.
[41] Chen Qian, Hoilun Ngan, Yunhao Liu, and Lionel M Ni. 2011. Cardinality estimation for large-scale RFID systems. *IEEE Transactions on Parallel and Distributed Systems* 22, 9 (2011), 1441–1454.
[42] Yan Qiao, Shigang Chen, and Tao Li. 2013. Tag-ordering polling protocols in RFID systems. In *RFID as an Infrastructure*. Springer, 59–82.
[43] Yan Qiao, Shigang Chen, Tao Li, and Shiping Chen. 2011. Energy-efficient polling protocols in RFID systems. In *Proc. of ACM MobiHoc*.
[44] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. 2008. Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents. In *CARDIS*, Vol. 5189. Springer, 89–103.
[45] Vahid Shah-Mansouri and Vincent WS Wong. 2011. Cardinality estimation in RFID systems with multiple readers. *IEEE Transactions on Wireless Communications* 10, 5 (2011), 1458–1469.
[46] Muhammad Shahzad. [n. d.]. RFID estimation tool. http://www4.ncsu.edu/~mshahza/publications.html. ([n. d.]).
[47] Muhammad Shahzad and Alex X Liu. 2012. Every bit counts: fast and scalable RFID estimation. In *Proc. of ACM MobiCom*.
[48] Muhammad Shahzad and Alex X Liu. 2015. Expecting the unexpected: Fast and reliable detection of missing RFID tags in the wild. In *Proc. of IEEE INFOCOM*.
[49] Muhammad Shahzad and Alex X Liu. 2016. Fast and Reliable Detection and Identification of Missing RFID Tags in the Wild. *IEEE/ACM Transactions on Networking* 24, 6 (2016), 3770–3784.
[50] Bo Sheng, Qun Li, and Weizhen Mao. 2010. Efficient continuous scanning in RFID systems. In *Proc. of IEEE INFOCOM*.
[51] Bo Sheng, Chiu Chiang Tan, Qun Li, and Weizhen Mao. 2008. Finding popular categories for RFID tags. In *Proc. of ACM MobiHoc*.
[52] W-K Sze, W-C Lau, and O-C Yue. 2009. Fast RFID counting under unreliable radio channels. In *Proc. of IEEE ICC*.
[53] Chiu C Tan, Bo Sheng, and Qun Li. 2008. How to monitor for missing RFID tags. In *Proc. of IEEE ICDCS*.
[54] Chiu C Tan, Bo Sheng, and Qun Li. 2010. Efficient techniques for monitoring missing RFID tags. *IEEE Transactions on Wireless Communications* 9, 6 (2010), 1882–1889.
[55] Jue Wang, Haitham Hassanieh, Dina Katabi, and Piotr Indyk. 2012. Efficient and reliable low-power backscatter networks. In *Proc. of ACM SIGCOM*. ACM, 61–72.
[56] Qingjun Xiao, Bin Xiao, and Shigang Chen. 2013. Differential estimation in dynamic RFID systems. In *Proc. of IEEE INFOCOM*.
[57] Lei Xie, Hao Han, Qun Li, Jie Wu, and Sanglu Lu. 2014. Efficiently collecting histograms over rfid tags. In *Proc. of IEEE INFOCOM*.
[58] Lei Xie, Qun Li, Xi Chen, Sanglu Lu, and Daoxu Chen. 2013. Continuous scanning with mobile reader in RFID systems: An experimental study. In *Proc. of ACM MobiHoc*. ACM, 11–20.
[59] Wei Xie, Lei Xie, Chen Zhang, Qiang Wang, Jian Xu, Quan Zhang, and Chaojing Tang. 2014. RFID seeking: Finding a lost tag rather than only detecting its missing. *Journal of Network and Computer Applications* 42 (2014), 135–142.
[60] Lei Yang, Yekui Chen, Xiang-Yang Li, Chaowei Xiao, Mo Li, and Yunhao Liu. 2014. Tagoram: real-time tracking of mobile RFID tags to high precision using COTS devices. In *Proc. of ACM MobiCom*.
[61] Lei Yang, Yao Li, Qiongzheng Lin, Xiang-Yang Li, and Yunhao Liu. 2016. Making sense of mechanical vibration period with sub-millisecond accuracy using backscatter signals. In *Proc. of ACM MobiCom*.
[62] Lei Yang, Qiongzheng Lin, Chunhui Duan, and Zhenlin An. 2017. Analog On-Tag Hashing: Towards Selective Reading as Hash Primitives in Gen2 RFID Systems. *Technical Report (arXiv:1707.08883)* (2017).
[63] Hirotaka Yoshida, Dai Watanabe, Katsuyuki Okeya, Jun Kitahara, Hongjun Wu, Özgül Küçük, and Bart Preneel. 2007. MAME: A compression function with reduced hardware requirements. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 148–165.

[64] Jihong Yu, Lin Chen, and Kehao Wang. 2015. Finding Needles in a Haystack: Missing Tag Detection in Large RFID Systems. *arXiv preprint arXiv:1512.05228* (2015).
[65] Jihong Yu, Lin Chen, Rongrong Zhang, and Kehao Wang. 2016. On Missing Tag Detection in Multiple-group Multiple-region RFID Systems. *IEEE Transactions on Mobile Computing* (2016).
[66] Y Yu, Y Yang, Y Fan, and H Min. [n. d.]. Security Scheme for RFID Tag: Auto-ID Labs white paper WP-HARDWARE-022. http://www.autoidlabs.org/. ([n. d.]).
[67] Hong Zhang, Jeremy Gummeson, Benjamin Ransford, and Kevin Fu. 2011. Moo: A batteryless computational RFID and sensing platform. *University of Massachusetts Computer Science Technical Report UM-CS-2011-020* (2011).
[68] Rui Zhang, Yunzhong Liu, Yanchao Zhang, and Jinyuan Sun. 2011. Fast identification of the missing tags in a large RFID system. In *Proc. of IEEE SECON*.
[69] Yan Zhang, Laurence T Yang, and Jiming Chen. 2009. *RFID and sensor networks: architectures, protocols, security, and integrations*. CRC Press.
[70] Yuanqing Zheng and Mo Li. 2013. Fast tag searching protocol for large-scale RFID systems. *IEEE/ACM Transactions on Networking* 21, 3 (2013), 924–934.
[71] Yuanqing Zheng and Mo Li. 2013. ZOE: Fast cardinality estimation for large-scale RFID systems. In *Proc. of IEEE INFOCOM*.
[72] Yuanqing Zheng and Mo Li. 2014. Towards more efficient cardinality estimation for large-scale RFID systems. *IEEE/ACM Transactions on Networking* 22, 6 (2014), 1886–1896.
[73] Yuanqing Zheng and Mo Li. 2015. P-mti: Physical-layer missing tag identification via compressive sensing. *IEEE/ACM Transactions on Networking* 23, 4 (2015), 1356–1366.