Kallima: A Tag-Reader Protocol for Privacy Enhanced RFID System

Yusuke Doi, Shirou Wakayama, Masahiro Ishiyama, Satoshi Ozaki, and Atsushi Inoue

Communication Platform Laboratory,
Corporate Research & Development Center, TOSHIBA Corporation,
1 Komukai-Toshiba-cho, Saiwai-ku, Kawasaki, Kanagawa, Japan
{ydoi, shirou, masahiro, fe, inoue}@isl.rdc.toshiba.co.jp

Abstract. Privacy is a major concern with RFID tags and many solutions have been proposed. As many approach requires secure hash function on each tag, cost of tags imposed by those solutions is significantly high for wide development. We propose a protocol that uses pre-calculated Bloom filter to send tag identity for increased privacy with little additional cost per tag. In our approach, secure hash function calculation is done in tag production phase and each tag does not have any hash functions. Instead, each tag must have random number generator and volatile memory.

1 Background

RFID (Radio-Frequency IDentification) tags are emerging devices that make automatic identification more convenient. However, convenient automatic identification is sometimes dangerous with respect to privacy. Some consumer groups oppose to RFID due to the privacy infringement. If a store uses RFID tags for inventory control and leaves the tags on the items after the checkout, customers themselves may be traced by the RFID tags. Tags on some personal belongings like purses, shoes, and clothes can help tracking a customer without being noticed. In this paper, we call this problem *owner tracing*. To keep user's privacy, one can use high-end tags. Some of high-end tags like Ferica [1] may provide high level security using cryptographic function.

On the other hand, low cost tags are needed for wide deployment of RFID systems. Because a major objective of RFID is cost reduction of item handling in distribution systems, RFID system must be low cost as much as possible to deploy. Peoples in Auto-ID Labs protests that a tag must be as cheap as 5 or 10 cents for wide deployment[1]. At the same time, a 5 cents tag is a difficult target to accomplish[2]. Hence, RFID tag system that aims wide deployment can afford very little cost for security per tag.

1.1 Assumptions

We assume the following entities in our RFID system model.

A victim has an item. The item has an RFID tag attached on it. RFID tag sends signals on read request.

[1] http://www.sony.net/Products/felica/

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 375–382, 2005.

An attacker has an RFID reader that can read signal of RFID tags within its range. We assume the attacker has no other clue than RFID signal to identify what item is in the range. For example, attacker may use a video camera with the reader to bind RFID signals and people. In case if an attacker can bind a RFID signal with the victim, the attack becomes easier. The case is discussed in the section 4.1.

We also assume the victim and other people appear within the attacker's range in random pattern. The reader captures many signals from those people and the attacker has no knowledge which one is from the victim's. In addition, a reader captures other casual signals from other tags that comes by chance. We assume arrival pattern and signals from those tags are random.

Owner tracing occurs as the reader successfully identify the signal from the victim's item. More precisely, attacker receives a sequence of signals from the reader. The attacker can trace a victim if the attacker can find relation between two or more signals of the victim's tag out of the sequence. In other words, owner tracing fails if attacker cannot find relation between the sequence of signals from the reader.

In addition, our model of trust assumes that a reader is a legitimate reader for a RFID tag if the reader knows the tag's identifier. The identifier itself is the shared secret to make trust relationship between the tag and the reader.

1.2 Our Goals

Our goal is to enable the development of RFID tags that has 1) resistance against owner tracing 2) with smallest additional cost per tag.

2 Related Works

Many studies on the prevention of owner tracing have been carried out. Some solutions reduce usability while others increase cost per tag by complex hardware for secure hash function.

Weis et al.[3] and Juels et al.[4] addressed the security problems of RFID systems and proposed some solutions for low-cost tags. One of their solutions (hash-lock) utilizes the one-way hash function. Due to the difficulty of implementing the hash function in small hardware logic (e.g. the SHA-1 hash function requires 20K to 50K gates), this solution entails a small but significant extra cost. In addition, this approach has a weakness in the unlocked state, that is, attackers can override the tag's authority in the unlocked state. Attackers may also turn their attention to the readers, because the readers may leak the secret if an attacker repeats a metaID sent by a locked tag. Although these attacks are detectable, it is also possible for denial-of-service attacks to be attempted against whole systems. Another solution (blocker tag) blocks the anti collision process to keep the tag ID be anonymous. However, the blocker tag itself has no intelligence and works indiscriminately. This may cause trouble as some blocker tag may block authorized readers.

A proposal from Ohkubo et al.[5] uses randomized hash chain to protect tag privacy. At ith read, a tag sends $a_i = G(s_i)$. At the same time, $s_{i+1} = H(s_i)$. Both H and G are different secure hash functions, and s_i is secret of the tag. Their argument describes forward security against tracing. Even if an attacker seized secret s_i of the tag,

the attacker cannot find past traces out of read records. With their proposal, a tag system can provide forward security. However, their approach also requires hash function implemented inside the tags.

3 Our Approach

As a step toward cheap RFID tags with privacy, we utilize a pre-calculated Bloom filter[6] in the protocol between RFID tags and readers. In this section we propose a tag-reader protocol called Kallima.

To prevent unauthorized readers from tracing a tag, communication between tags and readers must have no fixed id in plain form. In Kallima, the tag sends a Bloom filter with noise (NBF: noisy Bloom filter) and the reader tests its known IDs against the filter. We describe the protocol in the following order: bloom filter, components, tag identification, and anti collision.

3.1 Bloom Filter

A Bloom filter is a simple data structure that utilizes a set of secure hash functions and represents a data set S . We adopt it because it is resistant to one-way noise to make bits of value 0 to 1. With the resistance, a tag can disguise its identity in noise.

Generally speaking, one can test a Bloom filter of a set S if a data x is in S. The test yields a positive or negative result. A negative result means that x is definitely not in S. However, there is a risk of false-positives, which has a tradeoff relationship with space efficiency.

In Kallima, a Bloom filter is created from a tag ID x alone. A Bloom filter that represents $S = \{x\}$ is called SBF This Bloom filter is called a seeding Bloom filter (SBF). A SBF is created in the following steps.

First, a bit array B of length m is initialized to 0. Then, a set of k independent secure hash functions $h_1, \cdots, h_k$ with range $\{1, \cdots, m\}$ is applied to x. The hash value of x is assumed to be distributed uniformly between 1 and m. Finally, for each hash value v, the corresponding value of the bit array is set to 1. The result B is a SBF for x.

Receivers of a Bloom filter can find source tag ID of the filter if they know a tag ID y by testing y against the filter. Testing on a Bloom filter is performed in similar steps with Bloom filter creation steps. The same set of hash functions used in creation steps is applied to y. For each hash value v, the corresponding bit of the filter is checked. The test result is positive if all the checked bits are 1. The result is negative otherwise. In other words, tests with $B\&(1 << v) == 1$ for all $v = h_i(y)$ in $i = \{1 \cdots k\}$ yields a positive result.

A false positive occurs if all the hashed values of y are 1 by chance. In our case, noise can make some random bits to have value of 1. Figure 1 shows an example (each cube represents a bit and shadowed cubes are set to 1). In this case, $h_1(x) = h_2(y)$, $h_2(x) = h_1(y)$, and $x \neq y$. As a result, y seems to be in S. Actually it is not. The risk of a false positive for a Bloom filter is given as s^k, as s (saturation) is ratio of bits set to one among all bits in the filter. For example, saturation of filter `0b01101010` is 0.5. If $k = 3$, false positive rate is 0.125.

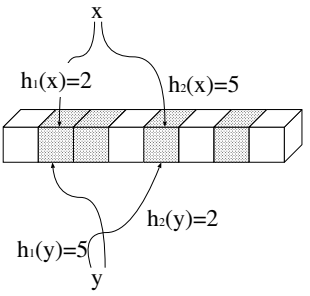

Fig. 1. An example of a false positive on a Bloom filter 0b01101010

3.2 Tags and Readers

Figure 2 shows an example of abstract block diagram of a tag. A tag's memory has two types of identifiers written in it. The tag ID and static Bloom filters calculated from the ID (SBFs: Source Bloom Filters). A SBF is calculated by standard computer at the time of tag production.

Note that a tag may have one or more SBFs with different parameters. We assume each SBF belongs to a *class* that defines the parameters for calculating the SBF. For example, a class A SBF may have $m = 512$, $k = 7$ and $H_n(d) = SHA1(d + n)$; a class B SBF may have $m = 1024$, $k = 13$,and the same $H_n(d)$ as in class A; and so on.

The simplest tag consists of a controller, a read-only memory for the ID, an RF module, and a one-way noise imposer. The one-way noise imposer is a method to disguise tag's identity. Other modules are the same as those in regular RFID tags without any security mechanism.

The one-way noise imposer has a simple function. It imposes one-way noise on the input bit streams. The one-way noise imposer turns 0 at a random place in the input bit stream into 1 until the bit stream saturation s comes to predefined value. We assume an expected saturation of the output bit stream as $s = 0.5$ for example.

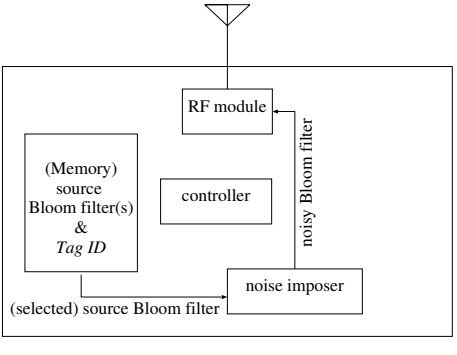

Fig. 2. Block diagram of a tag with Kallima

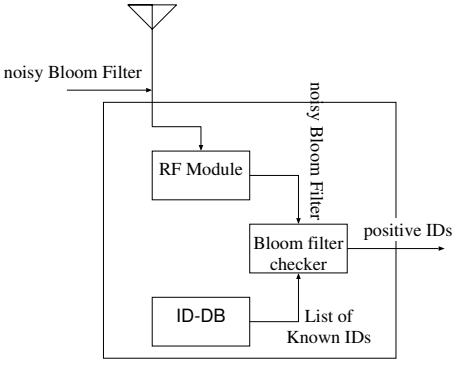

Fig. 3. Block diagram of a reader for Kallima

To turn an SBF of the tag into a bit stream looks random, the SBF is sent to the one-way noise imposer. The output is called a noisy Bloom filter (NBF). A Bloom filter does not become broken even if any additional bits are set to 1. A side effect of the added noise is an increased false positive rate ($(\frac{k}{m})^k$ becomes s^k).

Figure 3 shows an example of abstracted structure of a reader. A reader consists of an RF module, an ID-DB, and a Bloom filter checker. The ID-DB and the Bloom filter checker are the modules specially required for Kallima; the other modules are the same as those in ordinary RFID tag readers.

The ID-DB contains a list of tag IDs. In our model, a reader is an authorized reader for a tag if the tag's ID is in the reader's ID-DB. Otherwise, the reader is not authorized to identify the tag.

3.3 Identification of a Tag

Readers receive NBFs. They must test the filter to find tag IDs in their communication range.

The reader uses the Bloom filter checker to find candidate tag IDs. In the simplest implementation, it tries the known ID in the ID-DB one by one against the received NBF. The filter check process returns a set of candidate IDs or an empty set.

The process can produce three results that can lead for four implementation patterns.

- The process results in an empty set: there are no known tags
- Only one ID in the candidate set: the candidate ID as the detected tag (may perform validation process)
- Two or more IDs in the candidate set:
  - report entire candidate IDs
  - perform conflict resolution in some way

The conflict resolution process and validation process is not described in this paper.

The saturation of NBFs is controlled by one-way noise imposer, and the reader can estimate how false positives may occur. For example, if a reader knows 100 tag IDs, a 0.01% false positive rate corresponds to one expected read error among 100 trials.

If this risk is not negligible, the reader may request tags to send NBFs with increased m and k (i.e. NBFs of a higher class). If a reader knows 10^6 IDs and its application accepts only 1 misdetection for 10^6 reads, the situation requires an FPR of as low as 10^{-12}. This can be achieved with a Bloom filter with $k = 40$ if the filter saturation is 0.5 because $FPR = s^k = 0.5^{40} < 10^{-12}$.

An additional note is that the filter length m has a lower impact than in regular uses of Bloom filters as long as k is far lower than m. This is because the false positive rate is strongly bound with s and k, and the s of an NBF can be controlled by the one-way noise imposer. Discussions on m against k is described in section 4.1.

3.4 Anti Collision and Temporary ID

If the above protocol does not transmit any ID at all, application of ALOHA-style anti collision[7] is difficult. To avoid the problem, a tag should have a temporary ID that is valid for a period. Because tags have random number generators, it is easy to generate a randomized temporary ID at the beginning of communication. The temporary ID is held in a temporary memory such as an SRAM with capacitor, and is reused until the memory expires. Usually, ALOHA-style anti collision provide continuous energy until the process finishes.

4 Discussions

Here we discuss how our proposal achieve the goles outlined in section 1.2. We discuss the first goal in subsection 4.1 and the second goal in subsection 4.2 below.

4.1 Strength Against Owner Tracing

Compared to hash-lock and randomized hash chain, our approach has limited strength against owner tracing. In this section we describe when an attacker can trace tag and its owner.

An unauthorized reader for a tag is that do not know the tag's ID. In our model, the tag only sends NBFs and owner tracing occurs as an attacker reveals relation between one NBF and another.

If an attacker can take multiple NBFs from a tag, identification of an NBF becomes easy. First, an attacker collects two or more NBFs from the victim's tag. Then the attacker takes bitwise-and between all the NBFs. The bit stream produced by the operation contains less noise. The produced filter is called a guessed Bloom filter (GBF).

A well-produced GBF contains all the bits set to 1 in the SBF and some noise. Because k of the filter is a known number, the attacker can guess how much noise is left in the GBF and the certainty of each bits set to 1 in the GBF. Thus, with a GBF, the attacker can estimate if another anonymous NBF is related to the GBF. Figure 4 shows an example GBF constructed using four NBFs.

To make this kind of attack difficult, a tag should avoid to give multiple NBFs to attackers. In our assumptions, readers have no knowledge in advance to find a couple of NBFs out of many has come from a tag. If NBFs from a different tag are mixed in, half (actually, s) of the traces of the SBF of the victim's tag will be lost in the produced

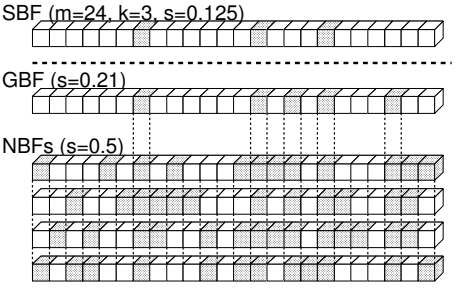

Fig. 4. An example of GBF

GBF. Hence, as long as the reader cannot isolate the victim's tag from the other tags, received series of signals are not usable to make a GBF.

To prevent attackers from obtaining multiple NBFs easily, a previously used NBF may be kept in a static memory with a reasonable lifetime. Because a tag needs a static memory to store the temporary ID for anti collision (described in subsection 3.4), the previously used NBF may be stored instead of temporary ID to make anti collision process.

Too long lifetimes, such as a day, help attackers to trace tags using the NBF or temporary ID, and too short lifetimes result in an increased chance for attackers to have successive access to an isolated tag. Expected period of time of tag isolation is longer if tags are placed or carried alone. If application expects many tags in an area the period of time can be short.

In addition to correlation of two NBFs, there are two ways that enables ID leakage. The first is leakage from an authorized tag reader, while the other is ID calculation from an anonymous NBF. Leakage from readers is beyond the scope of this paper because it includes various topics ranging from tamper-resistant hardware for readers to laws against intentional secret leakage.

The other mechanism is brute force attack. Our approach is not strong enough if an attacker deploys more computing power to track an ID.

An attacker with an NBF may try a brute force attack to find out the tag's ID. However, brute force does not help much in the situation because of the false-positives and the vast amount of tag ID data. For an NBF with $k = 40$ and $s = 0.5$, even a 64-bit long ID space can result in more than 16 million ($0.5^{40} \times 2^{64}$) IDs as candidate IDs.

It is clear that 16 million candidate IDs out of 64-bit long ID space is enough to identify a tag out of limited size of NBFs. If a newly received NBF is from the victim's tag, at least one of candidate IDs matches. If attacker receives an anonymous NBF, it may match with one or more candidates with as much possibility as $(1 - (1 - s^k)^{n_c})$ where n_c is number of candidate IDs. If $k = 40$, $s = 0.5$, and ID width is 64bits, only one NBF out of more than 65000 NBFs returns one or more candidate IDs as positive.

The attacker may use two or more NBFs from the tag to find small set of ID candidates. In that case, the risk of owner tracing will be described as follows.

Each NBF has possibility of false positive rate of s^k . With i NBFs from the same tag, false positive rate becomes $s^{m \cdot i}$. For two NBF with $k = 40$ and $s = 0.5$, one can find a 64-bit ID string that matches both NBFs with high probability ($0.5^{80} \times 2^{64} << 1$).

4.2 Tag Complexity

We believe tags for Kallima is simple enough for low cost manufacturing because our approach works without secure hash function and persistent memory on tags. As the result from previous discussions, a tag with Kallima should have a static memory as large as a maximum class of NBF, a capacitor to maintain the memory, and a one-way noise imposer in addition to the regular tag components.

For comparison, hash-lock requires secure hash function in addition to rewritable persistent memory to keep metaIDs. Randomized hash chain also requires hash function and rewritable persistent memory to keep ID secret.

5 Conclusions

In this paper, we discussed a solution for privacy issues on RFID systems of 5-cent tags. We can solve privacy issues on RFID systems as long as we can afford high price tag that uses complex hash function. However, we cannot afford it on low end system for now.

The challenge is how to hide tag identity without hash, and we propose combination of one-way noise with Bloom filter. As a solution, we propose Kallima, a tag-reader protocol that utilizes Bloom filter with noise as tag identifier. Because tags with Kallima does not need any secure hash functions and persistent rewritable memory on each tag, we expect Kallima can be as cheap as simplest tags used in the market.

References

1. Sarma, S., Brock, D.L., Ashton, K.: The networked physical world. Technical Report MIT-AUTOID-WH-001, MIT Auto-ID Center (2000)
2. Sarma, S.: Towards the 5c tag. Technical Report MIT-AUTOID-WH-006, MIT Auto-ID Center (2001)
3. Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and privacy aspects of low-cost radio frequency identification systems. In: Lecture Notes in Computer Science. Volume 2802. (2003)
4. Juels, A., Rivest, R.L., Szydlo, M.: The blocker tag: Selective blocking of rfid tags for consumer privacy. In: Proceedings of the 8th ACM Conference on Computer and Communications Security, ACM Press (2003) 103–111
5. Ohkubo, M., Suzuki, K., Kinoshita, S.: Cryptographic approach to a privacy friendly tag. In: RFID Privacy Workshop, MIT (2003)
6. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. In: Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing. (2002)
7. Finkenzeller, K.: RFID-Handbook, 2nd edition. Wiley & Sons, Ltd (2003)