Report of RFID Tag and Server Attack Detection Project

Xinxin Zhang, Hongting Wang

Introduction

Radio Frequency IDentifcation (RFID) tags are small electronic devices designed to store a quantity of information (such as the identity of the object the tag is attached to) and transmit that information when interrogated by a valid reader. RFID tags are being deployed for a diverse set of applications, ranging from inventory tracking, supermarket bar codes, identification and authentication, to biomedical use.

When RFID tags are used to store private information, security precautions are needed to prevent that information from falling into unauthorized hands. Unfortunately, tags have severe restrictions on computation and power consumption. As a result, many standard cryptographic operations are beyond the capabilities of tags. This makes design of security systems for RFID a unique and important challenge.

In this project, we study two aspects of RFID security and mutual authentication of RFID tags and RFID readers. In the first part of the project we operate authentication protocol analysis by simulating two common attractions: "Impersonate Reader Attack" and "Impersonate Tag Attack". we evaluate security of two RFID systems by analyzing traceability ad scalability of two proposed RFID systems: tree-based scheme and constant-time scheme.

Questions and Asnwers

1. The protocol makes use of cryptographic hash functions. Generallyspeaking, hash functions are difficult for an attacker to invert (i.e. find s given h(s)). Assuming that the hash function is secure, how is it that an attacker can still break the authentication protocol?

Because an attacker does not need to find the inverse of hash function (s) to break the protocol.
To impersonate tag, the attacker first queries the tag with a random string r1', and receives (M1', M2') from the tag. Then, the attacker uses random string r1 from the reader and sends (M1, M2) to the reader, where M1 = M1' ⊕ r1 ⊕ r1' , M2 = M2'. Since (M1', M2') is a valid reply to the query r1', M2 = M2' = f(M1' ⊕ t ⊕ r1') = f(M1 ⊕ r1 ⊕ r1' ⊕ t ⊕ r1') = f(M1 ⊕ t ⊕ r1) is a valid reply to the query r1 from the reader.
To impersonate server, the attacker first eavesdrops a valid session and record r1, M1, M2, M3, and blocks M3 from reaching the tag. Then, the attacker sends a random string r1' to the tag and receives (M1', M2'), so the attack can calculate M3' using [M3']L = [M1]R ⊕ [M3]L ⊕ [M1']R and [M3']R = [M1]L ⊕ [M3]R ⊕ [M1']L. Since r2 = M1 ⊕ t,

s = M3 ⊕ (r2 >> l/2)
  = [M3]L ⊕ [r2]R || [M3]R ⊕ [r2]L
  = [M3]L ⊕ [M1]R ⊕ [t]R || [M3]R ⊕ [M1]L ⊕ [t]L,

and s' = [M3']L ⊕ [M1']R ⊕ [t]R || [M3']R ⊕ [M1']L ⊕ [t]L
  = [M1]R ⊕ [M3]L ⊕ [M1']R ⊕ [M1']R ⊕ [t]R || [M1]L ⊕ [M3]R ⊕ [M1']L ⊕ [M1']L ⊕ [t]L

= [M3]L ⊕ [M1]R ⊕ [t]R || [M3]R ⊕ [M1]L ⊕ [t]L

= s,

s' is a valid reply to the tag. So without knowing s, the attacker can compute a correct M3 message through eavesdropped message during a valid session.

2. Out of the four properties defined above (tag identification, tag authentication, reader authentication, tag privacy), how does the protocol attempt to provide each one? Which properties are actually provided?

Tag identification: The protocol should provide a reader with a mechanism for determining precisely which tag it is talking to. The SM protocol attempts to provide a tag identifier t in tag T, and a mechanism that allows the reader to determine the tag identity. First the reader generates a random string r1, and then the tag generates a random string r2 and computes M1 = t ⊕ r2 and M2 = f(r1 ⊕ r2). The reader checks if M2 = ft(r1 ⊕ M1 ⊕ t) to identify and authenticate the tag T. This property is actually provided.

Tag authentication: The protocol should provide a reader with a cryptographic proof of a tag's identity. The SM protocol attempts to provide a tag identifier t in tag T, and asserts that only the valid server can extract t from the (M1, M2) pair sent by T (The authentication method is described above). But this property is not actually provided, since the attacker can impersonate a tag using the procedure described in problem 1 and thus allow communication with a valid reader without a valid tag identity.

Reader authentication: The tag should obtain cryptographic proof that it is communicating with a valid reader. The SM protocol attempts to provide a proof that a valid tag can only authenticate valid reader to update the secret information. After the reader authenticates tag T as valid tag, the reader computes r2 = M1 ⊕ t and M3 = s ⊕ (r2 >> l/2), then the tag checks if h(M3 ⊕ (r2 >> l/2)) = t to authenticate the reader. However, this property is not actually provided, because the attacker can impersonate a server using the procedure described in problem 1, and the tag will still authenticate the attacker as a valid reader.

Tag privacy: Private information about the tag, including the tag's identity, should not be revealed to any attacks (eavesdroppers). The SM protocol attempts to provide the tag information privacy and tag location privacy, as detailed below.

Tag Information Privacy: The detailed information Di of tag Ti is stored in the database of the server, which is assumed to be secure. A server and a reader communicate via a secure channel. Only a legitimate server can extract a tag identifier ti from the pair (M1, M2) sent by Ti. The server will send Di to a reader immediately after successful authentication of Ti. Thus, only an authorised server and a reader are able to access the information associated with a tag.

Tag Location Privacy: The tag only ever sends pairs (M1, M2), which cannot be linked to any particular tag. In other words, the eavesdropper can neither link tag responses to previous responses from the same tag, nor distinguish one tag's responses from another's. It is thus difficult to track the location of a tag.

However, the protocol fails to actually provide this property, because the values r1, M1, M2, M3 are easily revealed to eavesdroppers, and the attacker can impersonate the server to obtain tag identity and other secret information.

3. What assumptions are made about the computational capabilities of tags? Would the protocol of [1] be feasible on a tag that can only perform bitwise operations? Why or why not?

The tags need to be able to perform simple functions like left or right circular shifts and hash functions besides bitwise xor operations. The SM protocol is not feasible on tags that can only perform bitwise operations, because the tag need to compute M2 = f(t ‖ (r1 ⊕ r2)) and h(M3 ⊕ (r 2 >> l/2)) to allow authentication and update secret information.

3.2 Questions

After studying [3] and [4], complete the following:

1. In order to identify a tag, a valid reader must search the database based on the tag's response to the reader's initial query. In a system of 1,000,000 tags, how many searches are needed, on average, in each of the following schemes:

(a) The protocol analyzed in Part 1 (described in [1])?

Number of searches = 1,000,000, because the server needs to compare t with all tag identities in database, which is linear time search.

(b) The protocol described in [3]?

Number of searches = log(1,000,000) ≈ 20, because the server only needs to search along one branch, and the height of a tree is O(log n) when the number of tags is n.

(c) The protocol described in [4]?

Number of searches = 1, because when the reader receives a tag identifier Ψi,c, it goes to the table entry in M-I at address Ψn i,c, searches the table in M-II by the pointer p1 stored at that address, and searches the tag information in M-III by the pointer p2 stored at the address in M-II. While $E[k] = \sum_{k=0}^{\infty} k \cdot \Pr[k = k] = e^{-1} \sum_{k=0}^{\infty} \frac{1}{k!} = 1.$ means that expected size of the tables in M-II is 1, so the search time on average should be 1.

2. Simulate the tag compromise attack on the scheme proposed in [3]. As part of your simulation, you should generate a list of keys for N = 100 tags. Describe the order in which tags are compromised, and list the keys that are compromised by each tag compromise. Generate a plot of the probability that two tags can be differentiated from each other vs. the number of tags that have been

compromised. Describe the model of an attacker that you are using for your simulation; does your attacker compromise tags at random, or does (s)he chooses tags for compromise based on some criteria? Justify your assumptions. Compare your results to the theoretical results of [5], and explain any discrepancies.

The tags are assigned in a tree with the depth 3 and branching factor 10. The followings are lists of the keys that are compromised by each tag compromise:

Tag0 compromised keys:

'k20'
'k10'
'k00'

Tag1 compromised keys:

'k21'
'k10'
'k00'

Tag2 compromised keys:

'k22'
'k10'
'k00'

Tag3 compromised keys:

'k23'
'k10'
'k00'

Tag4 compromised keys:

'k24'
'k10'
'k00'

Tag5 compromised keys:

'k25'
'k10'
'k00'

Tag6 compromised keys:

'k26'
'k10'
'k00'

Tag7 compromised keys:

'k27'
'k10'
'k00'

Tag8 compromised keys:

'k28'
'k10'
'k00'

Tag9 compromised keys:

'k29'
'k10'
'k00'

Tag10 compromised keys:

'k210'
'k11'
'k00'

Tag11 compromised keys:

'k211'
'k11'
'k00'

Tag12 compromised keys:

'k212'
'k11'
'k00'

Tag13 compromised keys:

'k213'
'k11'
'k00'

Tag14 compromised keys:

'k214'
'k11'
'k00'

Tag15 compromised keys:

'k215'
'k11'
'k00'

Tag16 compromised keys:

'k216'
'k11'
'k00'

Tag17 compromised keys:

'k217'
'k11'
'k00'

Tag18 compromised keys:

'k218'
'k11'
'k00'

Tag19 compromised keys:

'k219'
'k11'
'k00'

Tag20 compromised keys:

'k220'
'k12'
'k00'

Tag21 compromised keys:

'k221'
'k12'
'k00'

Tag22 compromised keys:

'k222'
'k12'
'k00'

Tag23 compromised keys:

'k223'
'k12'
'k00'

Tag24 compromised keys:

'k224'
'k12'
'k00'

Tag25 compromised keys:

'k225'
'k12'
'k00'

Tag26 compromised keys:

'k226'
'k12'
'k00'

Tag27 compromised keys:

'k227'
'k12'
'k00'

Tag28 compromised keys:

'k228'
'k12'
'k00'

Tag29 compromised keys:

'k229'
'k12'
'k00'

Tag30 compromised keys:

'k230'
'k13'
'k00'

Tag31 compromised keys:

'k231'
'k13'
'k00'

Tag32 compromised keys:

'k232'
'k13'
'k00'

Tag33 compromised keys:

'k233'
'k13'
'k00'

Tag34 compromised keys:

'k234'
'k13'
'k00'

Tag35 compromised keys:

'k235'
'k13'
'k00'

Tag36 compromised keys:

'k236'
'k13'
'k00'

Tag37 compromised keys:

'k237'
'k13'
'k00'

Tag38 compromised keys:

'k238'
'k13'
'k00'

Tag39 compromised keys:

'k239'
'k13'
'k00'

Tag40 compromised keys:

'k240'
'k14'
'k00'

Tag41 compromised keys:

'k241'
'k14'
'k00'

Tag42 compromised keys:

'k242'
'k14'
'k00'

Tag43 compromised keys:

'k243'
'k14'
'k00'

Tag44 compromised keys:

'k244'
'k14'
'k00'

Tag45 compromised keys:

'k245'
'k14'
'k00'

Tag46 compromised keys:

'k246'
'k14'
'k00'

Tag47 compromised keys:

'k247'
'k14'
'k00'

Tag48 compromised keys:

'k248'
'k14'
'k00'

Tag49 compromised keys:

'k249'
'k14'
'k00'

Tag50 compromised keys:

'k250'
'k15'
'k00'

Tag51 compromised keys:

'k251'
'k15'
'k00'

Tag52 compromised keys:

'k252'
'k15'
'k00'

Tag53 compromised keys:

'k253'
'k15'
'k00'

Tag54 compromised keys:

'k254'
'k15'
'k00'

Tag55 compromised keys:

'k255'
'k15'
'k00'

Tag56 compromised keys:

'k256'
'k15'
'k00'

Tag57 compromised keys:

'k257'
'k15'
'k00'

Tag58 compromised keys:

'k258'
'k15'
'k00'

Tag59 compromised keys:

'k259'
'k15'
'k00'

Tag60 compromised keys:

'k260'
'k16'
'k00'

Tag61 compromised keys:

'k261'
'k16'
'k00'

Tag62 compromised keys:

'k262'
'k16'
'k00'

Tag63 compromised keys:

'k263'
'k16'
'k00'

Tag64 compromised keys:

'k264'
'k16'
'k00'

Tag65 compromised keys:

'k265'
'k16'
'k00'

Tag66 compromised keys:

'k266'
'k16'
'k00'

Tag67 compromised keys:

'k267'
'k16'
'k00'

Tag68 compromised keys:

'k268'
'k16'
'k00'

Tag69 compromised keys:

'k269'
'k16'
'k00'

Tag70 compromised keys:

'k270'
'k17'
'k00'

Tag71 compromised keys:

'k271'
'k17'
'k00'

Tag72 compromised keys:

'k272'
'k17'
'k00'

Tag73 compromised keys:

'k273'
'k17'
'k00'

Tag74 compromised keys:

'k274'
'k17'
'k00'

Tag75 compromised keys:

'k275'
'k17'
'k00'

Tag76 compromised keys:

'k276'
'k17'
'k00'

Tag77 compromised keys:

'k277'
'k17'
'k00'

Tag78 compromised keys:

'k278'
'k17'
'k00'

Tag79 compromised keys:

'k279'
'k17'
'k00'

Tag80 compromised keys:

'k280'
'k18'
'k00'

Tag81 compromised keys:

'k281'
'k18'
'k00'

Tag82 compromised keys:

'k282'
'k18'
'k00'

Tag83 compromised keys:

'k283'
'k18'
'k00'

Tag84 compromised keys:

'k284'
'k18'
'k00'

Tag85 compromised keys:

'k285'
'k18'
'k00'

Tag86 compromised keys:

'k286'
'k18'
'k00'

Tag87 compromised keys:

'k287'
'k18'
'k00'

Tag88 compromised keys:

```
'k288'
'k18'
'k00'
```

Tag89 compromised keys:

```
'k289'
'k18'
'k00'
```

Tag90 compromised keys:

```
'k290'
'k19'
'k00'
```

Tag91 compromised keys:

```
'k291'
'k19'
'k00'
```

Tag92 compromised keys:

```
'k292'
'k19'
'k00'
```

Tag93 compromised keys:

```
'k293'
'k19'
'k00'
```

Tag94 compromised keys:

```
'k294'
'k19'
'k00'
```

Tag95 compromised keys:

```
'k295'
'k19'
'k00'
```

Tag96 compromised keys:

```
'k296'
'k19'
'k00'
```

Tag97 compromised keys:

```
'k297'
'k19'
'k00'
```

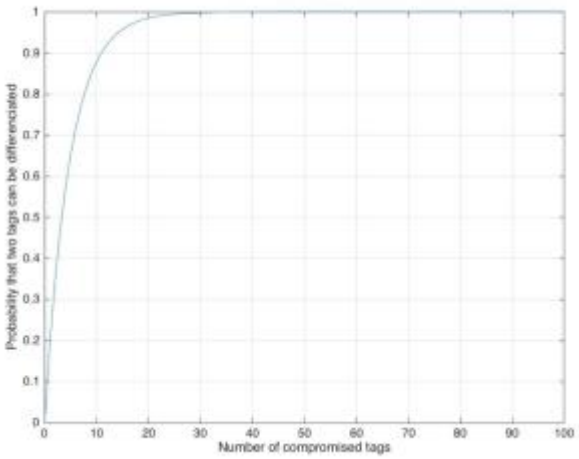
Tag98 compromised keys:

```
'k298'
'k19'
'k00'
```

Tag99 compromised keys:

```
'k299'
'k19'
'k00'
```

The following plot presents the probability that two tags can be differentiated from each other vs. the number of tags that have been compromised.



My model of attacker compromises the tags randomly. If the keys at depth 3 are different but on depth 2 are the same, I assume this is a successful attack, otherwise randomly choose another tag to get its key. My probability is about 86.6% when there are 100 tags with depth of tree as 3. However, from [5], the probability is between 95.4% and 99.1% under same condition. From my point of view, the discrepancy of the results might from the order of the tags that are chosen to be compromised.

3. How does the scheme of [4] defend against the tag compromise attack?

Each tag in the protocol in [4] has two pieces of secret information, a pseudonym and a key. Since tags' pseudonyms and keys are designed to be statistically independent for different tags, compromising some tags in the system does not affect the security of uncompromised tags. Although an adversary can compromise a tag in the system and attempt to obtain as many pseudonyms as possible by performing multiple protocol runs with a valid reader (calling Reveal oracle), the attacker's probability of having a advantage of distinguishing between two tags greater than zero is very small. Let N = number of distinct pseudonyms, q = the number of protocol runs an adversary performed using compromised tags, the probability is $1 - (N/(N - 1))^{2q}$.

4. What is the role of the constant C in the scheme of [4]?

The constant C represents maximum counter value C. It is used for reducing the probability that RFID tags can be traced by adversaries. The larger the value C is, the more hard it will be for adversaries to track the tag. However, the large the value C is, the larger size of the database will be, since the size is linearly related with the counter (the database size is O (NC)). Thus, C assures the tag's privacy but also increase system complexity.

5. In a system of 1, 000, 000 tags, estimate the storage overhead of each of the following schemes:

Assume N = 160, represents the number of tags;
(a) The protocol analyzed in Part 1 (described in [1])
Storage = 4*l*N+l = 4*160*1000000+160 = 640,000,160

(b) The protocol described in [3]
Storage = log(N) = log(1000000) = 19.9316 ≈ 20

(c) The protocol described in [4]
Storage = 1000000

Based on your answer to questions 1 and 4, comment on the scalability of these three schemes.

According to the answer above, the tree-based scheme described in [3] is the most efficient method to search the tags because it needs least storage and search times over three schemes.