RFID Systems: Protocols

Understanding the protocols that prevailing RFID systems use is important, both for understanding performance issues, and also as a way to see how some of the privacy and security problems can be solved.

In this section, we will look at protocols related to accessing tags, and at commands use to control how tags function.

We will also look at protocols related to NFC.

Protocol components

We have already looked at some aspects of protocols used in RFID systems.

The Medium: The way messages between tags and readers are exchanged.

- ✓ In the case of RFID, this is radio. Radiative and Inductive.

Message format: How messages made from symbols are formed.

- ✓ For RFID, these are formats like PIE and FM0.
- There are others formats as well, mostly to insure data is error free.

Medium access: How multiple tags share the medium.

- We will look at this, as this closely affects functionality and as we will see privacy *and* security as well.

Context and interpretation: How tags are commanded.

- We will also look at commands you can give tags, and how they follow them. This also can help with privacy issues.

Electronic Payment: NFC

NFC stands for *Near Field Communication*.

- At the radio layer, it is just passive inductive RFID. Not new.

- The differences and additions are at higher function levels.
- There are at least 3 major differences:

1. A reader can optionally emulate a tag. This means that two readers can in a peer to peer manner exchange data.
2. How data is formed and exchanged is formalized using *NFC Data Exchange Format* (NDEF). This standardizes data exchange.
3. An *element* for secure transactions. Electronic payment applications are driving this, although other secure applications can benefit.

NDEF Messages

- NDEF defines how messages are encapsulated in tags.
- Remember, there is no real limit on the amount of memory a tag can have.
- NDEF provides a scheme to take advantage of different memory architectures by breaking a message up into NDEF Records.
- The Type of data in a message is given by a Type Identifier.

These can be:

1. URIs
2. MIME data types
3. NFC specific types

Example: NDEF Messages

URI:

- Simple. The tag provides a URI, and the reader device uses it.
- http://www.kth.se (Go out and connect to this URL).
- sip:msmith@kth.se:5060:transport=udp;user=ip;method=INVITE…
  (Invite msmith to a SIP managed session)

MIME data types:

- Imagine a smart poster that sends:

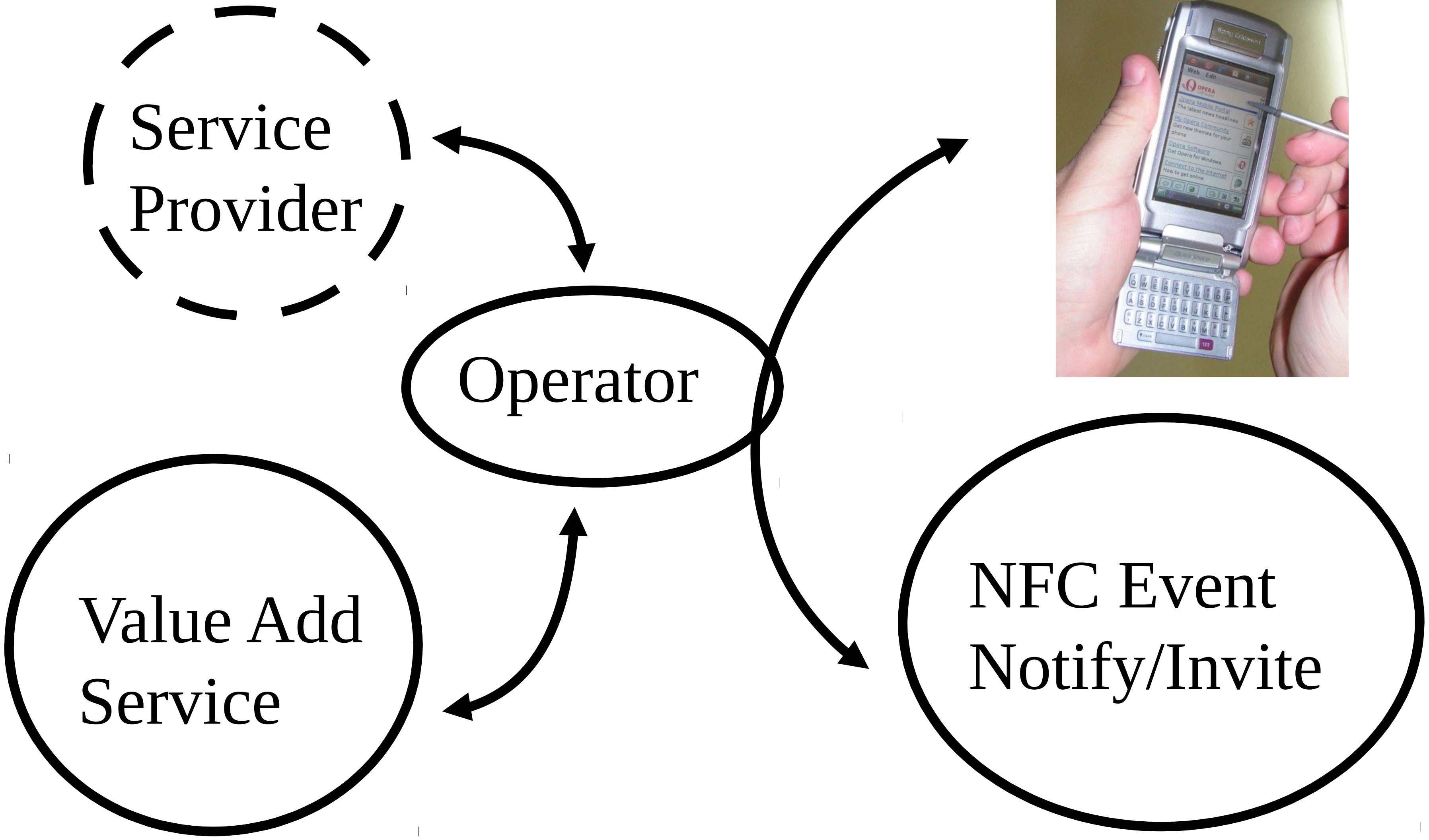
MIME-Version 1.0

Content-Type: image/jpeg

…

- The NFC device now knows exactly how to use the data.
- Over 90 MIME media types are supported.

NFC Transactions and Events



Could a model like this involve privacy problems or other threats?

Tag internal data organization

Tags are getting lower power, and more sophisticated. A typical modern tag architecture (EPC Gen2) with respect to data is shown below.

[See Figure 8.29 in the Textbook]

Most modern tags, including the ones we use in the lab, are similar.

EPC: Electronic Product Codes

- These product codes are standardized. They are not different between stores, only between products.
- They have different representations. Bar codes are most common.
- RFID is also a way to represent EPCs.
- To look up EPCs yourself, see the Global Electronic Party Information Registry (GEPIR):

http://gepir.gs1.org

Medium access and finding tags

We have seen in the lab that the RFID readers can find and identify several tags in range all at the same time. It can take *inventory* of tags in range.

- Seems mysterious. How can it sort them out? How does it control which tag has access to the medium (MAC)?
- Tag ID data is often 64 or 96 bits. It would take a looooong time to go through and check for each possible ID value.

- In the RFID systems we have, there are two approaches used.
- The TI Tag-IT systems (13.56 Mhz inductive tags) use a method called *binary tree walking* for MAC.
- The EPC Gen2 system uses a method called the *Q-protocol*. (This is different from the ‘Q’ used to describe radio circuits.)

Binary tree MAC

- Binary tree MAC works by going through each bit position in the 64 or 96 bit ID. It assumes that all tags have a pre-programmed, unique ID number.

1. Start at bit 0 of the ID. The reader asks all cards that can hear it what value they have for bit 0.
2. If it gets back nothing, there are no tags.
3. If it gets back only a zero, then it knows it only has tags with bit 0 equal to a zero. The reader assumes a similar result if it only gets back a 1.
4. If it gets back a collision, then it knows there are tags of both kinds. It then chooses a value for bit 0, and goes on to the next bit.

- By doing these steps for all bits in the ID field, it eventually can find all the tags, at least in theory.
- When it finds a tag ID, it sends a command to the tag with that ID telling it to not respond to inventory requests any more (until it is reset).

Example: Binary tree MAC

This example is simplified in that it assumes the ID has only 4 bits.

[See Figures 8.9 through 8.13 in the Textbook]

Problems with the binary tree method

Binary tree traversal is simple, and requires few reader and tag resources. But there are problems. Here are some of them.

- It can take a long time still! It has to go through 64 (or 96) bits!
- It won’t work if a tag does not already have a programmed ID.
- If a tag arrives after the start of the tree walk, then it may not be found. This is the *late arrivals* problem.
- If a lot of tags with the same ID bit value respond, the response can still look like a big jumble, and may be considered a collision.
- When a tag ID is successfully found, the reader will use the ID to send the tag a command to not respond further to the inventory.
- Unfortunately, that means that the ID is sent by the reader at relatively high power. Another radio can receive it, and know what tags are present without actually hearing the tags.
- That’s definitely a privacy concern.

The Q-protocol

The Q-protocol is newer, and is used by the standard EPC Gen-2.

- Q is a number between 0 and 15 (4 bits). The reader chooses Q based on a guess of how many tags there might be in the inventory.
- The guess, and the value of Q, can be refined later.
- The value of Q is then used to create 2^Q *slots*.
- When the reader starts an inventory, it commands all tags that can hear it the number of slots it has decided on.
- The tags then randomly choose a slot to be in. They also generate a 16 bit random number.
- Then, starting with slot 0, the reader asks if there are any tags in the slot. If there are none, or more than 1, the reader goes to the next slot.

The Q-protocol

- If a tag is the only one of the tag collection that has selected a slot, it will be the only one replying when its slot is reached.
- It replies with its 16 bit random number.
- The reader, if it hears it, will answer back with the same random number.
- When the tag hears its random number come back, it replies back to the reader with it’s EPC code (if it has one) and a new random number. It then marks itself has having been identified, and does not participate in the rest of the inventory.
- Note that the tag is really identified by the random number it generated. It doesn’t need to have a valid EPC or ID number at all.
- The new random number used by the tag and reader in this way to communicate is called a *handle*.

Refining the search.

When the last slot is tried, the reader can then do any of 5 things.

1. It can increase Q by 1 and do another pass through the slots.

   You would do this if there were few empty slots, and many slots with multiple colliding tags.

2. Leave the value of Q unchanged and do another pass through the slots.

   Do this if there is a balance between empty slots and ones with multiple tags.

3. Decrease the value of Q by 1 and do another pass through the slots.

   Do this if there are lots of empty slots and few with colliding tags.

4. Come up with a completely different value of Q and do another pass through the slots.

   Do this for the same reasons as #1 and #3 above, but the mis-balance is really big.

5. Stop. Do this because you found them all. No collisions.

Advantages of the Q-protocol

- It’s faster. You don’t go down long tree branches.
- It can be optimized. If you know ahead of time that you will have a small number of tags, you can choose a small Q. Same for if you know you will have large numbers of tags.
- It is more robust. Decisions are made based on the presence of just 1 tag. The jumble of all tags in a tree walk sending a 0 bit is avoided.
- It solves the late arrival problem. A tag appearing in the middle of an inventory will still participate and will be found.
- It is more secure. ID codes are not sent by the reader. Handles are used to communicate, and they are anonymous and disposable.

Commands

In the beginning, tags had no commands.

- Early RFID tags had to be very simple. They didn't listen.
- They would just power up, and send their ID over and over again.
- This was called *tag talks first*. Anti-theft tags are an example.
- In these early cases, the reader was simple and didn't talk at all.
- See section 8.2 in the textbook for interesting history.

- As applications and technology became more sophisticated, it became necessary and possible for the reader to control how tags function.
- This is done by having tags listen first for commands from the reader and then respond to them. The *reader talks first*.
- To work in this environment, a tag has to implement some sort of machine to be able to respond correctly to the commands.
- It still has to be simple, as we don't have a lot of power to burn.

Tag state machine

This one is used by the standard EPC Gen-2.

[See Figure 8.49 in the Textbook]

State meanings

- *Ready*: An idle state. Ready is entered after power up.
- *Arbitrate*: This is used during the Q-protocol to put the tag into a slot.
- *Reply*: This is used during the Q-protocol to reply to the reader when the tag's slot is reached.
- *Acknowledged*: This state is used when the tag is successfully seen (acknowledged) by the reader and now has a 'handle'.
- *Open*: This state is used to process a memory bank access request and looks at lock bits and/or passwords.
- *Secured*: This state is used when memory can be accessed because locks and passwords are satisfied. ("*Secured*" in English is another way to say you "have possession" of something.)
- *Killed*: This is a permanent state in which the tag no longer functions. Once in the killed state, always in the killed state.

Reader Commands

These are EPC Gen2 commands. They have been widely adopted in industry.

- Query: This informs the tags that a new inventory round is starting. It takes a lot of arguments, ie the value for Q, how data is modulated, etc.
- QueryRep: This commands a tag to reply if it is in the current slot.
- QueryAdjust: This tells the tag that the value of Q is changing, and to react appropriately.
- ACK: This command tells the tag that the reader got a valid 16 bit random number.
- NAK: This tells the tag that data from the tag was not correctly received.
- Req-RN: (Request Random Number) This command tells the tag to generate a new 16 bit random number for various uses, for example one that will become its handle or will protect further communication.

More Reader Commands

- Access: This command requests access to the tag’s memory. It takes arguments like bank number, passwords, handles, etc.
- Read: Read from any memory that is not locked. To successfully respond to a read command, the tag will have needed to perform an access command first.
- Write: Write to any memory that is not locked. Same thing about access command.
- Kill: This command puts the tag into the kill state. It usually requires a password.

- Future tags will probably have increased command sets as tag capabilities improve.

RFID privacy solutions

Again, many of these are controversial. The key, I think, is in *informed use*.

- Most of the solutions to RFID privacy problems revolve around the user being informed.
- By this, the users need to know if or when they are carrying tags.
- They also need to be given methods by which they can ‘opt out’ of having the tags read after they buy an item.
- In some cases, they may want to always be able to have a tag read, for example if it is attached to a part of their car.
- There is also the problem of trust. Although someone says they will or will not do something such as continue to read tags after an item has been bought, they may not be worthy of trust.
- Ultimately, the user has to decide what compromise is best.

Let the customer remove the tag

Just removing the tag is a simple, low cost solution. After all, customers usually remove price tags.

- For this to work, the tag has to be easy to identify and remove.
- One objection is that store returns are difficult, although for cheap items this may not be a significant problem.
- Another is that in some cases, the customer would want some tag information to be read, ie NFC tags on food.

- One solution is to just equip stores to make new tags.
- Another solution is to use two tags. One contains only information about the product, and the other contains the potentially private information.
- The one with private information can be removed.
- For expensive items, this might be practical.
- However, if the user is carrying a lot of tags, the private information could be inferred from the others.

Recoding

- Recoding means that after an item is bought, the store recodes the tag to contain no personal information.
- This is like removing the tag, except it is still there and contains only generic information like a EPC code.
- This may still be insecure, as a user may still have other tags they are carrying with personal information on them, and by inference the recoded tags are seen to belong to them.

- There is also the problem of vandals or thieves. If a store can recode the tag, then a thief can too. A thief might do this in order to change the price.
- Passwords help, but unless they are all the same it creates a big database problem with associated risks.
- Identical passwords create a security problem.

Aliases or pseudonyms

- In this case, a tag has many IDs. They are already in the tag.
- They also could be recoded by a ‘trusted’ reader.
- The ID a reader sees is dependent on time or number of reads. A reader will see one ID now, and a few minutes later or after having been read a certain number of times, it will return a different ID.
- The idea is that over that time, the user has moved, changed their belongings, or otherwise made it hard to correlate the different ID numbers to the same item or person.
- You could, in sophisticated tags, make this context dependant. Different context, different ID. This may be possible with future tags.

- You would need a huge database to keep track of all the different pseudonyms.
- You would also have to be able to trust the ‘trusted’ reader, if there were such a thing.

Blocker tags

- Remember the tree walking algorithm used to inventory some tags?
- What if you carried a tag with you that always sent out a collision? In other words, the environment looks like there are 2^{96} tags in it.
- It’s now impossible to finish the inventory as it would take thousands of years.
- This also works with the Q-protocol. The reader never sees a slot with only 1 tag in it.
- This is not jamming. A blocker tag is just replying like any other tag to an inventory request.

- Could be expensive to make such a tag unless they were made in large volumes. (Although, maybe you could make your own.)
- It blocks everything, including anything you might want to be seen. Ideally, there should be a way to turn them on and off.

Unblocker tags

- This tag works inside a trusted reader environment. It is the RFID equivalent of a browsers “Do Not Track” setting.
- The idea is that a reader is trusted NOT to read tags in range unless it sees the presence of a unblocker tag. It could have a special ID number, or a special Q-protocol slot that only it uses.
- The unblocker tag in effect gives the reader permission to read other tags in the same space.
- The idea behind this is that it give the user a kind of control. By having a unblocker tag, they can control when they give permission for their other tags to be read.

- For this to work, the user needs to trust the readers.
- If an entity is NOT trustworthy, why would they bother to set up a reader that checks for unblocker tags? They would just read as they want.
- Like blocker tags, it could be expensive unless they were made in large quantities. Again, could you build your own?

SNR discrimination

- The idea here is that a bad guy trying to read users’ IDs from a distance will put their RFID readers far away in an attempt to hide them.
- A tag will measure the signal power received from readers. If it sees a low SNR it can conclude that the reader is far away, and maybe is a reader owned by a bad guy.
- It also concludes that a strong signal is viewed as trustworthy because it is coming right out of a nearby, obvious reader that has no reason to hide that it is there.

- There are clearly lots of problems with this for both inductive and radiative RFID.

Shield the tag

- You know about skin depth and propagation of RF through objects.
- One approach that users can take is to shield the tag.
- Putting the tag into a metal container of the right thickness would work in most cases. The RF from the reader can’t pass though and get to the tag.

- This is practical for small tagged objects like ID cards and tokens.
- This clearly wouldn’t work very well for large objects like cars, or inappropriate items like clothes you are wearing now. (OK, it depends on the clothes.)

Tag killing

- You recall that ‘KILL’ is a tag command supported in some protocols.
- When a tag leaves the store, the KILL command is given, and the tag no longer will respond to any command of any kind.
- There is also partial KILLing. In this case only parts of the tag, presumably parts with information that have privacy risks associated with it, are killed.
- This is different from recoding in that once part or all of the tag is killed, it cannot be recovered or recoded. It’s functionally gone.

- This is good, as the average consumer may have difficulty locating and removing a tag.

Tag killing

- Functionally, tag KILLing is equivalent to removing the tag, and shares the same problems.
- It also has a trust problem. How do you know that the tag was really killed? How do you know it can never be restored?
- It is also possible to hack the cards and kill them all in the store. A vandal might want to do this. Note that the KILL command is usually protected by a password.
- Passwords help, but unless they are all the same it creates a big database problem and associated risks.
- If they are all the same, there is a password security problem.

Encryption of tag data

What about encryption techniques? Can't one just do that?

- You could encrypt ID and other tag data in advance, but now you have a big database problem. How do you distribute the database contents and still keep the database secure?
- The tag itself has such limited processing resources that it can't process strong encryption in any realistic time.
- Just because something is encrypted doesn't mean that it is secure. Keys and passwords can be sold. Databases hacked. 'Back-door' access can be secretly provided.
- Open source helps, but coding sophisticated encryption schemes properly is very hard. It's easy to make mistakes that defeat the purpose of the encryption.

- For lots of information on many topics in encryption and encryption policies, look for books and articles by Bruce Schneier.

Other issues

- Cheap readers can help. If users can somehow scan things themselves and see what is at risk, they maybe can make intelligent choices. There are trust issues here.
- Government policy will play a role in regulating what can be scanned, and what can be done with the data. There are trust issues here also.
- New RFID systems like NFC will bring new challenges and opportunities. NFC can run in a peer-to-peer manner. Every tag could also be a reader!
- Future tags will also carry sensors to collect context information.

Summary for RFID privacy solutions

1. Informing: Let the user know what is at risk and when. Knowledge is power.
2. Killing: Automatic disabling of tags
3. Recoding: Hiding tag meaning
4. Controlling: Letting the user choose when a tag works or not
5. Removing: The ultimate privacy assurance choice