Exercise 2: Consider credit cards with chips implemented for authentication and encryption, in which users use their personal identification number (PIN) (usually 4 digits) as a key to do transactions (see the example in Lecture 3).

(a) Give examples of confidentiality, integrity and availability associated with the systems.

Confidentiality can be achieved by several different methods, including encryption, access control, entity authentication, and physical security.

In this case, entity authentication is established through the challenge response protocol, where the chip-card reader delivers a challenge to the cardholder.

Integrity and authenticity can be achieved through different cryptographic methods, including message authentication codes (MAC), or a digital signature.

In this case, integrity and authenticity are established through the use of message authentication codes (MAC), which may involve the pre-sharing of a secret (authentication key). The cardholder subsequently generates a tag using the secret and the challenge.

Availability is achieved through ensuring timely and reliable access to and use of information.

Instead of pre-sharing a secret, the establishment of integrity and authenticity can be implemented through a digital signature, where the cardholder has both a secret key, as well as a certificate for their public key which has been registered in the public key domain. Therefore, when the cardholder generates a tag, any server (server 1 through server N) registered to the bank for performing tag verification is able to do so. The server redundancy ensures reliable access by eliminating a single point of failure – in this case the failure to perform tag verification if a server is down.

(b) Identify three different threats to each of the categories which results in a loss of security of the system.

Confidentiality can be threatened through an attacker's access to processing and computing capacity. Given the attacker has access to at least a pair of plaintext and ciphertext, they can perform a brute-force search to determine the encryption key by comparing the ciphertext generated using a random four-digit key against the actual ciphertext. The search space for a credit-card using a four-digit PIN is 10^4 which is roughly 2^{14}. This can be performed relatively quickly using a conventional computer. Even though secure encryption and decryption algorithms are used, using a four-digit PIN limits the search space that the attacker needs to explore in order to determine the key. AES encryption instead uses a 128-bit key, producing a search space of 2^{128}. Even with the most powerful conventional computing power, it could take more time to determine the key than the period of time that the universe has existed. However, if the attacker had access to resources such as a quantum computer, then it would only be a short matter of time before they are able to deduce public and private key pairs for any system.

Entity and authentication can be disrupted in the form of a man-in-the-middle attack, where the attacker can actively intercept messages transmitted through a wired or wireless link and modify them before forwarding the modified message to its destination. This can interfere with a cardholder's ability to verify their identity with the bank, or even the reader's ability to forward tag verification results to the cardholder.

Availability can be disrupted through jamming signals used to decrease the signal-to-noise (SNR) between the communicating entities. The communication channel can no longer ensure the reliable transfer of data.

(c) Identify three different methods that the attacker can use to reduce the search space for recovering the PIN and estimate the complexity for each of them (you may exploit some weakness in implementation, program vulnerabilities, application scenarios, ..., anything that can help you to reduce the complexity of the exhaustive search).

One such way to reduce the search space is through a dictionary attack, which requires that the attacker store as many possible plaintext and ciphertext mappings. Since it is not practical to store all such mappings, a dictionary attack involves guessing potential inputs based on provided ciphertexts. This is typically done provided a ciphertext, and guessing common combinations of data that a particular person may use in their password, such as their date of birth, etc. Therefore, the attacker may not need to test using every single permutation in the defined character set for the PIN.

Another way to reduce the search space is through a timing side-channel attack, by "measuring the time it takes the system to perform an encryption. With one encryption technique, the time to encrypt depends on the key, ... encryption time specifically depends on the size or the number of bits in the key. The time measurement helps attackers know the approximate key length, so they can narrow their search space accordingly." [page 212, PPM15]

Finally, we can suggest using a chosen plaintext attack such as differential cryptanalysis, which "seeks to find the difference between related plaintexts that are encrypted. The plaintexts may differ by a few bits. The attacker chooses the plaintext to be encrypted (but does not know the key) and then encrypts related plaintexts.

The cryptanalyst then uses statistical analysis to search for signs of non-randomness in the ciphertexts, zeroing in on areas where the plaintexts differ. Every bit of the related ciphertexts should have a 50/50 chance of flipping; the cryptanalyst searches for areas where this is not true. Any such underlying order is a clue to recover the key." [1] Provided these clues, the attacker can define the heuristics that can be used to deduce the remainder of the key through brute force, effectively reducing the search space.

Exercise 3: Consider the man-in-the-middle attacks in the following cases.

(a) Identify two consequences when some entries of a data base have been modified.

① Redirection

↳ MITM can intercept the request to add a backup email/number to an accont and set it to one of their own so when they request a password reset, the verification link is sent to the attacker's email/number ⟹ attacker can hijack accont now

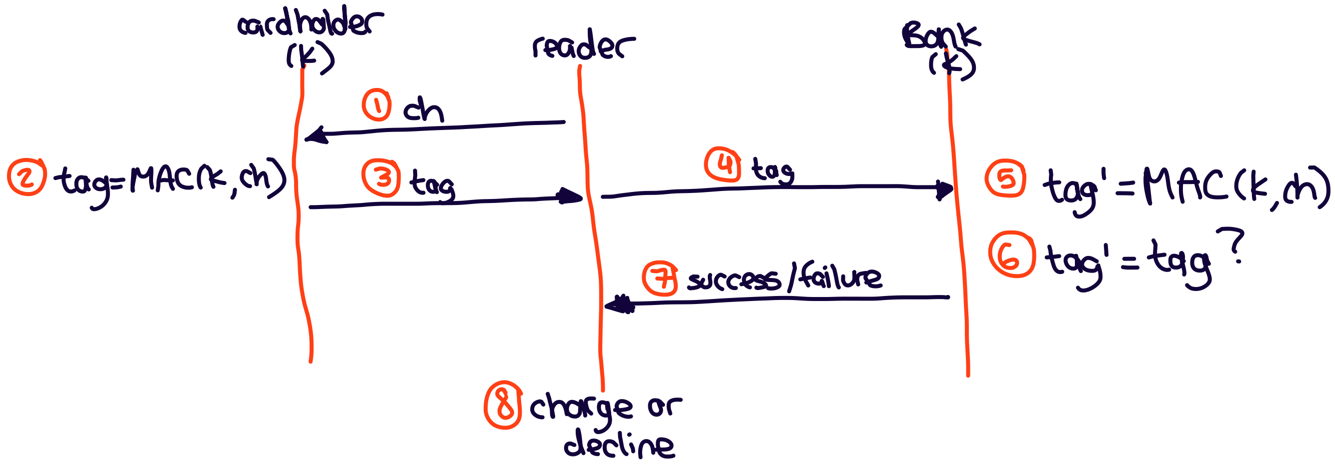
② Modify privileges

↳ attacker can intercept request to create an accont and modify the role of the client to expand their CRUD options within an application (i.e. grant system admin privileges to a less-privileged role).

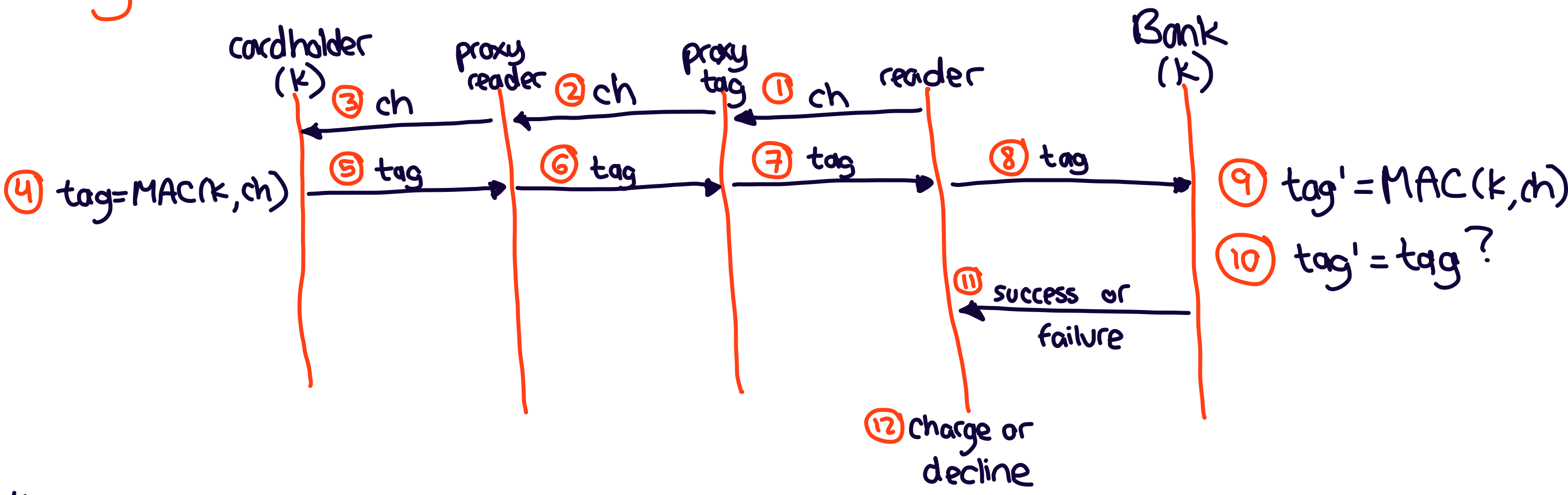
(b) Detail the entity authentication in the RFID system for contactless transaction, shown in the relay attack described in Lecture 3 (page 19).

Standard:

The reader authenticates the chipcard holder using the entity auth protocol (i.e. challenge response)



Relay attack:



in the relay attack case, the proxy tag can't compute the tag because it does not have the cardholder's key so it forwards the challenge from the reader to the proxy reader, which subsequently relays the challenge to the cardholder

(c) For (b), provide three countermeasures for the relay attack.

① RFID Blockers

↳ insulate card technology with metal (aluminum) sheets

② Timing analysis (tamper detection)

↳ measure latency of response times and monitor for discrepancies

③ Distance bounding protocols

↳ cardholder must convince reader that it is within range

Exercise 5: Program Vulnerabilities. For the following code, assume an attacker can control the value of basket passed into search_basket. The value of n is constrained to correctly reflect the number of dogs in the basket. The code includes several security vulnerabilities.

Some reminders:

- **snprintf(buf, len, fmt, ...) works like printf, but instead writes to buf, and wont write more than len - 1 characters. It terminates the characters written with a '0'.**
- **System runs the shell command given by its first argument.**

```
struct dog {
   char name [1024];
   int age;
};

/*
Searches through a basket of dogs of size at most 32.
Returns the number of puppies or -1 if there are no puppies .
*/

int search_basket (struct dog basket[], size_t n) {
   struct dog puppies [32];
   char puppy_names [1024], cmd [1024];
   int i, total_puppies = 0, names_size = 0;

   if (n > 32) return -1;
   for (i = 0; i <= n; i++) {
      if (basket[i].age < 12) {
                                                                    1024
         size_t len = strlen (basket[i].name);
         snprintf(puppy_names + names_size, len , "%s", basket[i].name);
         puppies[total_puppies] = basket[i];
         names_size += len;
         total_puppies += 1;         1024
      }
   }

   if (total_puppies > 5) {
      const char *fmt= " adopt - puppies -- num_puppies %d --names %s";
      snprintf(cmd, sizeof cmd, fmt, total_puppies, puppy_names);
      system(cmd);
   }
   return total_puppies;
}
```

(a) Circle three such vulnerabilities in the code and briefly explain each of the three.

Vulnerability I: dog age integer overflow – although this isn't a huge vulnerability, this demonstrates how the attacker can bypass the age check for each dog inside of the search_basket function by passing in a value that is greater than the max integer size, given that a signed integer is used to denote the dog's age. In this case, the integer overflow can cause the value contained in the age variable to ultimately be less than 12, giving the attacker access to write to the buffer within this block of code.

Vulnerability II: puppy_names buffer overflow (smash attack) – the name of each dog can be up to 1024 chars, while the buffer allocated to store each puppy name is also 1024 chars. Since there is no check to determine how much has already been written to the buffer, the buffer will overflow when the names_size counter has surpassed the size limit of the buffer. Consequently, the attacker can modify the contents of the stack beyond the buffer itself and manipulate variables such as the return address to intentionally point it towards planted malicious code.

Vulnerability III: command injection – puppy_names is a buffer that stores the string values provided to the name of each dog in the basket. It is added to the end of a formatted string that is passed into a system call – this can easily be exploited as it is possible to execute multiple commands in a single line. We can use an ampersand (&) in Windows, or a semicolon (;) in Linux between two commands in the same line to do exactly that. Not only that, but the actual command performed by this function can fail and the injected command will still be executed. The attacker can use this to perform a plethora of malicious attacks, as they effectively have access to the command line of the computer upon which this code is being executed.

(b) Describe how an attacker could exploit these vulnerabilities to obtain a shell.

See Vulnerability III from part (a). By doing this, the attacker can use something like the wget command on Linux to download and execute a script from a malicious source on the web, all in a single line.

References

1. Conrad, Eric. “Domain 5: Cryptography.” *CISSP Study Guide*, 2012, pp. 213–255., doi:10.1016/B978-1-59749-961-3.00006-6.