Programming Project 2

Course 01435: Practical Cryptanalysis
June 2013

Andrey Bogdanov
anbog@dtu.dk

1 Remarks

- hand in via campusnet before Thursday, June 27, 11:59 pm (no print out necessary)
- work alone or in groups of up to 4 people
- free choice of programming language
- the code should be well-structured and contain a sufficient amount of comments such that it can be understand be an external programmer
- some written documentation of the code is necessary and you have to be able to explain every single detail when ask
- it is your responsiblity to be able to demonstrate your program at the colloquium (make sure that a computer is available your programme runs on etc)

Programming Project (P)

The car protection system: We consider a remote control system for opening cars. It works as follows:

- The key fob has a 28-bit secret s embedded.
- When the button on the fob is pressed, it broadcasts a "Hello, key fob no. 0123456 is here!" message.
- If the matching car is within reach, it answers by broadcasting a random public 28-bit value u - the **challenge**.
- The fob uses the message digest algorithm MD5 to compute a response r:
  r `= lowest 28 bits of MD5(`$s||u$`)` (where '||' denotes the concatenation between s and u, i.e. MD5 receives a 56-bit input consisting of s and u written behind each other.)
  This response is again broadcasted and reaches the car.

- The car computes a corresponding value by using its own internal secret s^* to compute:
  r^* `= lowest 28 bits of MD5(`$s^*||u$`)`
- If $r = r^*$, then the two secrets s and s^* are probably the same, and the car unlocks the door.

In fact, this type of solution is widespread in modern car keys (only the key size is longer, if sometimes only slightly).

The attack: Our attacker is a person who tries to steal the car. He proceeds as follows:

- He picks an arbitrary, fixed challenge u and precomputes a Rainbow Table for the function
  $f(s) =$ `lowest 28 bit of MD5(`$s||u$`)`.
- When he gets access to a car key for a short time, he presses the button on the key fob. Then he acts "on behalf of the car" by sending a challenge u and receiving the response r.
- He now uses his Rainbow Table to find a value s with $f(s) = r$.
- He verifies this secret by sending some trial challenges to the key fob and checking whether he can compute the right response. If he can, then he has found the correct secret, meaning that he now can open the car at will.

The programming task: The goal of the programming project is to simulate this attack. For the precomputation phase, you can proceed as follows:

1. Start by implementing the function $f(s)$. For most programming languages, implementations of MD5 are available. For C and C++, you can download one from the internet. For Java, they are part of the security API (check out the class `MessageDigest`). Pick an arbitrary challenge u and implement the function $f(s)$ for later use.
2. Compute the Rainbow Table necessary for the attack. You may want to start with chains of length 2^{10} and a table size of 2^{18} rows, but you are welcome to change these parameters if it improves your table coverage[1]. Make sure to use a good random number generator to draw the starting values for the table, and not to use the same starting value twice (collisions **will** occur for such a large number of starting values). Functions $f_1, f_2, \ldots$ can be derived by setting $f_i(x) = (f(x) + i) \bmod 2^{28}$. Store the table in a suitable data structure. For C++, you can use the STL. For Java, check out the class `Hashtable`.

[1] The table coverage is the percentage of keys that is actually contained in the table.

3. Store the table in a file. For Java, you can make use of the fact that the class `Hashtable` is serializable.

The online phase should then be rather straightforward:

1. You assign the key fob a random secret s. Then you simulate step 2 of the attack by sending your pre-selected challenge u to the fob and receiving a response r.
2. Now your table is loaded from the file again.
3. You compute all possible successors of the known value r and check whether they are in the hash table.
4. If they are, you compute the predecessor to r (i.e., your candidate for s) and output it. Note that all of the following can happen:
   - No successor to r is in the table.
   - Even though a successor of r is in the table, r itself is not contained in the chain.
   - Even though r is in the chain, its predecessor is not the secret s we are looking for.
5. Output all candidates for the secret s that you find.
6. Test by additional challenge-response rounds whether one of your candidate values for s is the right one.

If you want to do more (not mandatory!):
You can continue to improve the programming project, e.g. by:

- Checking whether you can break larger keys (in fact, a few extra bits should be possible once the implementation is running).
- Trying out different parameters, and how they influence your success probability.
- Trying out how the performance is influenced if your table is on a hard disk or optical disk instead of lying in RAM.
- ...