Technische Universität München
Lehrstuhl Informatik VIII
Prof. Dr.-Ing. Georg Carle
Dr. Heiko Niedermayer
Dipl.-Inform. Ralph Holz

Network Security WS13/14
Assignment 2

Submission until **Wed, 11 December 2013, 23:00 UTC = 24:00 CET**.

Important directions:

- Submit by SVN. **Always** indicate your team: place a file `team.exercise02.txt` in **both** user directories under `exercise02/`. E.g. in `s_gu27jaf/exercise02/team.exercise02.txt` **and** in `s_guab38/exercise02/team.exercise02.txt`.
- **Always** show how you arrived at your answer to get full credits. I.e. show the computation, refer to the output, the man page, the RFC, etc. – whatever it takes to make your answer understandable.
- Be sure you do not violate our plagiarism guidelines
- Submit solutions for T-credits in a **PDF**. Submit **well-documented code**. We accept code in Python and Java. For details, see below.

Task 1 The X.509 PKI for the WWW (11 T-credits)

Recall that certificates are used to establish a cryptographically secure binding between an identity and the public key belonging to that entity: $Cert(B) = Sig_{CA}(B, Pubkey_B)$. In this task, we will have a closer look at the Public Key Infrastructure (PKI) as defined in the X.509 standard. Today, the most important use case for X.509 certificates is to secure HTTP connections with SSL/TLS. Here, certificates are issued to Web sites and browsers rely on them to establish secure connections. We will begin with a few theoretical questions and then turn to practically evaluating X.509 certificates in this task. The following RFC defines X.509: http://www.ietf.org/rfc/rfc5280.txt.

a) Figure 1 shows a simplified X.509 PKI. The Root CA does not issue certificates to end-entities itself. Rather, it delegates this task to Intermediate CAs.

1. Assume Alice has been issued a certificate. Let Alice be the leftmost leaf in Figure 1. Bob wishes to verify that a given public key belongs to Alice. He has obtained her certificate and trusts the root certificate. How does he proceed to check whether the certificate is valid (pay attention to how he finds the path!)? Concerning checks on certificate fields, you may restrict yourself to the 'vital' certificate fields (see slides). In particular, you may disregard anything that would be stored as an X509v3 Extension (see lecture slides). (1 T-credit)
2. When Bob goes through the steps, which entities must he implicitly 'trust' to issue correct certificates? (1 T-credit)

b) Let us turn to practice. **The following will only work reliably in the Virtual Machine we gave you. Please use this machine as there is no telling what other Linux distributions, *BSD, Windows or MacOS will do.** The main use of X.509 today is to allow SSL/TLS-secured connections to WWW servers. Let us have a look at the certificates of a few hosts. Download the following file first: `http://www.net.in.tum.de/fileadmin/TUM/teaching/netzsicherheit/ws1314/pub/certdata.crt` (you might have to use `wget` as Firefox likes to import `.crt` files right away). Then open a terminal and issue the following commands:

- `openssl s_client -connect twitter.com:443`
- `openssl s_client -connect twitter.com:443 -CAfile certdata.crt`

Now answer the following questions:

1. How do the two commands behave differently? (1 T-credit)
2. Thus, what does the `verify error` you get in the first command indicate? (1 T-credit)
3. Is there an *intermediate certificate*? If yes, what purpose does it serve? (1 T-credit)
4. Is there an *intermediate CA*, i.e. is there more than one organisation/company involved in the certification? Say why you think so. (1 T-credit)

c) Copy the part beginning with `---BEGIN CERTIFICATE---` up to `---END CERTIFICATE---` into a file `twitter.com.cert`. Then issue the following command: `openssl x509 -in twitter.com.cert -text`. The result is a text representation of the certificate content. Explain what is stored in an X.509 certificate (i.e. say what is stored in each field). You may skip all X.509v3 extensions except `CRL Distribution Points`, `Authority Information Access` and `Subject Alternative Name`. (1 T-credits)

d) Browse through the file `certdata.crt`.

1. How many certificates are in there? Give the command you have used to count. (1 T-credit)
2. Browse a bit more and try to find 2 Root CAs that you wouldn't have expected and/or find suspicious. Explain why you think they're unexpected or suspicious. (1 T-credit)

e) Now assume that a Root CA in the root store is hacked and under the control of an attacker, and this is not noticed by anyone for a few months. (Note: this question is intentionally a bit more open, we want you to think and discuss)

1. What further attacks can the attacker stage now? Sketch a possible attack setup. (1 T-credit)
2. In the attack you have described above, can browser users rely on CRLs or OCSP to protect them? Why? (1 T-credit)

Task 2 Implementing the Meet-in-the-middle attack on 2DES (20 P-credits)

The keylength of DES is too short for today's computers. A naive extension would be to encrypt twice with two different keys. Let p be the plain text, and c the cipher text. Let enc be the

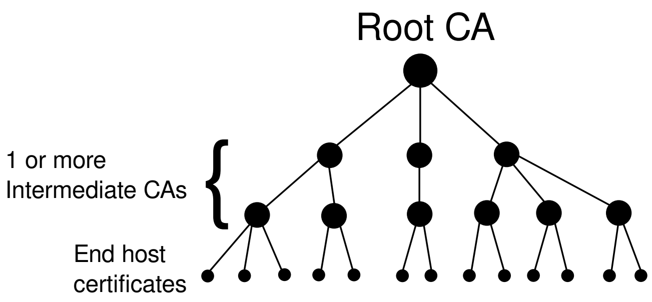

Figure 1: Ideal X.509 PKI with one Root CA.

encryption operation in DES, and *dec* the decryption operation[1]. 2DES is then: $c = enc(k_2, enc(k_1, p))$. Surprisingly, the strength of this encryption is not what one might expect. This is due to the following attack, called Meet-in-the-middle. This attack trades space vs. time. We are going to implement this attack, plus ask some theoretical questions.

Meet-in-the-middle The attacker needs to know one plaintext and one corresponding ciphertext (result of 2DES), and then does the following.

1. He computes and stores all $enc(k_i, p)$ for all possible k_i on the plaintext p.
2. He computes $dec(k_j, c)$ for all possible k_j on the ciphertext c.
3. He compares each $dec(k_j, c)$: if there is a match $enc(k_i, p) = dec(k_j, c)$, then $enc(k_j, enc(k_i, p)) = c$ must hold. He knows $k = (k_1, k_2) = (k_i, k_j)$.
4. If there is more than one match, he needs more plaintext-ciphertext pairs to eliminate the wrong candidates. This will not be needed in our implementation.

Key lengths To save you computational effort, our DES keys are stupid. In hex representation, a key is 16 ASCII characters long (64 bit), and we set most hex digits to 0. In `plain_cipher_pairs.txt`, you find the plaintext-ciphertext pairs that you need for the attack. The plaintext is always the same, but the ciphertext has been computed with different keys. In 3 cases, the effective DES key length was 3 hex digits (12 bit) – i.e. the first 13 hex digits of the DES key are 0, the rest is our secret. In one case, the effective DES key length is 4 hex digits (16 bit). Thus, the effective 2DES key length that you have to crack is severely reduced and it is more fun to work with.

General instructions

- You can find a framework to use on the lecture's home page. In `meetinthemiddle_skeleton.py`, you find the skeleton.
- If you use python, download the `pyDes` library from `http://twhiteman.netfirms.com/pyDES/pyDes-2.0.1.zip` and install it by executing *"sudo python setup.py install"* (see README). In `twodes.py`, you find how to use the `pyDes` library.
- We accept submissions in Python and Java. If you decide to use Java, make sure the DES implementation you choose gives exactly the same results. You can test this by using the (plaintext, expected ciphertext) entry in `doesmyDESwork.txt`. If you don't use a DES implementation that comes shipped with JVM 7, *make sure you attach the library to your submission.* In any case, make sure you keep exactly to the same output format as the Python code!

[1]Note that decryption is just using the keys from each DES round in reverse order.

- Make sure every *conceptual idea* in your code is well-documented. I.e. we do not require comments on lines like `a = b/7`. But we do want something like *'We now apply DES decryption. To this end, we first convert the hex string to bytes. The result is converted back to hex'*. I.e. tell us what you were thinking when you designed the logic of your program. Also tell us what you are doing if your code does non-obvious operations and/or uses non-obvious variable names (e.g. it is hard to guess what `compMFHX(plt, c, flag)` does).

a) Implement the attack (using our framework if you choose Python).

- There are four (plaintext, ciphertext) pairs that we give in `plain_cipher_pairs.txt`. Demonstrate your code works by finding the key pairs we used. There may be more than one key pair that works - it is OK to give the first one.
- Make sure that your input parameters follow exactly the input format of our skeleton code.
- Make sure to use the correct IV (see skeleton/twodes.py).
- Also make very sure that the last lines that you write to `stdout` follows the format in our skeleton code.
- Don't forget to rename your file names. We machine-test your code, e.g. we call `python meetinthemiddle.py plaintext ciphertext 3` or `java Meetinthemiddle plaintext ciphertext 3`. (15 P-credits).

b) We conclude with a few theoretical questions.

1. *Derive* the *average run time* of the algorithm as described above. Assume a keylength n. (2 P-credits)
2. Assume we had used full keylengths and a message length of 64 bit. What would be the *space requirement* in the *worst case*? Give your answer in GB, rounded to the next GB. Show how you arrived at this answer! (2 P-credits)
3. Explain: would this attack also work on AES – and why? (1 P-credit)