CSCI3381-Cryptography

Project 2: Cryptanalysis of Stream Ciphers Used in Practice

September 15, 2014

Stream ciphers permit very fast hardware implementations, and this has promoted their use in a number of widely-used standards for encrypting wirelesss network traffic, cell phone communication, and DVD copy protection. Serious weaknesses have been discovered in a number of these systems. In this project you will pick *one* of the three systems described below, and implement both the simple encryption algorithm and the attack. I have worked through the first of these, which is probably the most involved, in detail. The other two appear to be simpler, but I have not tried them. Full description of the attacks is provided in the referenced papers and websites.

1 Option 1. WEP Encryption

When you search for wireless networks to connect to, you'll find a few that are open (no encryption), many that are listed as protected by WPA encryption, and a shrinking number that are marked WEP. WEP stands for Wired Equivalent Privacy; the name is a boast that this system provides the same level of privacy as a hard-wired Ethernet connection. As we'll see, nothing can be farther from the truth!

The core of WEP is a stream cipher called RC4. RC4 is known to have some subtle weaknesses that are difficult to exploit in practice. The problem with WEP is not so much RC4 itself, but the way in which it is used.

See the Wikipedia page on RC4 or the research papers cited below for the simple details of how to implement the cipher. RC4 maintains a state consisting of 256 bytes, a permutation of the integers 0,..,255. There are two phases of the algorithm: The *key-scheduling algorithm* takes a key (between 5 and 16 bytes in general, but 8 bytes in our application) to initialize the state. The *pseudo-random*

number generation algorithm updates the state and emits a byte of output. These output bytes form the key stream.

In WEP, a 40-bit secret key is shared between the sending and receiving stations. For each packet sent, a 24-bit initialization vector (IV) is prepended to the secret key, and the 64-bit result is used as the input to the key-scheduling algorithm. The IV is also sent in the clear prepended to the encrypted data, so that the receiving station can reconstruct the 64-bit input to the key-scheduling algorithm.

Even with this brief description, some weaknesses are evident: the 40-bit key is a legacy of old US export restrictions. A brute-force attack, trying out all $2^{40} \approx 1$ trillion keys might take a couple of weeks on a personal computer, but only a few seconds with specialized hardware. WEP can also be used with 104-bit keys (resulting in 128-bit inputs to the key-scheduling algorithm) in which case such a brute-force attack is ruled out. More seriously, the 24-bit IV can take on only $2^{24} \approx 16$ million values, which means that after this many packets are sent, the stream cipher's keystream will repeat, which is bad news. Furthermore, in some implementations, the IV reset to 0 every time the router was powered up, which would cause the keystream to repeat.

An attack discovered by Fluhrer, Mantin and Shamir showed that if one could intercept enough packets with 3-byte IVs of the form $(b, 255, N)$, with $3 \leq b \leq 7$, then it is possible to recover the entire 40-bit key. If the IV's are generated randomly, this will require interception of about 2 million packets.

You can read about the details of the attack in the paper by Fluhrer, Mantin and Shamir (only Sections 7 and Appendix A are relevant). You might find it easier to read the account in Stubblefield, Ioannidis and Rubin (who actually deployed the attack against wireless networks).

Your job is to implement RC4 and the key recovery attack. The key recovery attack should take as input a large number of pairs consisting of a good IV and the first byte of the keystream generated using this IV. (The attack assumes that the first byte of the keystream is known, which turns out to be true in practice, since the first byte of plaintext in a packet has a fixed standard form.) You should perform experiments indicating how many packets with good IVs need to be supplied for the attack to work, and how many packets with randomly generated IVs need to be intercepted to find the requisite number of good IVs.

References

You can search for these on the Web; I also have copies.

The original account of the attack, only Sections 7 and Appendix A are relevant:

Fluhrer, Mantin and Shamir, 'Weaknesses in the Key Scheduling Algorithm of RC4'

An easier-to-read account of the core attack, together with a detailed discussion of how it was actually deployed to attack wireless networks:

Stubblefield, Ionnidia and Rubin, 'A Key Recovery Attack on the 802.11b Wired Equivalence Privacy Protocol'

I haven't read these, but they describe more efficient attacks. They concentrate on 104-bit keys, but it might be interesting to see what they give for 40-bit keys:

Tews, Weinmann and Pyshkin, 'Breaking 104-bit WEP in less than 60 seconds'

Beck and Tews, 'Practical Attacks against WEP and WPA'

2 Option 2: Content Scramble System

This is a method that was used for copy protection of DVDs. (Your report should include an explanation about how this encryption method prevents copying, since it is not at first clear what prevents someone from faithfully copying every byte of a protected DVD to a blank disk and some of the historical and legal background.) The encryption algorithm at the heart of the system consists of a pair of Linear Feedback Shift Registers whose outputs are combined in a nonlinear fashion.

One of the LFSRs is 17 bits long, and the other 25 bits. This makes 42 bits of state altogether. 40 of these bits are the secret key, two of the other state bits are hard-set to 1 to prevent the state from becoming 0. The outputs are combined in such a manner that if you know the stream of outputs from one of the registers plus the output of the keystream, you can determine the outputs of the other register. Since one of the LFSRs has only 17 bits, if you have just a few bytes of plaintext you can break the encryption as follows: use the plaintext to determine the first few bytes of keystream, guess the state of the 17-bit register, determine the first few bytes of output of the 25-bit register. Using the first 25 bits of this output stream, you can solve to find the state of the 25-bit register, and check if this configuration generates all the initial bytes of the keystream. Since there are only 2^{17} guesses to make, this attack does not require a lot of time.

You can find an awful lot about this notoriously insecure system just by Googling. I will be happy to provide you with sources I found useful.

3 ORYX

ORYX is a system for encrypting cellphone data communications. Once again, it relies on a nonlinear combination of several LFSRs.

I have not studied this attack in detail, so you're more on your own with this. However, the paper describing the attack is fairly brief, so it looks quite do-able. ('Cryptanalysis of ORYX' by D. Wagner, *et. al.,* at https://www.cs.berkeley.edu/ daw/papers/oryx-sac98.ps.) While the entire key for this cipher 96 bits, the attack succeeds by guessing only a crucial 16 bits of the state and requires 25 bytes of known plaintext.

4 Deliverables

These should include your code, along with a brief report on the problem, your approach, the results you obtained, and how to run your program. I will also ask you to demonstrate a working version.