Todd Louison
12/3/18
CSCI 4230

Cryptography Group 7

Todd Louison Writeup

Project Overview

Our project was an implementation of the SSL/SSH protocol to implement a peer-to-peer chatroom. We felt that this was an appropriate use case for this protocol because it is a classic, widespread use case for it in the real world. While it is currently more commonly used for cases such as securely accessing a remote shell or website security, the natural format of the protocol lends itself nicely to be demonstrated using back and forth message exchanging.

Our work was broken down for the three members of our group to work on separately: I focused on implementing SHA-1 and assisting with the key exchange logic, Shayne handled the socket programming and implementation of the key exchange/key encryption algorithms, and Milena handled bringing our ToyDES implementation up to speed to be a valuable public key cryptosystem.

Our delegation of roles played to our strengths in productive ways. SHA-1 turned out to be a larger task to implement than we first expected but I am a solid debugger and don't get frustrated at code easily, and that allowed me to fix and alleviate the problems we had. Shayne has more experience than both me and Milena, and this allowed him to act as our de facto project lead, delegating responsibilities and creating issues for us to tackle. Milena is a solid

team player and excellent coder, so we were able to delegate the expansion of ToyDES to her with confidence.

SHA-1 Implementation

My implementation of SHA-1 was based upon the specification found in the RFC Memo 3174. This algorithm is designed to take in an arbitrary length message m and hash it to a 160-bit value. The strengths of this algorithm are its large cascading effect, and it is computationally infeasible to brute force reverse the process. The cascading effect is powerful because it provides a significant change to the output with extremely high probability, while only changing one part of the original message. The computational infeasibility of breaking this hashing function has been invalidated unfortunately, as recent investigations into possible cryptanalysis attacks have shown methods to break SHA-1 in various ways.

To implement SHA-1, we followed the documentation in RFC Memo 3174. This was a memo sent out by Motorola and Cisco that detailed all of the implementation specifications. From this we implemented the padding, chunking, and computations described, and used the constants given. We chose to implement method 1 of the document, which meant using the line

```
w[i] = leftRotate( w[i-3] ^ w[i-8] ^ w[i-14] ^ w[i - 16], 1)
```

as opposed to the lines

```
s = t AND MASK;
if (t >= 16) W[s] = S^1(W[(s+13) AND MASK] XOR W[(s+8) AND MASK]
XOR W[(s+2) AND MASK] XOR W[s];
```

From this change, we lessened the runtime due to less computational complexity, but chose to slightly weaken the implementation in favor of simplicity due to the time restraint on our

project. Given more time, we would have certainly used the second method since runtime is not currently a large consideration.

There were multiple issues with implementing SHA-1 in our code. Due to our choice of working in python, working with bitwise operations was more difficult than other languages. We ended up deciding to do this implementation by using strings to represent binary integers, and by performing our operations on those. By doing it this way, it required a lot of converting between integers and strings to perform different operations on the bits, but overall allowed this implementation to be done in Python with less pain and confusing bitwise operations. There were many issues that stemmed from Python truncating our numbers upper end which required us to use strings as well, or risk losing information that is key to continuing the algorithm.

Message Authentication

In our project, like everybody else's, we implemented HMAC. HMAC is used to authenticate that the message we have sent and received are not only from the correct sender, but also that they have been untampered with in transit. Since we were required to use SHA-1 for our hash function in this project, we used it for our HMAC as well. This is a point of vulnerability in our code, as SHA-1 has been proven to be computationally broken, as previously mentioned. In an ideal system, HMAC is a secure way to verify the contents of the message and the sender of the message.

To perform HMAC, you run two rounds of hash computations. The first round consists of XORing your key with the i-pad value to get your I key pad, and with the o-pad value to get your

o key pad. You then concatenate the message to the end of your I key pad, hash it using SHA-1, then take that and perform the same operation with the o key pad and your result from the previous step. This provides a code that relies on the message, meaning if any bits are changed, the likelihood of the message not changing is near epsilon. This means that the recipient can tell if the message they are expecting is the one that arrived.

Using HMAC allows us to be sure about the security of many attacks, namely length extension attacks and chosen ciphertext attacks. This means that, if we were using a better hashing algorithm, the HMAC algorithm is an extremely secure concept, and provides high levels of certainty in cryptosystems.