CNT4406 – Network Security **Spring 2024**

Homework 3: Deadline Thursday 3/28

Instructor: Viet Tung Hoang

Typically a homework only carries 200 points, but this one has 270 points. That is, you have 70 bonus points.

1. (**Padding-oracle attacks**) (90 points) In this exercise, you are asked to implement the padding-oracle attack. The target server is `linprog2.cs.fsu.edu` (IP address is 128.186.120.158), and the port number is 31537. Due to some administrative restriction, to have a socket connection to this server, you need to run your code at the same machine. (You can log into this server using your linprog account.)

   The server maintains a secret string M which your program needs to recover. Currently, this string is (without the quote and case sensitive) "*C0ngr4tul4ti0n5!!! Y0u 5ur3 kn0w ur crypt0!!! :)*", and thus the byte length is 48. However, when I test your programs, I may change it to a different string, of a different length. You should *not* assume that the byte length of the secret is a multiple of 16.

   Interacting with the server. The server provides two types of requests. If you want an encryption of $P\|M$, for some chosen prefix P, then you need to send a request of the form "-e P". The server will send you a ciphertext C of $P\|M$ under the TLS 1.2 encryption on a random IV L. (This encryption scheme is theoretically secure, but if the implementation leaks the reason for decryption failure then you can mount the padding-oracle attack on it.) If you want to send a ciphertext core C with IV L for validation, you need to send a request of the form "-v C L", and the answer is "Valid", "Invalid: Wrong Pad", or "Invalid: Wrong Tag". Your goal is to recover the message M, and output it to `stdout`.

   For an example of how to interact with the server, see the Python script client.py in the course website. When I run `python3 client.py` in my terminal, the script waits for me to enter requests to the server. If I type `-e abcdef0123456789` (meaning my prefix is the **hexadecimal** string `abcdef0123456789`), I get back

   ```
   b‘Encryption:  80\n49 8d 8e 57 ...6a 3d de \n’
   IV: b‘a5244e79b9f94b4f5634a8b00e06e46c’
   -E abcdef0123456789
   ```

   Here the cipherext core C (in hex encoding) is `49 8d 8e 57 ...6a 3d de` that is 80-byte long, and the IV L (in hex encoding) is `a5244e79b9f94b4f5634a8b00e06e46c` If you want to encrypt with the prefix as the empty string, just use "`-E`". Likewise, if I type `-v aaaaaa 40beed9c1b5bfcbc997bc025a42b4c0a`, I get back `‘Invalid:  Wrong Pad’`. This means that after the server CBC-decrypts the ciphertext core `aaaaaa` with IV `40beed9c1b5bfcbc997bc025a42b4c0a`, it finds the padding incorrect.

   Another testing server. To help you to test your program on fragmentary secrets, I also set up another server at the same IP address, but the port is now 31536, and the secret is "*Hello World*".

   Deliverables. Upload to Canvas a zip file containing your source code, which includes a README.txt that informs me how I should run the program.

2. (**Android Keystore Attack**) (90 points) Let $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a good blockcipher. Let CBC$[E]$.Enc(K, M) denote the encryption of a message M under the CBC mode of blockcipher E under key K, and define CBC$[E]$.Dec(K, C) for decryption similarly. For simplicity, assume that here we only deal with full-block messages. Let $H : \{0,1\}^* \to \{0,1\}^n$ be a hash function.

A recent implementation in Android Keystore uses the following authenticated encryption scheme: to encrypt a message M under the key K, we output $\text{CBC}[E].\text{Enc}(K, H(M)\|M)$. For decryption, given the key K and ciphertext C, we first run $\text{CBC}[E].\text{Dec}(K, C)$ and parse the result as $T\|M$, where $|T| = n$. We then output M if $T = H(M)$, and output $\perp$ otherwise. In some sense, it's similar to the Encrypt-with-Redundancy paradigm that we studied and broke in class. However, the main difference here is that the redundancy $H(M)$ is put at the *beginning* of the message, instead of at the end.

Show that the Android encryption scheme is insecure by giving an authenticity attack.

3. (**S/key Identification Scheme**) (90 points) In the setting of one-time passwords, we have a client and a server that share a key, and the client has to prove his identity to the server. If an adversary can somehow steal the key from the server then all is lost. This is what happened to RSA's SecureID due to a hack at Lockheed Martin.

The S/Key identification scheme is an alternative to one-time passwords in which the client and server have *different* keys K_c and K_s, and the server key is *public*.

To set up the scheme, we generate a random string $L \leftarrow\!\!\$\ \{0,1\}^k$ (where k is sufficiently large, say $k \geq 128$). Let H be a cryptographic hash function (e.g., HMAC-SHA-256). Let H^n denote the n-th iteration of H, that is, $H^1(x) = H(x)$, and $H^i(x) = H(H^{i-1}(x))$ for every $i \geq 2$. Then, the client key is $K_c = (L, n)$, and the server key is $K_s = H^{n+1}(L)$ for some integer $n > 0$.

The scheme is then stateful (i.e., both the client and the server update K_s and K_c respectively). Concretely, upon running on input key $K_c = (L, i)$ for $1 \leq i \leq n$, the client sends $t = H^i(L)$ to the server, and sets $K_c = (L, i-1)$. If $i = 0$, the client does not do anything.

a) (30 points) Given a verifier key K_s and a token t, how should the server validate it? How should it update the key K_s?

b) (30 points) Explain informally why – assuming H is a good hash function – S/Key is a secure identification scheme against eavesdropping attackers.

c) (30 points) How would you choose the value n? Explain your choice.