# Graded Homework 2 – Due Thursday, March 22

1. In the handout on security models, it was shown that no stateless, deterministic encryption can be IND-CPA secure. For this problem, consider CBC mode in which the IV is picked in a way that can be predicted by the adversary (but may be stateful and/or non-deterministic).

   (a) Describe CBC mode using algorithm specifications (both encryption and decryption), where the IV is selected during encryption using a function named $\mathsf{getIV}(\lambda)$. You should use formulas similar to those on page 199, but give algorithm pseudocode with clear inputs (parameters) and outputs (return values), as well as any necessary looping and control flow instructions.

   (b) Assume the adversary has a $\mathsf{predictIV}(\lambda)$ function that always predicts the next IV that will be returned by $\mathsf{getIV}(\lambda)$ (and hence the next IV that will be used in an encryption). Define adversary algorithms $A_1^{\mathcal{E}}$ and $A_2^{\mathcal{E}}$ that can win the CPA game. (Hint: Think about what is fed into the block cipher as plaintext. Can you arrange it so that you can do multiple encryptions in which the same values are fed into the block cipher?) Analyze your algorithms to find the advantage of your adversary.

   (c) What if the $\mathsf{predictIV}$ algorithm isn't perfect? In other words, what if $\mathsf{predictIV}$ only predicts the correct IV with probability $p$? What is the advantage now?

   (d) CBC-Chain mode is a variant of CBC mode in which the encryption function keeps a record of the last ciphertext block produced, and uses that as the IV for the next encryption (so a series of CBC-Chain encryptions is the same as a single long CBC encryption). Show that CBC-Chain mode is not IND-CPA secure. (Note: This attack appears in the real world as the BEAST attack on SSL, discovered in 2011.)

2. It's clear that repeating a deterministic block cipher, as in ECB mode, is not secure. But what about repeating a secure (non-deterministic) cipher? In other words, let $E_K(P) \rightarrow C$ be an IND-CCA encryption scheme, and then define a two-block version of this by $E2_K(P_1, P_2) \rightarrow (C_1, C_2) = (E_K(P_1), E_K(P_2))$. It turns out that this construction is IND-CPA secure, but *not* IND-CCA secure. Prove the second part of that statement (in other words, give an adversary that wins the CCA game against the $E2$ two-block encryption scheme — like almost all CCA attacks, the trick is to disguise your decryption oracle requests so they don't exactly repeat the challenge ciphertext). Make sure you analyze the advantage of your adversary!

3. For this question, you are to do some experimentation with the Nextcloud end-to-end encryption feature. The directions below explain how to set up an environment where you can work with Nextcloud. Note that there are a lot of steps to get right, from setting up the server to working with Android Studio and the Android emulator. I have made the steps as simple as possible, but there are still a zillion ways this can go wrong — start *soon* so you can seek help if/when things don't work as expected. There is nothing to turn in for the setup steps (i1 to i5) — the questions you need to answer follow the instructions.

   *i1.* Follow the instructions in the "Working with Nextcloud" handout (given in class) to set up your own copy of the Nextcloud server, enable the end-to-end encryption app, and then create a regular user account for you to use.

   *i2.* Visit `https://developer.android.com/studio/index.html` to download and install Android Studio.

   *i3.* Follow the directions in the "Working with Nextcloud" handout to clone the Nextcloud android client into your Android Studio workspace. The first thing you should do is go into the `src/main/java/com/owncloud/android/MainApp.java` file, find the line that says

   ```
   OwnCloudClientManagerFactory.setUserAgent(getUserAgent());
   ```

   and change it to

   ```
   OwnCloudClientManagerFactory.setUserAgent(getNextcloudUserAgent());
   ```

   While you are cloning a dynamic, possibly changing version of the source code, currently this code is at line 133.

   *i4.* Create a phone emulator that you can use to run the Nextcloud client: In Android Studio, go under "Tools" and then "Android" to select "AVD Manager" ("AVD" is "Android Virtual Device"). Click the button that says "Create Virtual Device," select "Nexus 5X" and click "Next". The next screen asks what version of Android should be installed on your virtual device, and the best one I have found for this is Oreo — use the version that lists "x86" as the ABI, and "Android 8.0 (Google Play)" as the target, and you'll get a full install, including the Google Play store, for a more realistic environment. Click "Next" from this screen, and it will take you to the final screen, where you can change the "AVD Name" for this device — I like to have a dedicated device for Nextcloud testing, so I name it "NextcloudTest" to keep it separate from other projects. Click "Finish" and you're done. Note that this may have to download system images, so this may take a little while.

   *i5.* Now you are (finally!) ready to run the Nextcloud client, so go back and click the green "Run" triangle in Android Studio, and select the AVD you just created. The system will then build the client, boot the virtual phone, and run the program for you. You can log in and should see your files.

Once everything is set up and running properly, answer the following questions, which will give you insight into how Nextcloud stores data (both encrypted and unencrypted), and you'll also see where some key operations are located in the client code. The following questions assume you are running as a new user, and have not uploaded any files or used encryption yet. If you *have* already tried this, then log in to the Nextcloud web interface as administrator and create a new "clean" user account to use.

(a) In the Android client, create a new folder and select "Set as encrypted" in the actions menu. After that, click on the folder to navigate into the folder. Describe what you see (any changes in icons, messages that pop up, etc.) when you do this.

(b) Open the web browser on the emulated phone, and find an image on the web to use for the following tests. Upload the image twice: Once to your encrypted folder, and once to an unencrypted folder. To upload, simply do a "long click" on the image and select "Share image," followed by choosing to share via Nextcloud. After uploading, log in to the server and explore the "`data`" directory in your Nextcloud server installation. Locate the files you just uploaded, and report the locations, filenames, and sizes of each file. The encrypted version should be slightly larger than the unencrypted version — how much larger is it (report in bits as well as bytes)?

(c) Upload a second file to the encrypted folder, and report the file names you see on the server. Other than the file sizes, are there any clues as to the name or types of the files?

(d) All encrypted files have some associated "meta-data" that must be used in order to decrypt the files. Explore the `data` directory on the web server to locate this data (hint: it's not with the user's files). Report where you found the metadata, and cut-and-paste the contents into your homework submission (or print out on a separate page) – this is a JSON file, so you can use the "JSON pretty-printer," a program named `json_pp` on `csc580`, to make a more readable version.

(e) What fields are defined for each file? What do you think each one represents?

(f) Locate the part of the Android client source code that performs the encryption (hint: look in the `utils` directory for a file with an obvious-sounding name). Figure out and report the symmetric cipher, mode, and keylength used to encrypt files — when you report this information, identify specific lines in the code and copy the statements into your answer as a justification.

(g) Locate the place in the client code where it makes the decision on whether to upload the file encrypted or not. The easiest way to do this is to use Android Studio features to trace the calls back to the point where the decision is made — just go to the top of the method from the previous question, click on the method name to select it, right click and select "Find Usages." When you trace back to where the code decides whether to upload encrypted or not, it will be obvious. Describe what you found.