ECEN 5823 Spring 2025

Assignment 8: BLE Server with Security

Objective: Modify your Server implementation to use Numeric Pairing for additional security and MITM protection to guard against passive eavesdropping. Verify your implementation with a new characteristic with a value that represents the state of the PB0 button (push button 0) on the Server and displays as a characteristic in the Silicon Labs mobile phone app (more on this below in testing). This new characteristic shall require the Bluetooth link to be encrypted before its value can be transmitted.

Build a command table to show the commands you used to implement the A6 Server plus the commands you will use to perform pairing/bonding for this assignment.

Note: This assignment will start with the completed code from Assignment 7, configured as a Server.

Instructions:

1. Start by creating your assignment repository using the assignment link at https://classroom.github.com/a/kVR6YbBE. You will not need to clone this from the GitHub repo you just created into a new local directory since you will be starting with your code from the previous assignment. Instead, follow these instructions and run these commands with git bash within your assignment directory. Ensure you are setup to use SSH keys before proceeding, see the instructions from github at https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/ and be sure to use Git Bash For Windows or Cygwin if using a Windows environment. (Cygwin .ssh directory will be different than Git Bash.)

    a. On your local PC, duplicate the folder created in Assignment #7 to a new folder, name it something like ecen5823-assignment8-<username> where <username> is your GitHub username.
    b. Change your current directory to this new folder ecen5823-assignment8-<username> and execute the following git commands.
    c. git remote remove origin
    d. git remote add origin <url>
        i. Where <url> is the URL for the repository created with the link above
        ii. This adds the new submission repository created in the link above as your new origin.
    e. git push origin master
    f. Import ecen5823-assignment8-<username> into Simplicity Studio

i. Remember to perform a **Clean…** immediately after you import the project into Simplicity Studio.

2. In the questions folder, copy the file "Assignment 6 - Client Command Table for A7.xlsx" into a new file named "Server Command Table A8.xlsx".

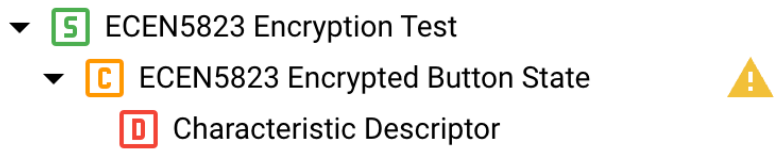
3. Design your program: Edit (delete events specific to Clients and responses) your Server Command Table A8.xlsx to include all of the events you will need to handle and commands you will need to add in order to implement the bluetooth "Numeric Comparison" pairing process for a Responder (Your Server) as described in:
   a. https://docs.silabs.com/bluetooth/latest/general/security/pairing-processes#numeric-comparison
   b. https://docs.silabs.com/bluetooth/latest/general/security/using-bluetooth-security-features-in-silicon-labs-bluetooth-sdk

   using Push Button PB0 as the confirmation input (Yes button) on the Server. Plan ahead, we'll use PB0 for the confirmation on the Client in the next assignment as well. Please note, we are using PB0 for two different functions in our Server:
   - PB0 presses and releases update our local GATT database, and send indications when enabled, and
   - PB0 presses are used during the pairing/bonding process as the "Yes" button.

   c. The "Responder" will be the Server.
   d. The "Initiator" will be the Client (the EFR Connect phone app).
   e. The Server will use the **DisplayYesNo** IO capabilities option to display the passcode on the LCD for verification, and use PB0 as the confirmation (the Yes button).
      i. When the passkey is being confirmed, the device should display "Passkey XXXXXX" on the DISPLAY_ROW_PASSKEY line and "Confirm with PB0" on the DISPLAY_ROW_ACTION line.
      ii. When the push button is pressed in this scenario, the device should complete the pairing process and clear the DISPLAY_ROW_PASSKEY and DISPLAY_ROW_ACTION lines. Use interrupts for PB0. Make sure to use external signals to prevent calling Gecko API commands from your interrupt service routines.
   f. Use sl_bt_sm_delete_bondings() on boot or connection closed events to prove you can complete the bonding procedure successfully. This ensures we don't save bonding information in the Gecko or mobile phone, and forces a re-pair each time we connect. You wouldn't do this on a real-world production design, but this allows us to easily test reconnection scenarios.
      i. **Important**: See note below on iOS Bluetooth pairing and bonding.
   g. When pairing completes, the LCD should show "Bonded" on the DISPLAY_ROW_CONNECTION.
   h. Extra credit (+10 points) is available if you include a bubble state diagram of your temperature taking state machine in your command table spreadsheet for A8.

4. Add a custom encrypted characteristic which cannot be read by the Client or sent as an Indication until the link is bonded, using the Bluetooth GATT Configurator, using the "Add new item" button to add the new service, and the new characteristic.
    a. Use service name ECEN5823 Encryption Test and UUID value 00000001-38c8-433e-87ec-652a2d136289. Advertise the service.
    b. Use characteristic name ECEN5823 Encrypted Button State and ID value button_state with UUID value 00000002-38c8-433e-87ec-652a2d136289.
        i. Set value length to 1 byte (check box checked).
        ii. Set the properties to include Read with Bonded, and Indicate with Bonded.
    c. Save the .btconf file. This should trigger the generator to run, creating new gatt_db.c/.h files (these files are the source of your GATT database).

The new service and characteristic hierarchy should look like this, see the Appendix at the end:



5. Custom encrypted characteristic updates:
    a. Indications should be supported (enabled, disabled, and sent when enabled. **Reminder**: Indications for this new characteristic are completely independent from the Health Thermometer Characteristic (HTM) indications - indications for each characteristic in a GATT database is independently enabled/disabled).
    b. The characteristic value in the GATT database should be set to 0x01 when the button is pressed
    c. The characteristic value in the GATT database should be set to 0x00 when the button is released.
    d. The conditions under which a button_state indication is sent shall be: connection open AND button_state indications are enabled AND bonded.

6. LED0 shall reflect the state of HTM indications (off = disabled, on = enabled), and LED1 shall reflect the state of custom button indications (off = disabled, on = enabled).
7. Since we have button_state indications being sent asynchronously with respect to our temperature taking state machine, you will need to build a circular queuing mechanism to:
    a. Save indications in a queue that cannot be sent at the time the event occurs because:
        i. There is an indication currently in-flight, or
        ii. If there is no indication in-flight, but your queue has entries that haven't been sent yet. This maintains the temporal order of indications. This is a requirement for this assignment.

   b. Then periodically check, in an energy efficient way, to see if there are unsent indications in your queue and send the saved indication(s) after receiving an indication confirmation event.
8. Add displayPrintf() to display A8 on row DISPLAY_ROW_ASSIGNMENT
9. For the code you submit for grading, there should be no calls to __WFI() or __BKPT() or LOG_***() functions. No calls to Systick routines are allowed. However, do log errors returned from API calls.
10. Test your code using the "Testing" instructions below and verify your program functions correctly.

Testing:

Use the "**EFR Connect**" app to test your code.

Your firmware should pass the following tests:

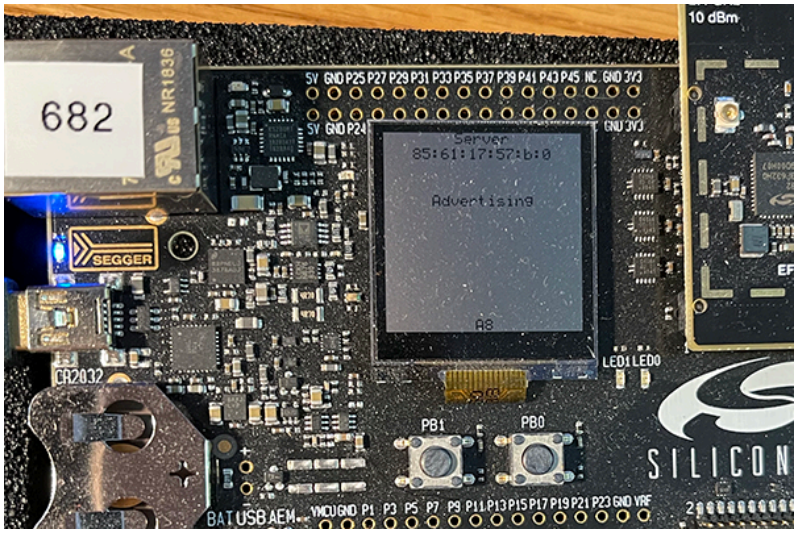
1. All previous behavior from Assignment 6 should be supported regarding the HTM characteristic indications and those tests should continue to pass as written.
2. When selecting the device in the "Bluetooth Browser" a service with UUID 00000001-38c8-433e-87ec-652a2d136289 should be listed.
3. This service should include a characteristic with UUID 00000002-38c8-433e-87ec-652a2d136289.
4. Indications for the button state characteristic should function identically as the Health Thermometer characteristic with the following addition. The button characteristic value represents "something secret" that we don't want a man in the middle to be able to observe. Therefore bonding is required in order for the Server to send button state characteristic indications to the Client (to send button state indications, the link must be bonded and the button characteristic indication must be enabled). Similarly, the link must be bonded for reads of the button characteristic value. Validate that indications for each characteristic are independently controllable, and button state indications can be sent once the Server is paired to the Client.
5. The state of PB0 should be continuously displayed on the LCD regardless of the connection state or bonding state. When PB0 is pressed LCD Row 9 should display "Button Pressed" and when PB0 is released LCD Row 9 should display "Button Released".
6. When expanding this attribute and clicking "Read" in the phone app for the first time, a pairing request (sequence) shall start. PB0 shall be used as the Yes button to confirm the pass key value.
   a. The Server LCD should show the pairing content and confirmation message.
   b. The value shown in the Pairing dialog on the mobile phone application should match the value on the Device LCD.
   c. After completing the pairing process on both devices it should be possible to click the "Read" button on the mobile phone application to read the button state characteristic value on the Server. The value of the attribute shown in the phone app should reflect the pressed state of the button when you tap on the "Read" button in the phone app.

7. When indications for the HTM characteristic are enabled set LED0 to on, and set LED0 to off when disabled. When indications for the button state characteristic are enabled set LED1 to on, and set LED1 to off when disabled.
8. When disconnected from the BlueGecko application, the display should show "Advertising" once again.
9. It should not be necessary to reset your device to reconnect with the Blue Gecko or from the mobile application.

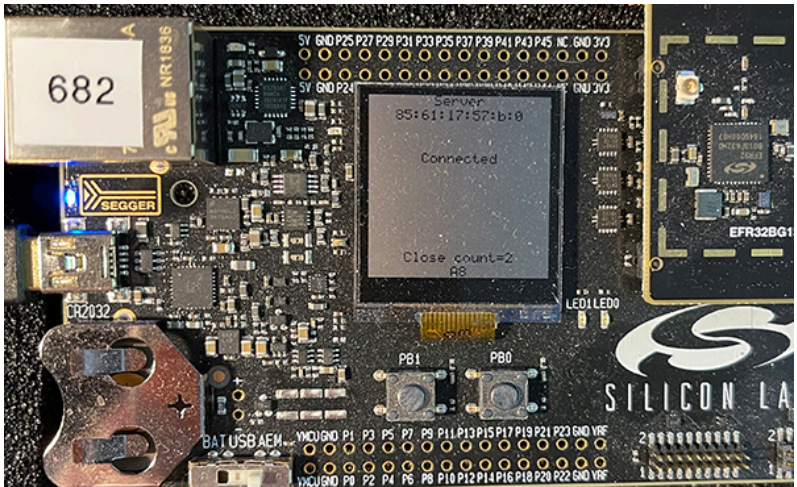
Note: iOS 9.1+ forces a device that has completed the pairing process into the bonded state. This means that the keys that were exchanged are stored on your iPhone. After successful pairing (bonding) you have to go into Settings / Bluetooth, select your Gecko device and tell your iPhone to forget the device. This assumes that you call sl_bt_sm_delete_bondings() on a boot or close event in your server Gecko code. If you don't forget the device on your iPhone, when you attempt to connect a second time, the connection will be rejected by the iPhone app because your server code deleted the pairing information as a result of calling sl_bt_sm_delete_bondings().

Here are pictures of the Gecko and phone app proceeding through the pairing/bonding process:

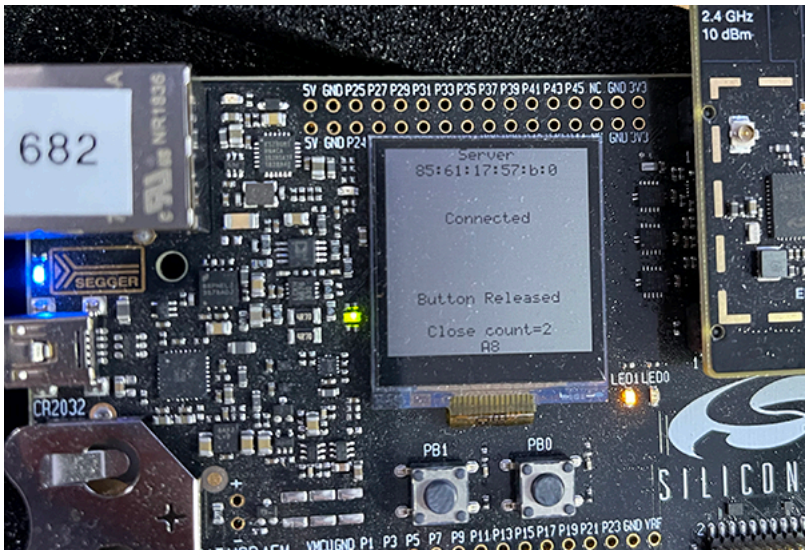
Not in a connection, advertising:



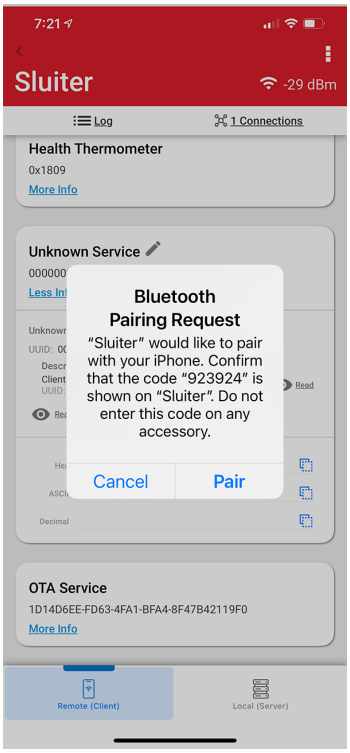
Connected to the phone app:



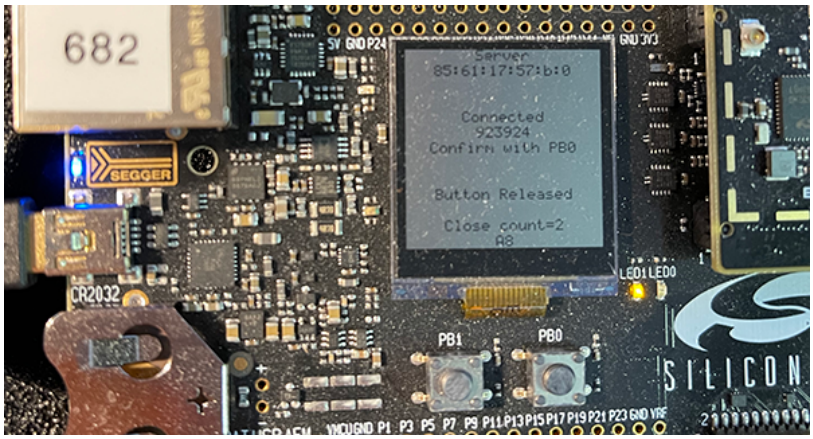
Phone app **not receiving** button_state indications from the Gecko when button_state indications have been enabled (LED1 on) but before bonding, when PB0 is pressed and released on the Server, button_state is not displayed on the phone app:



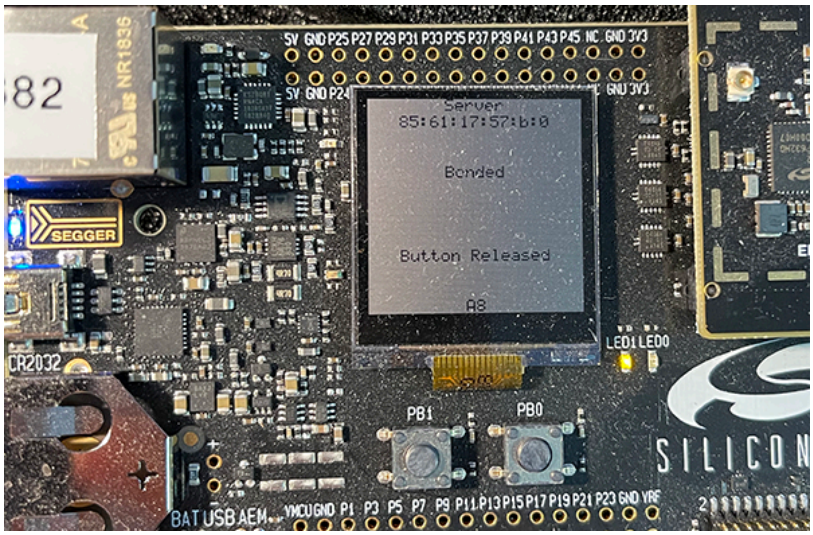
Tapping “Read” characteristic in the phone app and resulting prompt to pair:



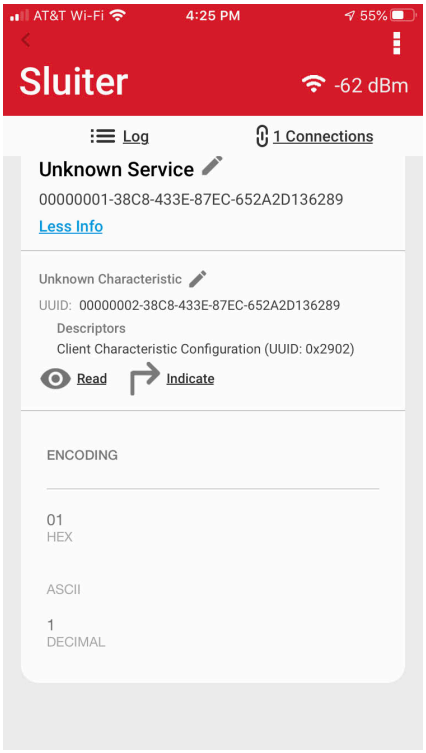
Corresponding pairing state on the Gecko:



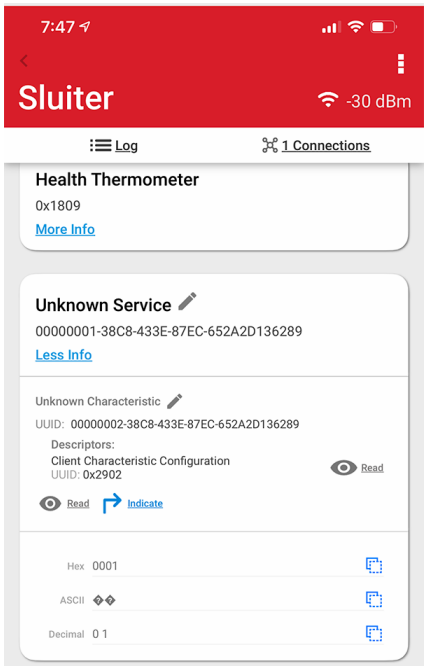
After pressing PB0 to confirm pairing:

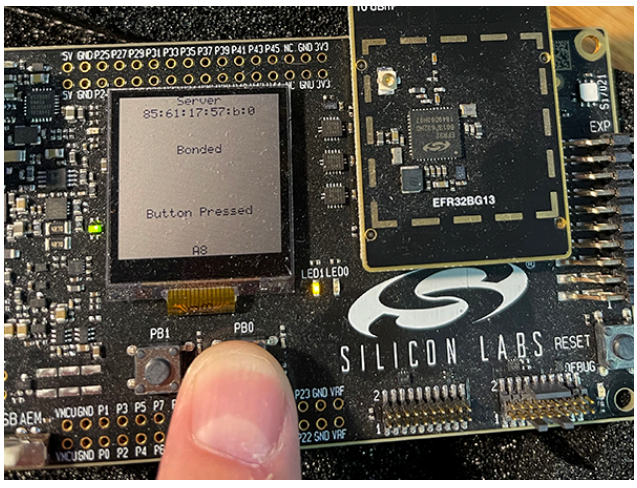


Tapping on “Read” in the phone app when bonded and button_state indications are disabled while pressing PB0 on the Server:

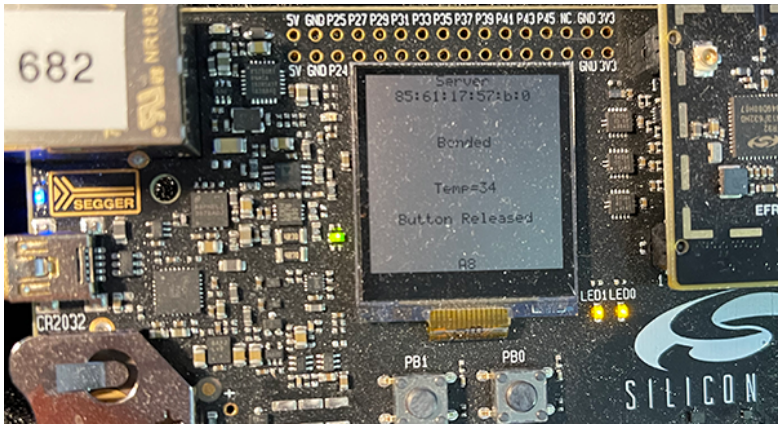


Receiving button_state indications when bonded and button_state indications are enabled (LED1 on) while pressing PB0 on the Server:





Button_state and HTM indications both enabled (LED1 and LED0 on):



Deliverables:

- Your “Server Command Table A8.xlsx”, saved in the “questions” folder of your submission repository. The content should match your A8 Server implementation. Use the git add command to add this new file to your repository.
- The completed assignment 8 program project to be run on the instructing team’s computer for grading.
- Submission via **github classroom** and your repository setup via the classroom link at above. Use the following git command to submit your assignment: git push origin master
- Verify your ecen5823-assignment8-<username> repository contains your latest/ intended code and files in the questions folder. It is your responsibility to push the correct version of your project/code. Use a web browser to verify the correct version of your project/code has been uploaded to GitHub.
- In **Canvas**, submit the URL to your github repository and in **GitHub** create a tag of your GitHub submission prior to the due date with the text of A8-final. The date and time of the tag creation is what we use to determine whether your submission was on time or late. These 2 steps will complete your assignment submission.

Approach/Guidance

1- Add the new service, characteristic and CCCD in the Simplicity Studio GATT Configurator. Generate new GATT files. This will create new gatt_db.c/.h files.

2- Configure GPIO for PB0. Set it for input with glitch input filtering enabled (You know how those pesky push buttons like to bounce - and if you don't know, please ask me!). Use:

```
GPIO_ExtIntConfig (PB0_port, PB0_pin, PB0_pin, true, true, true);
```

You'll need to figure out the port and pin numbers for PB0.

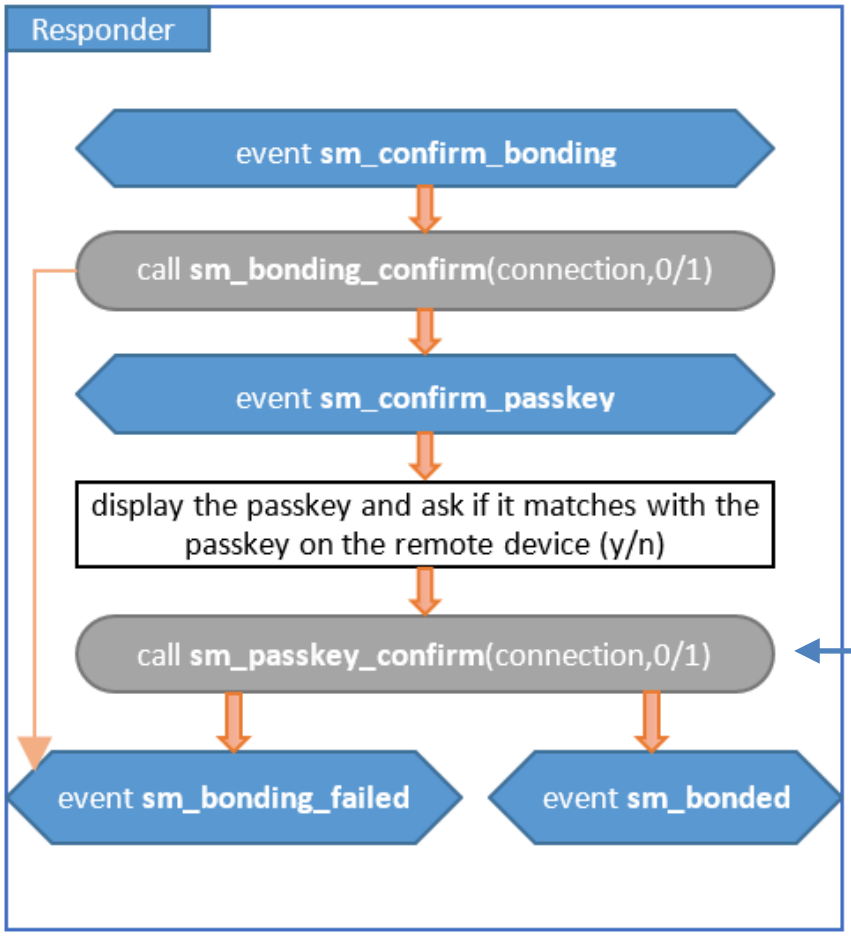
3- Develop the code for the ISR GPIO_EVEN_IRQHandler(). Add scheduler "SetEventXXX()" functions that in turn will call gecko_external_signal() with events for PB0 when pressed and released. Validate that PB0 interrupts are working. Some students in the past have implemented the GPIO_ODD_IRQHandler() for PB1 to act as the "No" button for pairing. We would do this in a commercial application, but it is not required for this assignment.

4- Add code to initialize your Server's GATT database with sl_bt_gatt_server_write_attribute_value(). Handle new events to update the button state characteristic value using sl_bt_gatt_server_write_attribute_value() when the button is pressed and released. This updating of the local GATT database shall happen regardless of whether indications are enabled or disabled for the button_state characteristic.

5- Now configure and initialize the security manager. Add code to handle the events for "Numeric Comparison". The SiLabs flowchart for numeric comparison won't be followed verbatim. You'll call sl_bt_sm_passkey_confirm() when PB0 is pressed for confirmation, and only after the preceding security manager calls in the flow chart on the SiLabs documentation page have successfully executed. State bubble diagrams and UML diagrams are your friend.

6- Remember to check return codes for the new API calls you've added to your code. Log all errors.

7- For your Server, your existing state machine for taking a temperature measurement should not change. Adding support for security for your Server is really just responding to events created by actions from the Client or by pressing PB0. So there is no need to create an additional state machine. For your Server it's more like: we get a boot event and make some API calls, we get an open event and make some calls etc. Security for your Server is the same. Here is the Responder flow chart from Silabs documentation:



Hint: Call this from your PB0 Event Responder

So you need to have event handlers for the events listed in this flow chart. After you display the passkey to the user, you'll have to figure out how to call sl_bt_sm_passkey_confirm() from your external signal event handler for PB0. You probably want some qualifiers (if statements) to check the bonded state and if you have a passkey after/when PB0 ("Yes") is pressed, to qualify calling sl_bt_sm_passkey_confirm().

The events for the security responder should be:

```
sl_bt_evt_sm_confirm_bonding_id
sl_bt_evt_sm_confirm_passkey_id
sl_bt_evt_system_external_signal_id (PB0 pressed)
```

Note: PB0 is used for 2 purposes:

a) Update the local GATT database, and to send indications to the Client for the button_state characteristic value.
b) To function as the "Yes" button for Numeric comparison.

To implement the functionality for these 2 purposes think "Isolation of functionality" - one section of code for GATT purposes, a second section of code for security purposes.

If configured correctly your app will receive either:

```
sl_bt_evt_sm_bonded_id
```
or
```
sl_bt_evt_sm_bonding_failed_id
```

APIs to call:

```
sl_bt_sm_delete_bondings()
sl_bt_sm_configure(flags, sm_io_capability_displayyesno)
sl_bt_sm_bonding_confirm()
sl_bt_sm_passkey_confirm()
sl_bt_gatt_server_write_attribute_value(button_state)
```

Closely read the documentation for sl_bt_sm_configure() to determine the appropriate value for the flags parameter.

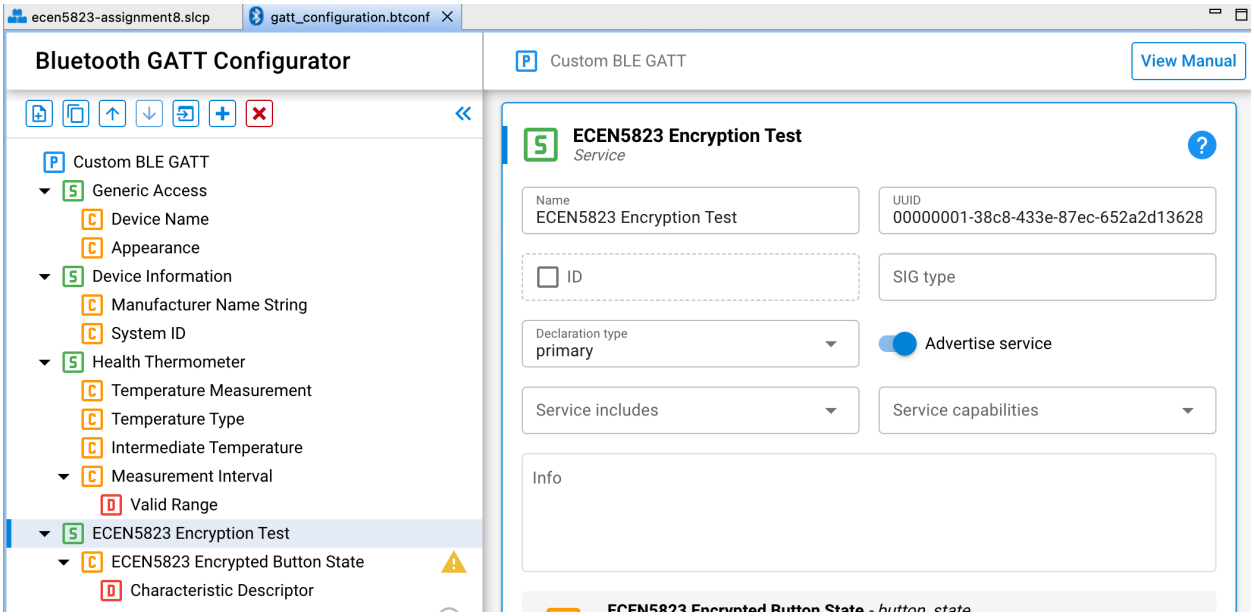
8- I recommend you get all of the button interrupts/events, then button state GATT updates and sending indications, and security code implemented and working first to get to the bonded state (guidance steps 1 to 5 above). Lastly add the circular queuing (circular buffer) functionality for indications. Add your circular buffer code to ble.c/.h.

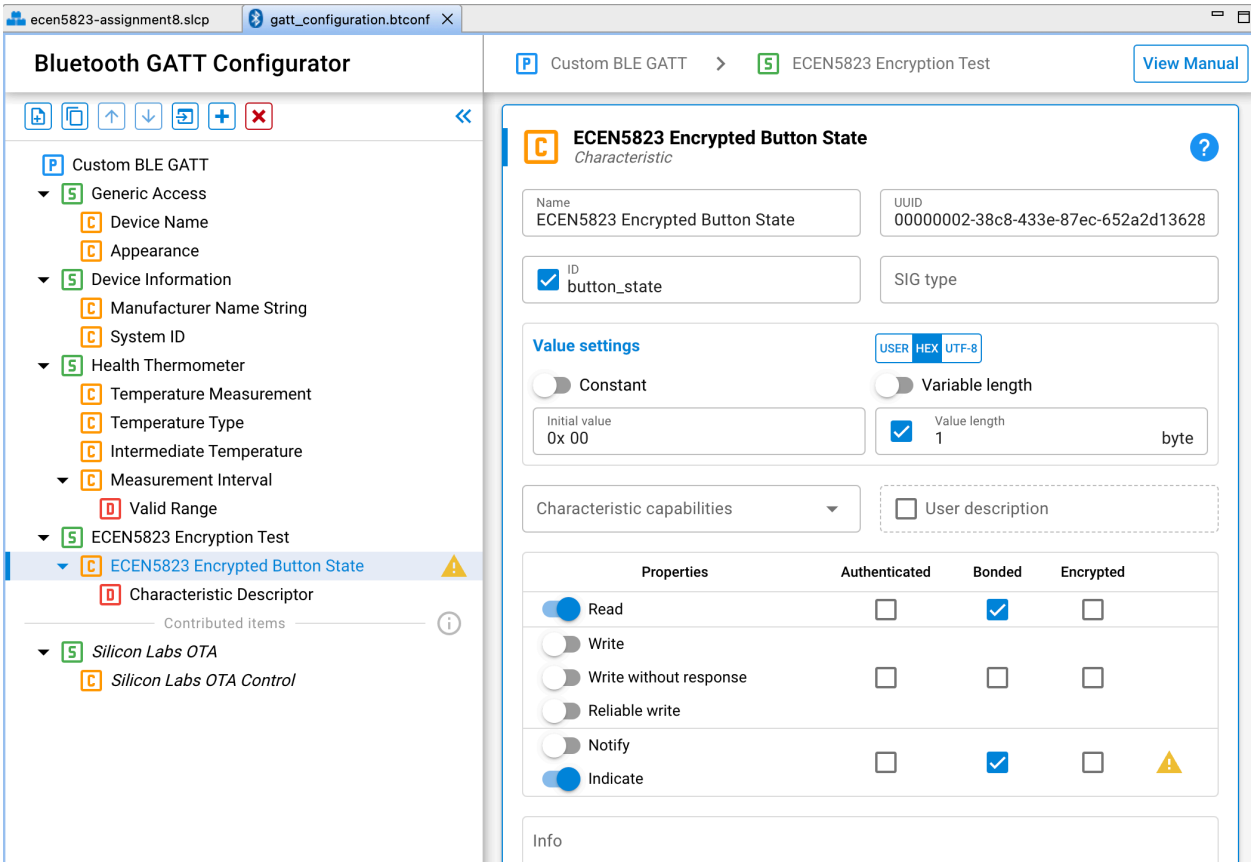
9- All previous functionality from A6 regarding temperature measurements and display of the temperature measurements shall be maintained.

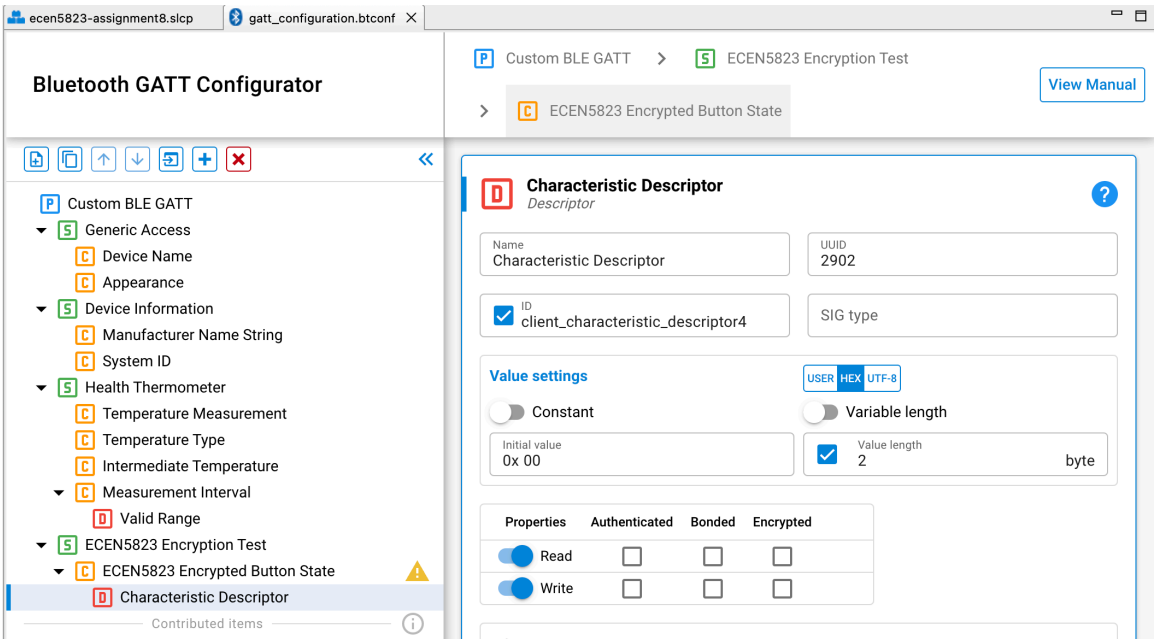
Files in your src/ directory should look like:

Appendix

Custom encrypted button service, characteristic, and CCCD example, for SSv5.7:



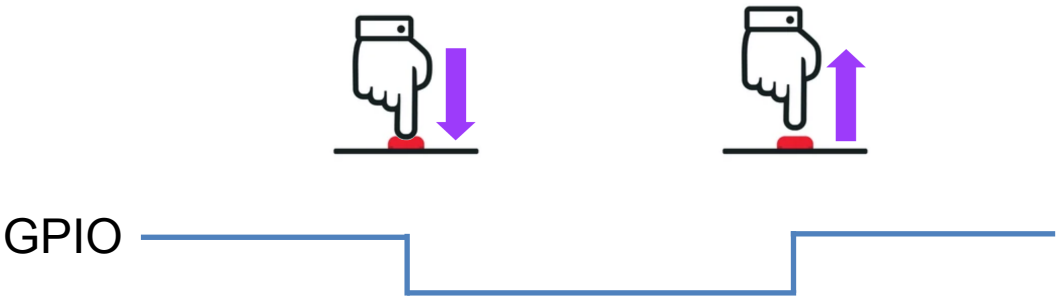




Recall that **bonding** requires **pairing** to establish the STK, and then establishing the LTK, which encrypts the link between the devices. Once both devices have the LTK and has saved the LTK, the devices are considered **bonded**, and the link is **encrypted**.

Appendix

What constitutes a button press? The “press” action or the “release” action? I believe valid arguments can be made for either case. The following figure shows the value read by software when we press and release one of the buttons on our Gecko base board:



Make a choice as to which edge you consider “pressed”. In my implementations I chose the falling edge for “press”. Remember: The MCU has glitch filtering available on the input to help you debounce the button presses and releases.