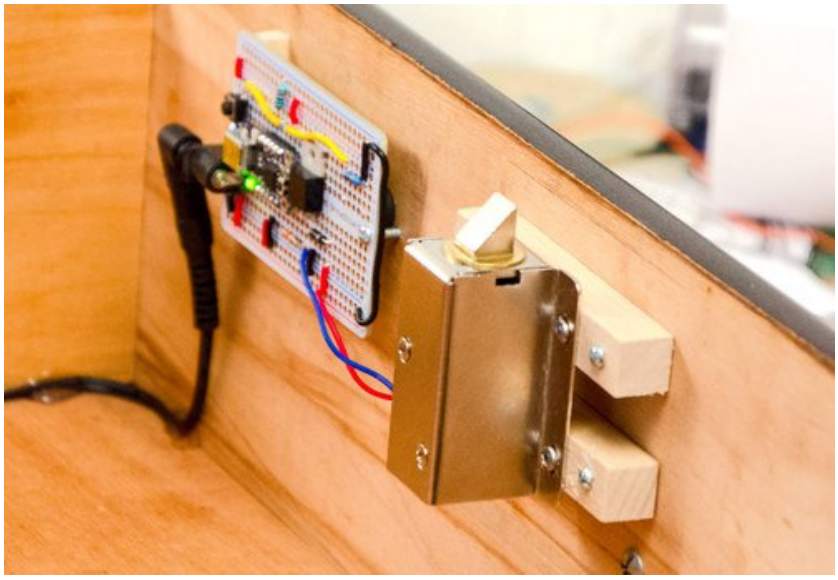
Secret Knock Activated Drawer Lock

Created by Steve Hoefer



Last updated on 2016-10-17 03:16:01 AM UTC

Guide Contents

Guide Contents 2
Overview 3
Parts & Tools 4
Parts & Materials 4
Getting Started 5
Code 6
Wiring 11
Build Notes: 12
Operation 13
Installation 14
Installing The Lock Solenoid 14
Installing The Lock Sensor 14
You're done! 15
Troubleshooting 16

Overview

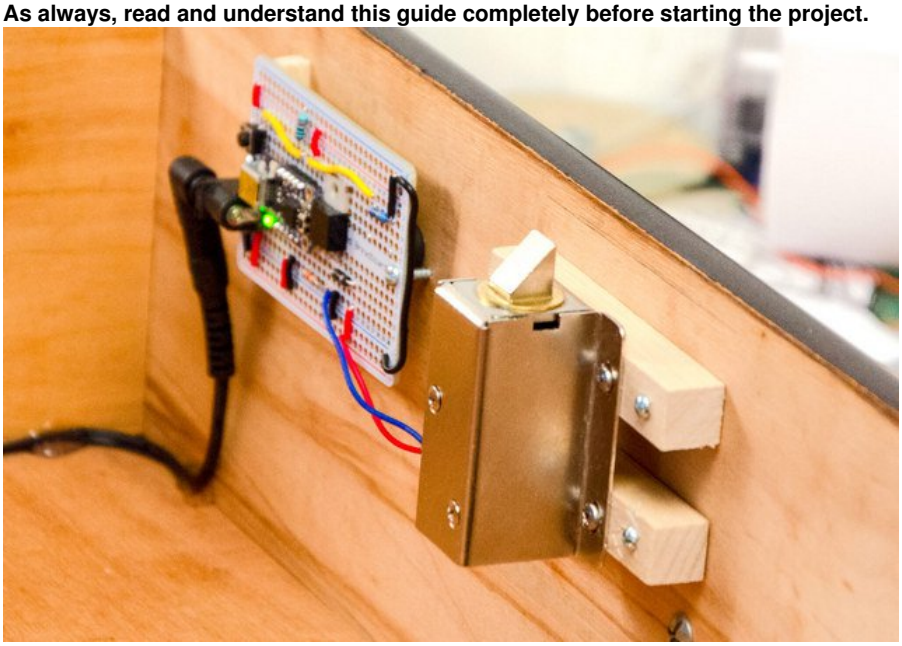
Nothing says "There's something valuable here!" than the sight of a lock. But what if the lock was invisible and the unlocking key could be transmitted through solid matter? No one would even know there was a lock, much less how to pick it.

That's what the Secret Knock Activated Drawer Lock does. It hides all of the lock mechanism away and can only be unlocked by something you know: a secret pattern of knocks.

A solenoid locks secures the drawer while a piezo buzzer listens for knocks. ATrinket (http://adafru.it/1501) compares the knock pattern to the stored secret knock and if they match the solenoid latch retracts and the drawer can be opened. Setting your own custom knock is as simple as holding down a button and tapping the new rhythm.

The project is relatively straight forward and you should be able to complete it in an afternoon.

As always, read and understand this guide completely before starting the project.



The project is for fun. Don't use it for serious security or to lock people inside of anything.

Parts & Tools

This project only requires common tools and materials. You might need to make changes or additions depending on your specific installation.

Parts & Materials

- **A drawer or door** (Preferably wood, without a latch.)
- **Adafruit Trinket (http://adafru.it/1501)** 5V.
- **Lock-style solenoid (http://adafru.it/1512)**. 12V.
- **Piezo Buzzer (http://adafru.it/160)**. 0.5" - 1" (12mm-24mm) diameter.
- **TIP120 (http://adafru.it/976)** Darlington Transistor.
- **6mm Tactile switch (http://adafru.it/367)**
- **1MΩ** 1/4W resistor.
- **10KΩ** 1/4W resistor.
- **2.2KΩ** 1/4W resistor.
- **12V 1A power supply (http://adafru.it/798)**. (With 2.1mm center positive power plug.)
- or, a **8xAA battery pack** (http://adafru.it/875) (w/2.1mm plug) - will last about a week powering the circuit
- **2.1 mm barrel jack (http://adafru.it/373).**
- **1N4001 diode (http://adafru.it/755)**
- **Perfboard (http://adafru.it/571)**.
- **Hookup Wire**.
- (6x) **#4 3/8" round head wood screws.** (Length depends on your installation. I used 2 1/2" and (4) 3/4" screws for the demonstration.)
- **Thin scrap wood.** (Again, this depends on your installation. I used 6" of 1/2" square dowel and 2" of 1/4" square dowel for the demonstration.)

Tools

- Safety glasses
- Soldering iron & solder
- Wire cutters
- Wire strippers
- Drill and assorted drill bits
- Pliers
- Screw driver
- Ruler

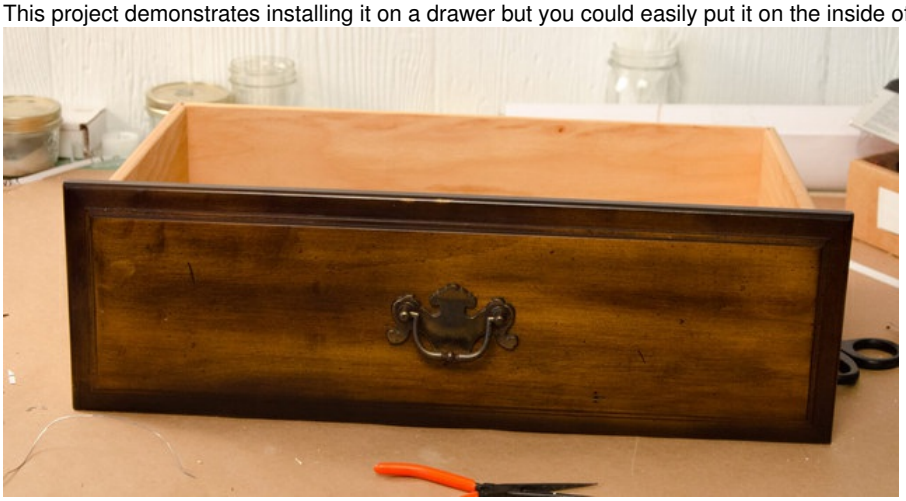
Getting Started

If you're not already experienced with the Trinket microcontroller, take the time to set it up and make sure you can load working code onto it. The "Introducing Trinket" (http://adafru.it/cEu) guide will walk you through everything you need to know to get it set up and working.

You will also need a good drawer or door to install the lock. A "good" drawer means:

- At least 2 1/4" deep. (57mm)
- A wood front at least 3/8" (10mm) thick.
- Something that you can drill holes into without getting into trouble.

This project demonstrates installing it on a drawer but you could easily put it on the inside of a cabinet door.



Code

This project uses straight-forward Arduino code and the standard Arduino EEPROM library. If you haven't set up the Arduino IDE to work with the Trinket yet, follow the "Introducing Trinket (http://adafru.it/cEu)" guide before continuing.

Copy and paste the code below into a new sketch and upload it to your Trinket.

```
/* Secret Knock Trinket
   Code for running a secret knock lock on the Adafruit Trinket.

   Version 13.10.31  Built with Arduino IDE 1.0.5

   By Steve Hoefer http://grathio.com
   Licensed under Creative Commons Attribution-Noncommercial-Share Alike 3.0
   http://creativecommons.org/licenses/by-nc-sa/3.0/us/
   (In short: Do what you want, as long as you credit me, don't relicense it, and don't sell it or use it in anything you sell without contacting me.)

   ------Wiring------
   Pin 0: Record A New Knock button.
   Pin 1: (uses the built in LED)
   Pin 2 (Analog 1): A piezo element for beeping and sensing knocks.
   Pin 3: Connects to a transistor that opens a solenoid lock when HIGH.
*/

#include <EEPROM.h>
const byte eepromValid = 123;    // If the first byte in eeprom is this then the data is valid.

/*Pin definitions*/
const int programButton = 0;   // Record A New Knock button.
const int ledPin = 1;          // The built in LED
const int knockSensor = 1;     // (Analog 1) for using the piezo as an input device. (aka knock sensor)
const int audioOut = 2;        // (Digial 2) for using the peizo as an output device. (Thing that goes beep.)
const int lockPin = 3;         // The pin that activates the solenoid lock.

/*Tuning constants. Changing the values below changes the behavior of the device.*/
int threshold = 3;                // Minimum signal from the piezo to register as a knock. Higher = less sensitive. Typical values 1 - 10
const int rejectValue = 25;       // If an individual knock is off by this percentage of a knock we don't unlock. Typical values 10-30
const int averageRejectValue = 15; // If the average timing of all the knocks is off by this percent we don't unlock. Typical values 5-20
const int knockFadeTime = 150;    // Milliseconds we allow a knock to fade before we listen for another one. (Debounce timer.)
const int lockOperateTime = 2500;  // Milliseconds that we operate the lock solenoid latch before releasing it.
const int maximumKnocks = 20;     // Maximum number of knocks to listen for.
const int knockComplete = 1200;   // Longest time to wait for a knock before we assume that it's finished. (milliseconds)

byte secretCode[maximumKnocks] = {50, 25, 25, 50, 100, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};  // Initial setup: "Shave and a Hair Cut, two bits."
int knockReadings[maximumKnocks];   // When someone knocks this array fills with the delays between knocks.
int knockSensorValue = 0;           // Last reading of the knock sensor.
boolean programModeActive = false;   // True if we're trying to program a new knock.

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(lockPin, OUTPUT);
  readSecretKnock();   // Load the secret knock (if any) from EEPROM.
  doorUnlock(500);     // Unlock the door for a bit when we power up. For system check and to allow a way in if the key is forgotten.
  delay(500);          // This delay is here because the solenoid lock returning to place can otherwise trigger and inadvertent knock.
}

void loop() {
  // Listen for any knock at all.
  knockSensorValue = analogRead(knockSensor);

  if (digitalRead(programButton) == HIGH){  // is the program button pressed?
    delay(100);   // Cheap debounce.
    if (digitalRead(programButton) == HIGH){
      if (programModeActive == false){     // If we're not in programming mode, turn it on.
        programModeActive = true;          // Remember we're in programming mode.
        digitalWrite(ledPin, HIGH);        // Turn on the red light too so the user knows we're programming.
        chirp(500, 1500);                  // And play a tone in case the user can't see the LED.
        chirp(500, 1000);
      } else {                             // If we are in programing mode, turn it off.
```

```
      programModeActive = false;
      digitalWrite(ledPin, LOW);
      chirp(500, 1000);                 // Turn off the programming LED and play a sad note.
      chirp(500, 1500);
      delay(500);
     }
     while (digitalRead(programButton) == HIGH){
      delay(10);                        // Hang around until the button is released.
     }
    }
    delay(250);   // Another cheap debounce. Longer because releasing the button can sometimes be sensed as a knock.
  }


  if (knockSensorValue >= threshold){
     if (programModeActive == true){  // Blink the LED when we sense a knock.
      digitalWrite(ledPin, LOW);
     } else {
      digitalWrite(ledPin, HIGH);
     }
     knockDelay();
     if (programModeActive == true){  // Un-blink the LED.
      digitalWrite(ledPin, HIGH);
     } else {
      digitalWrite(ledPin, LOW);
     }
     listenToSecretKnock();           // We have our first knock. Go and see what other knocks are in store...
  }

}

// Records the timing of knocks.
void listenToSecretKnock(){
  int i = 0;
  // First reset the listening array.
  for (i=0; i < maximumKnocks; i++){
    knockReadings[i] = 0;
  }

  int currentKnockNumber = 0;               // Position counter for the array.
  int startTime = millis();                 // Reference for when this knock started.
  int now = millis();

  do {                                      // Listen for the next knock or wait for it to timeout.
    knockSensorValue = analogRead(knockSensor);
    if (knockSensorValue >= threshold){                  // Here's another knock. Save the time between knocks.
      now=millis();
      knockReadings[currentKnockNumber] = now - startTime;
      currentKnockNumber ++;
      startTime = now;

       if (programModeActive==true){     // Blink the LED when we sense a knock.
        digitalWrite(ledPin, LOW);
       } else {
        digitalWrite(ledPin, HIGH);
       }
       knockDelay();
       if (programModeActive == true){  // Un-blink the LED.
        digitalWrite(ledPin, HIGH);
       } else {
        digitalWrite(ledPin, LOW);
       }
    }

    now = millis();

    // Stop listening if there are too many knocks or there is too much time between knocks.
  } while ((now-startTime < knockComplete) && (currentKnockNumber < maximumKnocks));

  //we've got our knock recorded, lets see if it's valid
  if (programModeActive == false){          // Only do this if we're not recording a new knock.
    if (validateKnock() == true){
      doorUnlock(lockOperateTime);
    } else {
      // knock is invalid. Blink the LED as a warning to others.
      for (i=0; i < 4; i++){
```

```
        digitalWrite(ledPin, HIGH);
        delay(50);
        digitalWrite(ledPin, LOW);
        delay(50);
      }
    }
  } else { // If we're in programming mode we still validate the lock because it makes some numbers we need, we just don't do anything with the return.
    validateKnock();
  }
}


// Unlocks the door.
void doorUnlock(int delayTime){
  digitalWrite(ledPin, HIGH);
  digitalWrite(lockPin, HIGH);
  delay(delayTime);
  digitalWrite(lockPin, LOW);
  digitalWrite(ledPin, LOW);
  delay(500);   // This delay is here because releasing the latch can cause a vibration that will be sensed as a knock.
}

// Checks to see if our knock matches the secret.
// Returns true if it's a good knock, false if it's not.
boolean validateKnock(){
  int i = 0;

  int currentKnockCount = 0;
  int secretKnockCount = 0;
  int maxKnockInterval = 0;               // We use this later to normalize the times.

  for (i=0;i<maximumKnocks;i++){
    if (knockReadings[i] > 0){
      currentKnockCount++;
    }
    if (secretCode[i] > 0){
      secretKnockCount++;
    }

    if (knockReadings[i] > maxKnockInterval){   // Collect normalization data while we're looping.
      maxKnockInterval = knockReadings[i];
    }
  }

  // If we're recording a new knock, save the info and get out of here.
  if (programModeActive == true){
      for (i=0; i < maximumKnocks; i++){ // Normalize the time between knocks. (the longest time = 100)
        secretCode[i] = map(knockReadings[i], 0, maxKnockInterval, 0, 100);
      }
      saveSecretKnock();                // save the result to EEPROM
      programModeActive = false;
      playbackKnock(maxKnockInterval);
      return false;
  }

  if (currentKnockCount != secretKnockCount){  // Easiest check first. If the number of knocks is wrong, don't unlock.
    return false;
  }

  /*  Now we compare the relative intervals of our knocks, not the absolute time between them.
      (ie: if you do the same pattern slow or fast it should still open the door.)
      This makes it less picky, which while making it less secure can also make it
      less of a pain to use if you're tempo is a little slow or fast.
  */
  int totaltimeDifferences = 0;
  int timeDiff = 0;
  for (i=0; i < maximumKnocks; i++){    // Normalize the times
    knockReadings[i]= map(knockReadings[i], 0, maxKnockInterval, 0, 100);
    timeDiff = abs(knockReadings[i] - secretCode[i]);
    if (timeDiff > rejectValue){        // Individual value too far out of whack. No access for this knock!
      return false;
    }
    totaltimeDifferences += timeDiff;
  }
  // It can also fail if the whole thing is too inaccurate.
  if (totaltimeDifferences / secretKnockCount > averageRejectValue){
```

```
    return false;
  }

  return true;
}


// reads the secret knock from EEPROM. (if any.)
void readSecretKnock(){
  byte reading;
  int i;
  reading = EEPROM.read(0);
  if (reading == eepromValid){    // only read EEPROM if the signature byte is correct.
    for (int i=0; i < maximumKnocks ;i++){
      secretCode[i] =  EEPROM.read(i+1);
    }
  }
}


//saves a new pattern too eeprom
void saveSecretKnock(){
  EEPROM.write(0, 0);  // clear out the signature. That way we know if we didn't finish the write successfully.
  for (int i=0; i < maximumKnocks; i++){
    EEPROM.write(i+1, secretCode[i]);
  }
  EEPROM.write(0, eepromValid);  // all good. Write the signature so we'll know it's all good.
}

// Plays back the pattern of the knock in blinks and beeps
void playbackKnock(int maxKnockInterval){
      digitalWrite(ledPin, LOW);
      delay(1000);
      digitalWrite(ledPin, HIGH);
      chirp(200, 1800);
      for (int i = 0; i < maximumKnocks ; i++){
        digitalWrite(ledPin, LOW);
        // only turn it on if there's a delay
        if (secretCode[i] > 0){
          delay(map(secretCode[i], 0, 100, 0, maxKnockInterval)); // Expand the time back out to what it was. Roughly.
          digitalWrite(ledPin, HIGH);
          chirp(200, 1800);
        }
      }
      digitalWrite(ledPin, LOW);
}

// Deals with the knock delay thingy.
void knockDelay(){
  int itterations = (knockFadeTime / 20);      // Wait for the peak to dissipate before listening to next one.
  for (int i=0; i < itterations; i++){
    delay(10);
    analogRead(knockSensor);                  // This is done in an attempt to defuse the analog sensor's capacitor that will give false readings on high impedance sensors.
    delay(10);
  }
}

// Plays a non-musical tone on the piezo.
// playTime = milliseconds to play the tone
// delayTime = time in microseconds between ticks. (smaller=higher pitch tone.)
void chirp(int playTime, int delayTime){
  long loopTime = (playTime * 1000L) / delayTime;
  pinMode(audioOut, OUTPUT);
  for(int i=0; i < loopTime; i++){
    digitalWrite(audioOut, HIGH);
    delayMicroseconds(delayTime);
    digitalWrite(audioOut, LOW);
  }
  pinMode(audioOut, INPUT);
}
```

For the curious, here's a quick summary of what the code does:

1. It initializes everything, set the appropriate input and output pins, and loads the saved knock pattern (if any) from EEPROM.
2. After everything is initialized it listens for a spike on the piezo sensor. When its above a certain threshold it counts as a "knock".

3. When it hears a knock it starts a timer and then listens for more knocks.
4. When it gets another knock it saves the time between knocks to an array.
5. When there are no more knocks (There is at least 1.2 seconds without a knock) it checks to see if the sequence is correct.
6. It does this by normalizing the time between knocks, that is making them relative to each other, not the exact milliseconds between them. So the longest time between knocks becomes 100, half that time becomes 50, etc. To use the language of music whole notes are 100, half notes are 50, quarter notes are 25, etc.
7. It compares these values to the stored knock. If they match (or nearly match) the lock opens for a couple seconds and then goes back to #2. If the knock doesn't match, it blinks a light for failure and then goes back to #2.

It does other things too, like read the button to see if it should program a new knock, etc.

The code is liberally commented and but here are a couple things you might be interested in modifying:

```
int threshold = 3;
```

(line 29) If the knock sensor is too sensitive you can try raising this value. Usually values below 10 work best, but values up to 250 have been known to work. (If you need to raise this above 250 then you probably have something wrong with your circuit or components.)

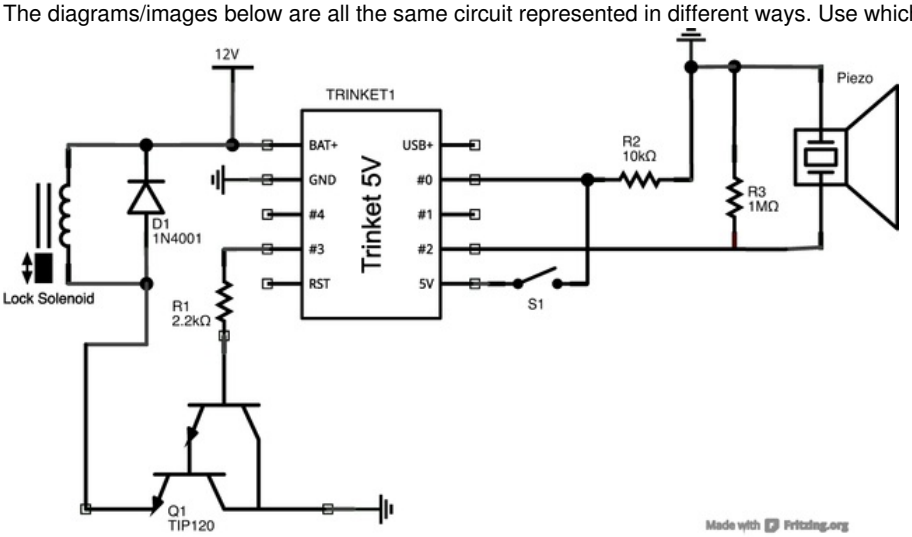
```
const int rejectValue = 25;
const int averageRejectValue = 15;
```

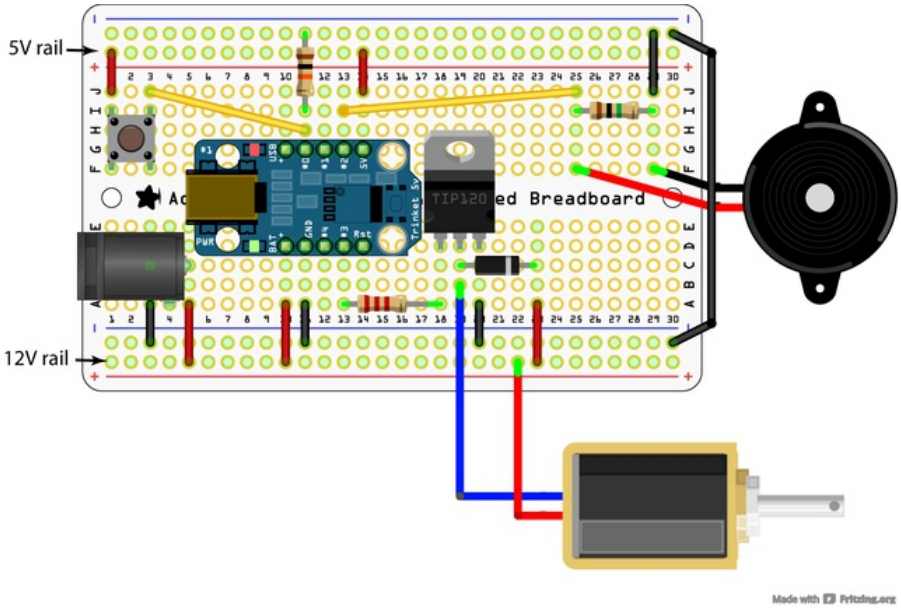
(line 30-ish) These two percentages indicate how accurate the knock needs to be to unlock. Raising these numbers allows a sloppier knock to work. Lowering them requires a more strict knock.

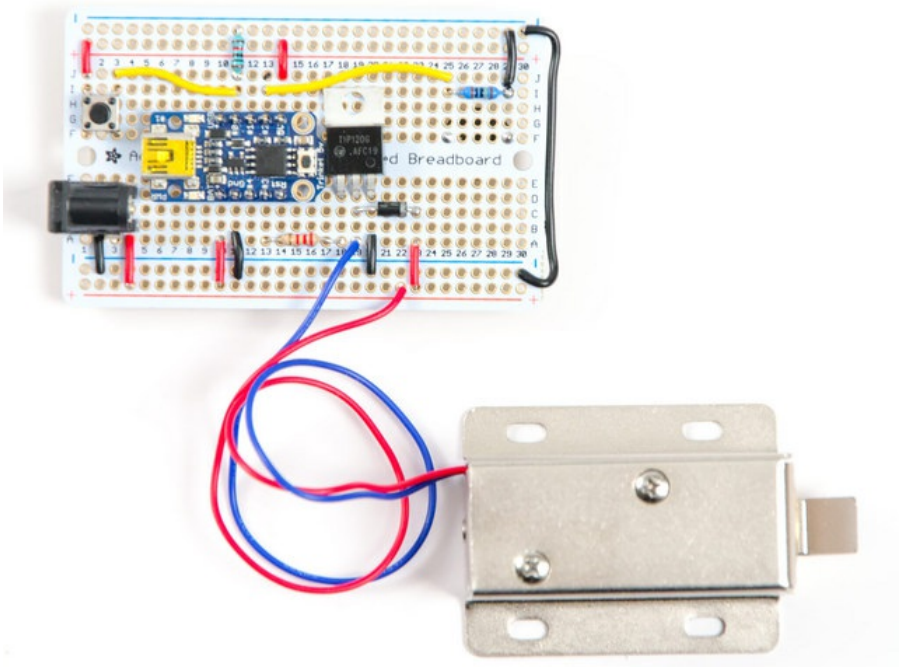
Wiring

I recommend building the circuit on a solderless breadboard first to make sure that everything works before soldering it into place. Check out the "Operation" section for use and testing.

The diagrams/images below are all the same circuit represented in different ways. Use whichever are most helpful.







Build Notes:

1: The Piezo. Solder the piezo buzzer to the ***back*** of the PCB so it can be installed flush against the desk/drawer/etc and hear knocks more clearly.

2: Wire Length. The length of wire between the lock and the knock sensor depends on where you plan to mount it. (See "Installation" section for more information.) Measure your distances and be sure to allow some extra for movement of drawers and doors. You should be able to put the lock several feet from the detector without any problems.

If you're installing them close to each other then simply cut the connector off the solenoid lock.

3: Programming. The circuit uses the Trinket's pin #3. In some cases you might need to disconnect the 2.2KΩ resistor from pin #3 to program the circuit.

Note that my sample programs just fine with pin 3 connected, but your mileage may vary. If you want to be extra sure of being able to program it after it's soldered in place consider using female headers (http://adafru.it/598) to mount the Trinket, or put a jumper between pin #3 and the 2.2KΩ resistor.

Operation

Make sure everything works before installing it.

Unplug the Trinket from USB and plug in the 12V power supply and then be sure it can accurately detect a knock, unlock on command, and record new knocks, as described below.

If it doesn't behave, see the "Troubleshooting" section for what might be going wrong.

Startup:

1. The Trinket's green power LED will light up.
2. The Trinket's red LED will light up.
3. The solenoid lock will click open twice. (First really quickly, the second time for about a second or two.)
4. The Trinket's red LED will turn off and the lock will close.

When the red light goes out and the lock closes, it's now listening for a knock and is ready to go.

Operation:
The default secret knock is the classic Shave And A Haircut (http://adafru.it/cSr). To change it, see "recording" below.

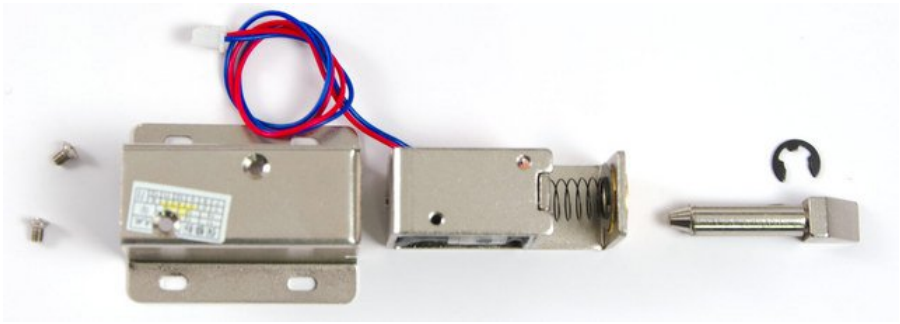
1. The red LED will blink when it senses a knock. (It should also sense if you lightly tap the piezo with your finger.)
2. If the wrong sequence if knocks (or taps) is detected the red LED will flicker briefly.
3. If the correct sequence is detected the red LED will light and the lock solenoid will retract for a couple seconds.

Recording a new secret knock:

1. Press the record button. The red LED will light and the piezo will play a short tone to let you know its listening for a new knock. (Note: This button won't work if it senses a knock in the last couple of seconds.)
2. (If you change your mind, press the button again and the red light will go off, and it will play a different tone.)
3. Kock your new knock pattern. The red light will blink off when it hears a knock.
4. When you're done, wait a second or two for it to register. It will then playback the knock pattern by both blinking the red LED and beeping the piezo.
5. This is now saved as a new knock and it will be remembered even if it's powered off.
6. If you don't like the knock or you made a mistake, press the button again to record a different pattern.

Now that you're sure that it works, it's time to install it.

Installation



Installing The Lock Solenoid

First you need to rotate the latch so it will latch when you close the drawer or door.

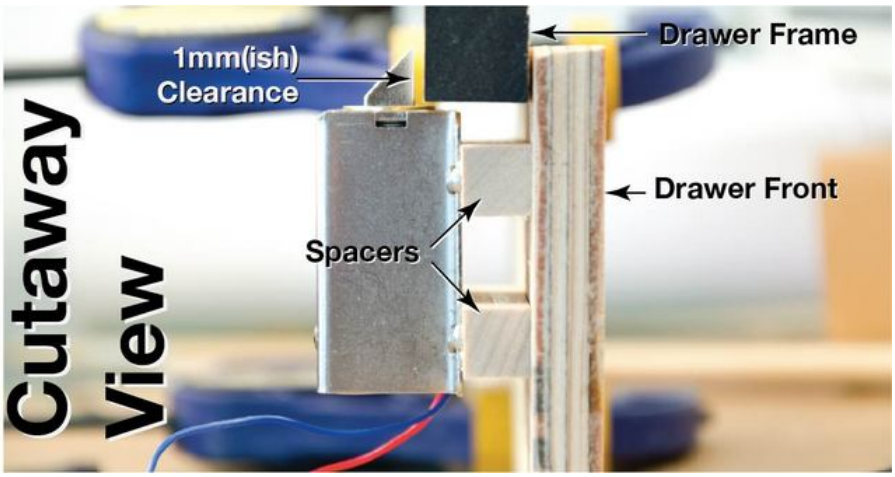
1. Remove the screws from the solenoid cover. (One of mine was hidden under a sticker.)
2. Remove the E-ring ring that holds the latch slug in place. (It's kind of hidden between the spring and the square brass collar.) If you don't have the specific tool from removing these, you can usually pry it off with needle-nose pliers and/or a screwdriver.
3. Rotate the latch slug 90º so the long square side is on the side of the lock with the mounting holes.
4. Replace the E-ring and screw the cover back on.

We'll do the simplest installation which places the lock solenoid on the front of the drawer and latches against the frame of the drawer or door.

The lock solenoid needs to have enough spacers behind it so the latch will clear the frame when the drawer is closed so the latch will lock.

The latch also needs to be able to move freely when the drawer is closed. The solenoid isn't super powerful, and if it's wedged too tightly it won't retract when activated. You will probably need to add add some spacers between the lock and the drawer/door.

Position it vertically so the latch smoothly slides under the drawer frame and fasten it in place with #4 wood screws. Drill pilot holes for the screws so the wood doesn't split.



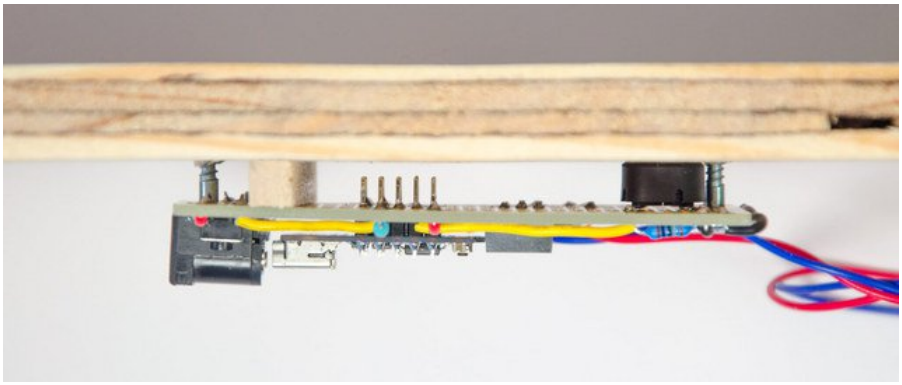
For example in the side view above, the drawer frame is 3/4" (19mm) thick. The latch has 5/16" (8mm) of clearance built in, so I need to add 1/2" (12mm) of clearance to make sure it clears the frame and can still retract fully. I used two short pieces of 1/2" square dowel.

Installing The Lock Sensor

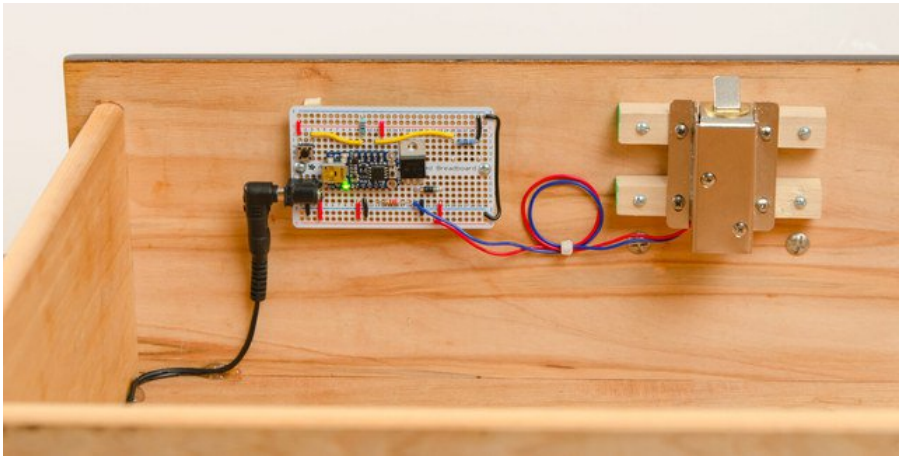
The knock sensor needs to be fastened somewhere out of sight, but where you can knock easily and clearly on the other side. For a door that location is probably on the inside of the door. For a drawer it can be inside of the face of the drawer, along the side of the cabinet or under the surface of a the desk.

Wherever you put it you'll probably want to be able to reach the programming button.

Use 1/2" #4 wood screws to attach the perfboard to the inside of the sounding surface. Attach it so the piezo is against the knocking surface, and carefully tighten the screws to hold it in place. Reinforce and stabilize the perf board with some spacers of scrap lumber. (I used a bit of 1/4" (6mm) square dowel between the perfboard and the drawer face, you can see it to the left in the picture below.)



After everything is in place and tested, tack the cables in place with hot glue, cable ties, or cable staples.



You're done!

Remember: if you forget the knock or something else goes wrong you can disconnect the power for a moment to unlatch the lock for a couple seconds. Otherwise enjoy your super secret hiding place!

Troubleshooting

Problem: It doesn't seem to work very reliably
Solution: The knock sensor needs at least 1 1/2 seconds of silence before*and after* you enter your knock pattern. If there are any stray knocks during that time it will mess up the reading and not unlock.

The knock sensor picks up any vibration that's strong enough. That means things like loose drawer pulls, wobbly drawers, and loose items lying on top of the desk can inadvertently set off the knock sensor. Sitting down a glass, slamming a door, and even loud music can trigger a knock. The knock surface should also be secure and firm. If it visibly moves when you knock it can trigger phantom knocks. Either improve the environment around the knocks ensor, move it to quieter place, or see "*The Knock Sensor Is Too Sensitive*" for other ways to work around it.

Alternately the sensor might not be sensitive enough! See "*The Knock Sensor Is Not Sensitive Enough*" for other ways to work around it.

How do you tell the difference? Knock on your surface while looking at the knock sensor. The red light will blink each time it hears a knock.
Problem: The red LED blinks constantly or randomly.
Solution 1: This usually happens when there is an error or a short in the circuit. Check each connection in the circuit for continuity and shorts, be sure all of the components are the correct values and are connected the right way.

Solution 2: This can rarely be caused by a noisy power supply or external electronic noise near the sensor. Try a regulated power supply and move any high-current devices (electric heaters, refrigerators, air conditioners, etc) away from the sensor.
Problem: The knock sensor isn't sensitive enough!
Solution 1: Change the value of int threshold = 3; in the sketch (line 29) to 2, 1 or even 0. Be sure to upload the modified sketch to the Trinket after you make the change. You might need to temporarily disconnect the connection at pin #3 for it to program correctly.

Solution 2: Use a more sensitive piezo element. A 3/4" element will often more sensitive. (Avoid larger, louder elements as they include driver circuitry which makes them poor sensors.)
Problem: The knock sensor is too sensitive!
Solution 1: Increase the value of of int threshold = 3; in the sketch. (Line 29) This might need to go as high as 200. If you have to go higher than that check your circuit for errors. (See also "The red LED blinks constantly or randomly" above.

Upload the modified sketch to the Trinket after you make the change. You might need to temporarily disconnect the connection at pin #3 for it to program correctly.

Solution 2: Try a smaller piezo element. 1/2" (12mm) elements are about as small as you want to go.
Augh! I forgot my knock and now I can't get in!
Unplug the power to the lock for a bit, then plug it back in. When the Trinket is rebooting it will open the solenoid for a second or two, letting you open the drawer and record a new knock.
I know I knock the correct pattern, but the latch doesn't open.
If the lock opens when the drawer is open, but doesn't open when the drawer is closed that usually means that the lock latch is binding somewhere. It doesn't have a lot of pull so it needs to be able to move freely. Make sure the drawer is closed fully and that the solenoid latch can move freely.
Problem: It reboots when it's trying to unlock
Solution: You probably need a power supply with more amperes. Make sure your power supply provides at least 1A (1000mA) or the latch will pull too much power when it tries to open and the Trinket will reboot.