KidCQ: A Prototype System for Direction Finding Abducted Children

Caroline Guay - VA3WYZ, Mike Kennedy - VA3TEC & Brian Neill -VA3BPN

August 10, 2004

Abstract

This paper was written to document the design and implementation of the KidCQ prototype tracking system which could help in locating abducted children. KidCQ places an emphasis on protecting the privacy of children. Strong protective mechanisms have been incorporated to prevent fraudulent use of this tracking system.

1 KidCQ Overview

The KidCQ prototype is a child tracking system with a strong focus on protecting the child's privacy. Essentially this paper goes beyond presenting another tracking system and focuses on describing a set of protective mechanisms that are used to prevent system abuse and protect children wearing a tracker.

With the above assertion in mind, the first "rule" of the KidCQ system, is that only the Police may activate a KidCQ tracking device. The device itself is quite simple, it is a passive device that listens to a designated frequency for an activation code. If an activation code is received and proved to be authentic, then the device will enter a beacon mode, transmitting a tone on a designated frequency so that authorities or ham operators may Radio Direction Find (RDF) the device.

Police authorities are the only group capable of generating KidCQ activation codes, this is a primary concept of the the KidCQ system that is thoroughly discussed in the following sections with the aid of technical implementation details and source code samples.

Parents also play a key role in activating a KidCQ tracking device. Activation Codes are tailor made for each device using a unique serial number. Before Police can generate an Activation Code, the parents of the missing child must give Police the serial number of their child's KidCQ tracker.

Once the serial number has been used to generate the activation code, the code must be broadcast over a designated 2 meter frequency until a tone from the sought KidCQ device can be heard. At which point RDF techniques can be used to pinpoint the device and the missing child.

Transmitting the activation code, searching for a signal, and RDFing the tracking device are all key areas where Ham radio expertise would be required by Police. The KidCQ system relies fundamentally on the emergency preparedness of the amateur radio community and our long standing tradition of providing communication services during times of emergency. An abducted child is just such an emergency.

The KidCQ system requires very close collaboration between Police, parents and the amateur community. The system would only be used within the context of a life-threatening emergency, similar in scope to the activation requirements of the well known Amber Alert program.

1.1 Technical Overview

The tracking device will not transmit any information about the child's location until it is activated by the Police. In order for Police to activate the child's tracker, the parents must first provide the police with the serial number of their child's device at which point the Police can generate an activation code using a special KidCQ smart card.

Once an activation code has been generated for the child's tracking device, it is broadcast using the APRS protocol on a designated 2 meter frequency.

All KidCQ devices are listening to this frequency for an APRS based KidCQ activation messages addressed specifically to them. When a message is received, the device evaluates the message, and if an authentic activation code is present, the device will transmit a beacon tone on 2 meters.

The activation code is discussed in further detail later, but it is important to note that activation codes are generated and authenticated using a form of **cryptography** known as **digital signatures**. The legitimacy of experimenting with this technique as a Ham radio operator is also discussed in sections below.

For now, the reader is asked to take it on faith that secret encrypted messages are not being transmitted on Amateur bands. A device that receives a digitally signed activation code, will immediately verify that the code is authentic and was produced by Police. At which time, the device will begin actively broadcasting a beacon tone where RDF techniques can be used to track the device.

The ability to generate an activation code is restricted to Police using smart cards. A smart card is similar to a banking card. It is made mostly of plastic and has the same form factor as the wallet sized credit card. And like a bank card, the owner is required to have the card in-hand and have knowledge of a secret Personal Identification Number (PIN) in order to do anything useful.

While smart cards look like a typical magnetic strip credit card, in the KidCQ prototype system, the smart card is actually a small computing device with a processor and internal memory[1]. When the card is inserted into a reader, power is supplied and the user interacts with the card using a PC that is attached to the smart card reader. A Police officer inputs a PIN and a KidCQ device serial number, the smart card processes the request and sends back an activation code. The smart card has secret information that it uses to generate the activation code. However, the activation code itself is not a secret, as anyone can read, validate and understand its meaning.

Much of the security of the KidCQ system is dependent on keeping information in the smart card private and controlled, so what happens if a KidCQ smart card is stolen along with the PIN? This is a valid concern, as a single KidCQ card literally holds the key to the entire system. The key is just a piece of data, but if it is divulged then the system is broken.

The system is also broken if someone who steals the card, never actually sees the secret data, but is able to use the card to make an infinite number of activation codes starting at serial number 1 and working their way up.

To mitigate this risk, each KidCQ smart card can produce a maximum of 5 activation codes. This small fixed number of uses, allows for a search to be conducted for a child who may be wearing one of many possible tracking devices. The intention however, is that one smart card is used for one abduction search before it must be replaced. After the card has been used up, it erases it's internal memory and wipes out the secret data that is used to create KidCQ activation codes.

Limiting the use of the smart cards allows Activation Code creation to be controlled beyond simply trusting local police departments not to lose the card. Human nature will allow for more than one card to be lost or possibly stolen. Keeping low the number of code generations that can be performed by a single card will decrease the impact of losing a smart card in the field.

A smart card is like a small computer, it can perform computations and store small amounts of data, but most importantly it is self contained. Cryptographic calculations are performed on card with keys that are stored on card. This is very useful for keeping the KidCQ cryptographic keys a secret, Police can use the key to make an activation code, but they can never see the key or divulge its value.

Also, removing the secret keying data from the smart card is not an easy task. Any physical attempts to "open" the card will typically destroy it.

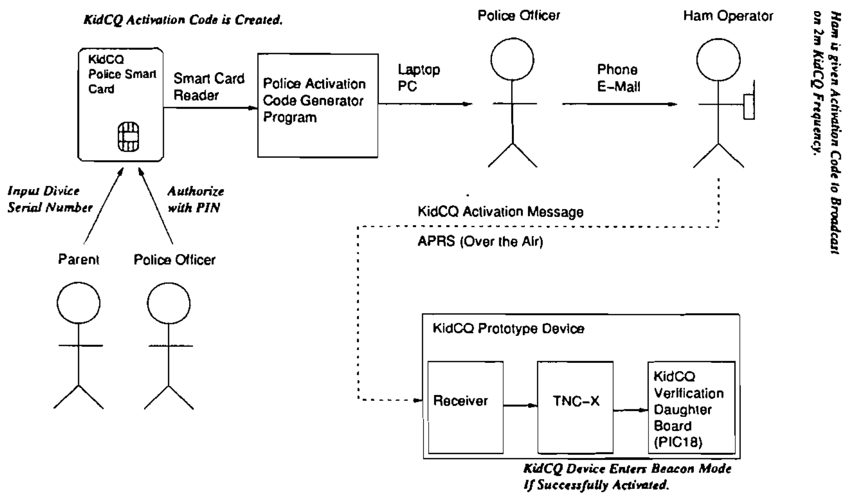

KidCQ Activation Code is Created.
KidCQ Police Smart Card
Smart Card Reader
Police Activation Code Generator Program
Laptop PC
Police Officer
Phone E-Mail
Ham Operator
Ham is given Activation Code to Broadcast on 2m KidCQ Frequency.
Input Divice Serial Number
Authorize with PIN
Parent
Police Officer
KidCQ Activation Message
APRS (Over the Air)
KidCQ Prototype Device
Receiver
TNC-X
KidCQ Verification Daughter Board (PIC18)
KidCQ Device Enters Beacon Mode If Successfully Activated.

Figure 1: KidCQ Activation Code Data-flow

1.2 Police Officer: Generating an Activation Code

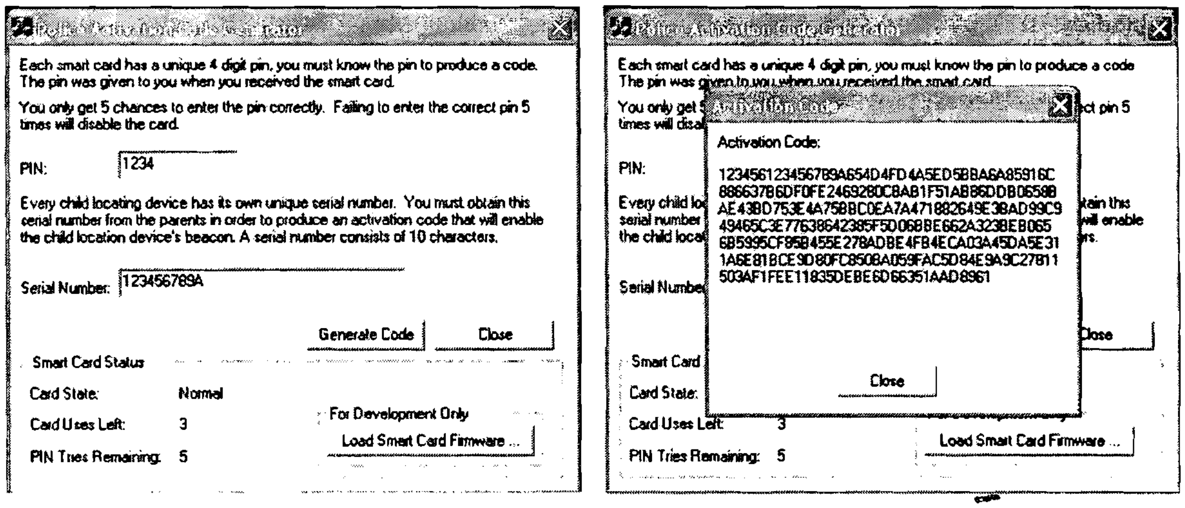
With the serial number in hand, a Police officer inserts a KidCQ smart card into a reader that is connected to a computer. A windows application as shown in figure 2 screenshots is used to interface with the smart card in the reader. The screenshots show two input fields, one for entering the PIN that will unlock the smart card, and a second field for entering the KidCQ device serial number for which an activation code is required. Once the Generate Code button is pressed, both pieces of information are sent to the smart card where the PIN will be confirmed and if the PIN is valid an activation code for will be returned for the specified serial number.

[1]but no power supply

It is important to note that the activation code is not calculated on the computer, and that the program in the screen shots is only an interface to the smart card where the real computation takes place. Details about the smart card implementation are described in section 3 below.

At the bottom of the program window is a boxed off section called "Smart Card Status" that is used to display state information queried from the smart card. This box will tell the user if the wrong smart card is plugged in, if a the KidCQ smart card has expired due to failed login attempts or if the card is inactive due to over use.

Also present in the screen shot is a button titled "load smart card firmware". This button was added for the purpose of prototype development and for easier demonstrations it is a convenient short cut for loading firmware to the card and initializing the card with a static serial number and prototype keying data.



Each smart card has a unique 4 digit pin, you must know the pin to produce a code. The pin was given to you when you received the smart card.

You only get 5 chances to enter the pin correctly. Failing to enter the correct pin 5 times will disable the card.

PIN: 1234

Every child locating device has its own unique serial number. You must obtain this serial number from the parents in order to produce an activation code that will enable the child location device's beacon. A serial number consists of 10 characters.

Serial Number: 123456789A

Smart Card Status

Card State: Normal

Card Uses Left: 3

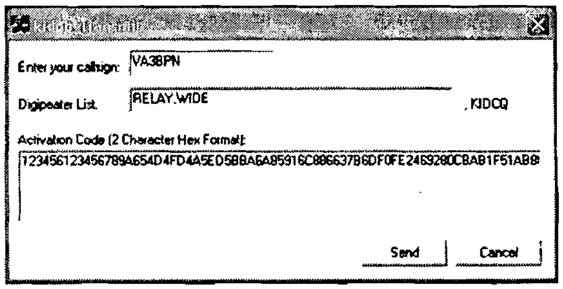
PIN Tries Remaining: 5

For Development Only

Activation Code:

```
123456123456789A654D4FD4A5ED5BBA6A85916C
886637B6DF0FE246928OC8AB1F51AB86DDB0658B
AE43BD753E4A75BBC0EA7A471882649E38AD99C9
49465C3E77638642385F5D068BE662A323BEB065
6B5995CF95B455E278ADBE4FB4ECA03A45DA5E31
1A6E81BCE9D80FC8508A059FAC5D84E9A9C27811
503AF1FEE11835DEBE6D66351AAD8961
```

Figure 2: Police Activation Code Generator



Enter your callsign: VA38PN

Digipeater List: RELAY,WIDE ,KIDCQ

Activation Code (2 Character Hex Format):

```
123456123456789A654D4FD4A5ED5BBA6A85916C886637B6DF0FE246928OC8AB1F51AB8
```

Figure 3: Program To Broadcast KidCQ Activation Message

1.3 Ham Operator: Transmitting the Code with APRS

Once an activation code has been generated by the police (see figure 2), it can be handed over to Ham radio operators for broadcasting on the designated KidCQ 2 meter frequency.

To broadcast the activation code, a Ham operator is required to have a PC, a TNC and a transmitter. Since the KidCQ activation message is wrapped in an APRS packet, a Ham is also required to have software capable of transmitting the APRS based activation message.

The prototype system uses the kidcq_transmit application shown in figure 3 to wrap the activation code in a custom APRS message. The KidCQ APRS message is specified in Appendix A: KidCQ protocol specification.

The kidcq_transmit program broadcasts the message by interfacing with a KISS mode TNC over the COM1 serial link on the windows platform. The operator must enter the activation code provided by police as well as their callsign and APRS digipeater aliases list.

KidCQ makes use of APRS aliases to propagate and filter the packet.

1.4 Direction Finding the Device

Direction finding a device that is emitting an RF signal is a distinct topic in itself. There is an entire sub-culture of Hams that practice tracking down rogue transmitters for sport. And techniques range from complex distributed triangulation of signals using using APRS[2] to lone Hams that home into a signal using a directional antenna. The later is presented as an example using a simple dipole.

It is easier to use the null (or end) of a simple dipole to determine the direction from which the signal is coming from than to use the main lobes of the dipole. This is advantageous since the null on a dipole has a narrow beam width and can be rotated and moved around until a signal can barely be heard. The spot at which the signal strength is at its weakest should position the dipole so that it is pointing to the source of the signal. Circling around the area of the potential signal source will confirm the position of the signal if the dipole is always pointing to the centre of the circle.

For the KidCQ prototype it was decided that a beacon transmitter operating in the 2m band would be used. The main reason for choosing this band was because most Hams own equipment and can operate in the 2m band. Also, it is much easier to RDF in the 2 meter band due to the fact that there are fewer reflections than in higher frequencies.

Choosing 2 meters has its drawbacks, it is a lot more challenging to design hardware within the constraints of a device that must be carried by a child. Most of the available[3] low power consumption ICs that can be found, operate in the 800-900 MHz band. And certainly receiver and transmitter ICs greatly impact size and practicality of any tracking device. But none the less, time and money can solve the problem of making a small tracker. The KidCQ prototype presents a proof of concept system that guards privacy.

2 Cryptography and Digital Signatures

Secure data communication can be a confusing topic. To help with the discussion of cryptography, two fictional characters Alice and Bob are introduced. Alice and Bob would like to communicate with each other over an untrusted, non-secure channel.

Now, in the case of ham radio, we are not so concerned with communicating secrets. In fact, this is an illegal use of our privilege. However, it may be the case that we would like to prove the identity of the person with whom we are communicating. So long as the information communicated is not secret, we are operating within the legitimacy of our license.

This section presents both methods of secure communication with Alice and Bob being used to illustrate.

In symmetric key cryptography, both Alice and Bob share a single key. If Alice wants to send a secret message to Bob, Alice will encipher the message with their shared key, send the resulting cipher text to Bob who will use the same shared key to decrypt the message and read the plain text. Now, it's extremely important to note, that in this short scenario, both Alice and Bob are not acting as ham radio operators since this method of enciphering can only be used to send secret messages. And sending secret messages over amateur bands is strictly prohibited.

Symmetric key cryptography is typically what we think of when we hear stories of the allied code breakers versus the Japanese and German code machines of World War 2. Keep in mind that all of the secrecy provided by symmetric key cryptography is dependent on a single shared key, and that the two communicating parties must know the key value and must agree on the key value at some point before using it. This type of "key management" has some inherent problems, after all, during WWII keys were not always discovered by allied code breakers, sometimes they were acquired by boarding enemy submarines or during special operations behind enemy lines. With this in mind, the second type of encryption, public key cryptography is next presented. This method of cryptography does not rely on a single shared secret key.

In 1976, Whitfield Diffie and Marty Hellman introduced a new method of enciphering, referred to today as Public Key Cryptography. Let's say Alice wants to send a secret message to Bob using public key cryptography. Bob will first produce a "Key Pair" consisting of a Public Key and a Private Key. These two keys are mathematically related, however it is considered extremely hard to deduce or calculate the Private key from the Public one. Bob's Public key is published in a phone book, or something equivalent that Alice knows she can trust, and Bob keeps his private key secret to himself. If Alice wants to communicate with Bob, she will first lookup Bob's public key in the phone book, and use it to encrypt her message to Bob. Here's the important part: the math behind public key cryptography ensures that Alice (or anyone else) can encrypt a message for Bob, but only Bob can decrypt the ciphertext using his Private key. So even though Alice encrypted a message for Bob, she is not able to decrypt her own message, only Bob can decrypt the ciphertext message.

As with symmetric key encryption, public key encryption is illegal for hams to use over the air. In fact, any time we use the word "encryption" it is a safe bet that we are talking about data secrecy, something that is strictly off limits to Ham radio operators. So why are we discussing encryption within this paper at all?

It turns out that there is a unique property of public key cryptography that can be twisted to make it both legal and useful within ham radio. In the example above, Alice encrypts a message with Bob's public key producing ciphertext that only Bob can decrypt with his Private key. Consider what would happen if instead, Bob used his private key to encrypt a message so that anyone could read the message by decrypting

[2]PocketAPRS for the palm pilot, has triangulation features built in for this purpose.
[3]Within the budget of an Amateur radio operator.

it with Bob's public key. In this case, we have a coded message that is public, where anyone who wants to read the message can do so and everyone knows that the message could only have originated from Bob. Bob has just created a digital signature.

A digital signature is a piece of data that is appended to a message and calculated using a person's private key. Anyone who reads the message can verify that the message originated from the proposed sender, by verifying the signature using the sender's public key. This method of "encryption" does not involve any message secrecy, since anyone can read the original message, and anyone with the public key can decrypt/verify the digital signature portion of the message.

The fact that digital signatures are not used to make a message secret is very important when used in the context of Ham radio. In Canada, Hams are governed by the Radio Communications Act. The regulations of that act state that[4] :

```
Communications with Radio Apparatus in the Amateur
Radio Service

[SOR/2000-78, s. 8]

47. A person who operates radio apparatus in the
    amateur radio service may only

(a) communicate with a radio station that operates
    in the amateur radio service;

(b) use a code or cipher that is not secret; and

(c) be engaged in communication that does not
    include the transmission of

(i) music,

(ii) commercially recorded material,

(iii) programming that originates from a
      broadcasting undertaking, or

(iv) radiocommunications in support of industrial,
     business or professional activities.
     SOR/2000-78, s. 9.
```

The wording is extremely important, note that the regulations do not state codes and ciphers can not be used, it specifically states that codes and ciphers may not be used for secrecy.

The bases for this presentation of encryption, digital signatures and the Radio Communications Act is to justify KidCQ's use of digital signatures in its customized APRS based communications protocol. KidCQ requires the transmission of digitally signed messages, and countries that allow codes and ciphers to be transmitted over the air for purposes other that data secrecy, will allow experimentation with the KidCQ protocol.

[4]Notice item 47(b)

3 Protecting Children with Smart Cards

The purpose of using a smart card to dole out activation codes is ultimately to protect children from fraudulent use of the KidCQ System. The entire system teeters on the premise that only a police officer can make an activation code that will be used to enable a device that tracks a child. Underneath the covers, this restriction is made possible by the fact that the KidCQ cryptographic private key remains a secret. If the private key is compromised, stolen, or accidentally divulged then anyone who knows the private key can generate their own valid KidCQ activation codes.

The more people that know a secret, the more likely that secret will be compromised. By using a smart card, access to the private key can be given to a police officer but the private key will never be revealed. The smart card has a small amount of internal memory and is capable of performing computations, all it requires from the outside world is a small amount of power and some input data to operate on. By performing all cryptographic signing operations on the smart card, the private key can remain safe within the card and never be divulged.

Other protective mechanisms are implemented on the smart card as well. If a user of the card enters the wrong PIN too many times, the card will consider the user a threat to the KidCQ system and destroy its copy of the private key. Also, the card is of limited use and after successfully generating a limited number of activation codes, the card will erase the private key. The justification for this action is that a legitimate police officer can easily obtain another KidCQ activation card, but a thief can not. If someone were to steal a KidCQ smart card, the damage that could be done through fraudulent use of the card is limited to the generation of 4 or 5 activation codes.

Not all smart cards are made equal. In essence, a smart card is an integrated circuit (IC) embedded in a piece of plastic, that has the form factor of a standard size credit card. The first difference people notice about a smart card is the serial interface connector that is about the size of a fingernail and can be seen on the top of the card.

Generally speaking, a smart card can be a variety of specialized circuits from a small piece of flash memory to a small computer capable of computations. However, one of its defining properties is that a smart card does not have its own power source, nor does it have a human interface. A smart card must be inserted into a reader terminal so that it can receive power and communicate with a user.

The KidCQ system uses a special type of smart card called a JavaCard. JavaCards are flash programmable and natively capable of some sophisticated mathematical routines commonly used in cryptographic algorithms. As one would suspect, you program a JavaCard using a subset of the Java programming language. This subset is defined in the JavaCard specification that is maintained by Sun Microsystems.

While Sun defines the programmer interface for JavaCards, these smartcards are manufactured and sold by companies that specialize in smartcard technologies. For example, ST microelectronics manufactures smart card ICs and will embed them in a plastic credit card package. A company like Axalto, will contract a company like ST microelectronics to mask the smart card ICs with special program code during the manufacturing process. Axalto would then sell these customized smart cards in the marketplace with it's own branding, such as "Cyberflex Access" JavaCards.

The KidCQ activation card is a Cyberflex Access JavaCard. This card natively supports all of the cryptographic operations required to generate an activation code, so programming the card is relatively easy, the only thing that must be implemented is a communication protocol and a simple state machine. All of the cryptographic math comes for free.

As an example, the following code snippet is used on the smart card to generate the activation code. All of the function calls in this code sample are part of the native JavaCard Application Programming Interface (API), meaning that no other low level crypto programming was required to generate the activation code.

```
.
.
.
// Create Sha1 digest object.
digest = MessageDigest.getInstance(
                        MessageDigest.ALG_SHA,
                        false);

// Create an RSA Private Key object.
// Fill it in later.
priv_key = (RSAPrivateKey)KeyBuilder.buildKey(
                    KeyBuilder.TYPE_RSA_PRIVATE,
                    KeyBuilder.LENGTH_RSA_1024,
                    false);

// Create an RSA signature object.
signature = Signature.getInstance(
                    Signature.ALG_RSA_SHA_PKCS1,
                    false);

// Signature object is in sign (not verify) mode
// using the specified private key.
signature.init(priv_key, Signature.MODE_SIGN);
.
.
.

private void genSig(APDU apdu) {
  byte buffer[] = apdu.getBuffer();

  // Check state of the card is not in HALT.
  if (state != STATE_NORMAL) {
    ISOException.throwIt(SW_WRONG_STATE);
  } // if

  pin.reset();

  // Check Usage Count.

  if (usage_count == 0) {
    ISOException.throwIt(SW_USAGE_EXPIRED);
  } // if

  // get the incomming user data.
  short readLen = apdu.setIncomingAndReceive();
  short offset = ISO7816.OFFSET_CDATA;

  // Validate Pin.
  if (!pin.check(buffer, offset, (byte)0x04)) {
    ISOException.throwIt(SW_WRONG_PIN);
  }
  offset += 4;

  // Inialize Activation Code Buffer with 0's
  Util.arrayFillNonAtomic(act_code, (short) 0,
                  (short) 256, (byte) 0x00);

  // Copy this Smart Card's serial number to
  // first 3 bytes of the activation code.
  Util.arrayCopy(sc_id, (short)0, act_code,
              (short)0, (short)3);

  // Copy the 5 bytes of Device Serial number
  // sent to the card by the user, to the
  // activation code buffer.
  Util.arrayCopy(buffer, (short)offset, act_code,
              (short)3, (short)5);

  // Hash the two serial numbers.
  digest.reset();
  digest.doFinal(act_code, (short) 0, (short) 8,
              digestBuffer, (short) 0);

  // Generate the signature. Append it to the
  // Activation Code.  Will be 128 bytes in size.
  try {

    signature.sign(digestBuffer, (short) 0,
              (short) 20, act_code, (short) 8);

  } catch (CryptoException e1) {
  }

  // Decrement Usage Count.
  usage_count -= 1;

  // Concatinate data and signature and return.
  apdu.setOutgoing();
  apdu.setOutgoingLength((short) (128 + 3 + 5));

  // Send the Activation Code back to the user.
  apdu.sendBytesLong(act_code, (short)0,
                  (short) (128 +3+5));

} // genSig()
```

The beginning of this code sample shows how the digest, key and signature objects are instantiated. This code is executed when the program is flashed to the smartcard. These objects maintain their state on the smart card until the program is deleted or re-flashed.

The genSig() smart card function in the sample above is called when the user attempts to generate an activation code. For example, when a user inserts the smart card into a reader and uses the Police Activation Code Generator Windows application.

From looking at the source sample, it is important to note how the activation code is actually generated. When the user asks for an activation code for a particular KidCQ device's serial number, the device serial number and the serial number

[5]The serial number of the smart card is assigned to the card at the same time as when the KidCQ private key is injected.

of the smart card^5 are placed into a buffer, the buffer is hashed and signed using the KidCQ private key. The signature is appended to the buffer and finally the buffer is returned to the user as the activation code.

This process effectively fingerprints each activation code being generated by a KidCQ smart card and so any activation code can be traced back to the police agency that owns the smart card.

The full KidCQ JavaCard program listing is approximately 80 percent larger than the code snippet that is shown. Most of the remaining code was written to handle data communications with the host PC at the other end of the smart card terminal. The rest of the program also implements the state machine for the card that is presented below in figure 5.

The smart card does not have the KidCQ private key when the program is first flashed to the card. The private key is injected into the card at a later time, and this is reflected in the smart card's state machine. The intention of this feature is to isolate the card manufacturing process from having to know the private key. By separating key injection out of this process, it leaves the possibility for some group, possibly federal police, to handle key injection and card distribution.

The state machine also shows how failed PIN attempts and card usage affects the state of the card. When either of these counters reach 0, the card enters a HALT state where the private key is erased and the card will no longer respond to code generation requests. In the prototype, each of these counter values are initialized to 5, so the user only has 5 attempts to correctly enter the PIN, and the card will only generate 5 activation codes.

4 Transmitting the Activation Code using APRS

In order to activate a child's KidCQ device, the digitally signed activation code must be delivered over the air. This is a Ham experimental project with one of the primary goals being to take advantage of existing Ham infrastructure. Naturally TNC based transmission of AX.25 packets on the 2m band was selected for a delivery method. However, a quick glance at the KidCQ protocol in the appendix will reveal that KidCQ activation messages are also wrapped in custom APRS messages.

APRS was used for a number of important reasons. The first being the wide proliferation of the protocol within the amateur community. Hams that do not care to investigate packet radio have most likely seen an APRS node in action. So it is much easier to communicate the nature of a KidCQ message by saying that it is simply a special APRS message type instead of explaining unconnected AX.25 packets.

There are good technical reasons for employing APRS as a delivery mechanism, such as the use of APRS digipeater aliases. Piggybacking KidCQ messages on the APRS protocol and infrastructure means that the benefits of packet propagation using aliases like RELAY, WIDE and IGATE can be acquired for free. Also, if KidCQ ever grows beyond a prototype system then transmission of KidCQ activation messages can easily be integrated into the numerous existing APRS software packages, taking advantage of existing protocol stack implementations.

As noted earlier, the KidCQ protocol specification is included as an appendix to this paper.

5 KidCQ Device Prototype

5.1 Hardware Design

While small size and low power consumption are not the focus of the KidCQ prototype, it was considered important to present a relatively small device with minimal power consumption using existing components from the amateur community where possible. For the purposes of experimentation and prototyping the KidCQ device is adequate but certainly not complete. Ideally, a future development project for a real KidCQ device will be to shrink the design by using surface mount parts and ICs with ultra low power consumption.

The device as it is, performs the following functions (refer to the block diagram in Appendix C figure 4:

- The receiver gets the signal and passes it to the TNC-X.
- The TNC-X takes the signal and divides it into APRS packets.
- These packets are passed on to the TNC-X's daughter board, which is a custom made for the KidCQ application.
- If the daughter board finds that the information contained in the packet is the activation code that this particular device is looking for, and the code validates, it sets a high signal on pin #2 of the pic18f252.
- The high on pin #2 turns on the beacon transmitter. This same high signal is used to turn off the receiver since the receiver and the beacon transmitter use the same antenna for their operation.

A 1 KHz tone was chosen for the the beacon transmitter since it was a frequency that could be heard by the human ear. The tone is generated by a MAX038 chip using a sine wave. A simple oscillator would have worked just as well, however the MAX038 was what was lying around the workshop and it helped save on experimentation and integration time.

5.2 TNC-X and KidCQ Daughter Board

The KidCQ child device prototype uses John Hanson's TNC-X board to transmit packets over the air. The TNC-X has been extended using the daughter board interface to include a second PIC microcontroller to process incoming packets. The second PIC is a model pic18f252 that was selected for its increased RAM and 8x8 bit multiplier.

A KidCQ digital signature is a very big integer value and it takes 128 bytes to store this value. In order to verify an RSA digital signature, this 128 byte integer value must be cubed and reduced by a modulus defined in the public key. i.e. Keep dividing the cubed number by the public key number until there is only a remainder.

So the signature is 128 bytes long, the public key is 128 bytes long, and multiplying two 128 byte numbers together will potentially result in a 256 byte number. Most of the PIC 16 series chips have less than 512 bytes of RAM. Without some very expensive and sophisticated memory paging operations, a Pic 16 series chip can not store all of the required data. Also, pic16 chips do not have a multiply operation making exponentiation of large integers extremely time consuming.

In order to handle the number crunching requirements of an RSA signature verification, the KidCQ prototype uses the more expensive Pic18 series chips, specifically the pic18f252 model that includes 1.5 kB of RAM and an 8x8 bit multiplier. This microcontroller has enough RAM to store the public key, digital signature, incoming packet and still meet the mathematical scratch space requirements of the signature verification algorithm. Also, the 8x8 bit multiply operation allows a 128 byte integer to be cubed and reduced in a reasonable amount of time. Tests during prototype development have shown that a signature can be verified on the order of 20-25 seconds.

5.3 KidCQ PIC18 Program

The KidCQ program will be described in this section with a top down approach, starting with the main routine. The program begins by waiting for a KISS packet from the TNC-X on the UART communication port of the pic18f252. As described above, there are many protocol wrappings in between the KISS message and a KidCQ activation code. Essentially the program tries to filter messages that are not KidCQ at the first opportunity that is presented while processing each protocol layer.

The following code snippet was taken from the KidCQ Pic18's program. It is written in C and the CSS PCH compiler is being used:

```
void kidcq_receive(KIDCQ_RSA_TYPE *kidCQ) {
  int8 packet[AX25_PACKET_LEN];
  int8 packet_len;
  int1 rv = 0;

  while (rv == FALSE) {
    packet_len = AX25_PACKET_LEN;
    kiss_receive(packet, &packet_len);

    rv = ax25_process(packet, &packet_len);
    if (rv == FALSE) {
      continue;
    } // if

    rv = aprs_process(packet, packet_len, kidCQ);
    if (rv == FALSE) {
      continue;
    } // if
  } // while
} // kidcq_receive()
```

Here we loop (forever), processing incoming packets and removing the KISS and AX.25 protocol wrappings. The function call tree is kept flat, meaning that each of the functions above, like ax25_process, will avoid making any further function calls with the packet and where possible, packets are processed in place to avoid "blowing the stack". Each packet is upwards of 135 bytes and so balancing code readability and memory limitations can be tricky at times. Fortunately the CSS compiler that was used in this project is a big help for managing available memory. Once a valid KidCQ message is received and parsed, the resulting data is stored in the following structure:

```
typedef struct {

  //Device serial number
  int8 cld_sn[CLD_SN_SIZE];

  //Police card serial number
  int8 police_sn[POLICE_SN_SIZE];

  //Digital signature
  int8 signature[RSA_SIZE];

} KIDCQ_RSA_TYPE;
```

This filled structure is passed back up to the main routine after one final validity check. The device serial number that was transmitted with the message is compared to the Serial Number that is statically embedded in the program[6]

Upon returning to the main routine with a KidCQ activation message, the program will next try to validate the signature using the KidCQ public key that is also statically embedded in the program code. Refer to Appendix B for a listing of the KidCQ RSA public key.

As mentioned above, the main routine uses this public key to verify the signature in the KidCQ message:

```
//NOTE: CLD stands for Child Location Device.

void main() {
  int8 message[8];
  int8 digest[20];
  KIDCQ_RSA_TYPE kidCQ;
  int1 valid = FALSE;

  fprintf(DEBUG, "started...\r\n");

  // used to filter KidCq messages for this
  // device only.
  memcpy(kidCQ.cld_sn, CLD_SN, CLD_SN_SIZE);

  while (!valid) {

    // listen to the TNC-X for a KidCq message.
    kidcq_receive(&kidCQ);

    // concatenate the serial numbers of device
    // and police smart card.
    memcpy(message, kidCQ.police_sn, 3);
    memcpy(&message[3], kidCQ.cld_sn, 5);
```

[6]Admittedly, statically embedding the device serial number in the program code does not lend well to scalability, but for the purposes of a prototype it is more convenient to re-flash the PIC than to create some special serial number provisioning application.

```
    // Hash the serial numbers, the digest will
    // be compared to that of the signature.
    sha1_final(message, 8, digest);

    // "Decrypt" the signature, and compare the
    // digest.
    valid = rsa_verify(g_M, g_m_prime,
                       g_R_cubed_mod_m,
                       kidCQ.signature,
                       digest);

  } // while

// Enable the RF beacon.
output_high(PIN_A0);

} // main()
```

First the device serial number and police smart card serial number are concatenated and fed into a hashing function. The resulting digest is fed into the RSA signature verifier along with the public key and the signature that was sent over the air.[7] The signature verifier will "decrypt" the signature using the public key, and compare the decrypted digest to the digest that was passed into the verifier from the main routine.

5.4 Signature Verification in a PIC18

An RSA signature verification is calculated by treating the digital signature as a very large integer. Cube the signature, and find the remainder after dividing by the Public Key[8]. The remainder is compared to the message being verified. If they are the same, than the signature was successfully verified.

The RSA verification engine consists of two components, the high level RSA verifier, and the big integer arithmetic unit. Both pieces of code have been built for this prototype from scratch since the need for big integer math on a PIC is not very common.

The custom big integer math library contains the following functions:

```
void multiply_1_n(int8 x, int y[], int8 y_len);
void add_n_n(int8 x[], int8 y[], int8 len);
void subtract_n_n(int8 x[], int8 y[], int8 len);
int8 compare_n_n(int8 x[], int8 y[], int8 len);

void MontgomeryProduct(int8 m[],
                       int8 x[], int8 y[],
                       int8 m_prime, int8 A[],
                       int8 buffer[], int8 len);
```

In the functions above, the parameter `len` determines how many bytes long an integer array is and so the maximum integer value would be calculated using the formula 2 ^ (8 * len). Most of these functions are standard grade school arithmetic algorithms, albeit operating in a base 256 radix. The last function MontgomeryProduct() is an implementation of Montgomery Multiplication,[9] which is a time optimized multiplication algorithm that will multiply 2 big integers, interleaving reduction steps (i.e. the modulo operation) throughout the algorithm such that the number of discrete operations required to multiply and reduce two large integer values is reduced.

```
#define LEN 129
#define KEYSIZE 128

int1 rsa_verify(int8 public_modulus[],
                int8 m_prime,
                int8 R_cubed_mod_m[],
                int8 signature[],
                int8 digest[]) {
  int8 i = 0;
  int8 A[KEYSIZE + 2];
  int8 A2[KEYSIZE + 2];
  int8 scratch_space[KEYSIZE + 1];
  int8 comp_sig;

  MontgomeryProduct(public_modulus,
                    signature,
                    signature,
                    m_prime,
                    A,
                    scratch_space,
                    LEN);

  // Shorten the A buffer for the next op.
  for(i=0;i<LEN-1;i++) {
    A[i] = A[i+2];
  } // for

  MontgomeryProduct(public_modulus,
                    A,
                    signature,
                    m_prime,
                    A2,
                    scratch_space,
                    LEN);

  // Shorten the A2 buffer for the next op.
  for(i=0;i<LEN-1;i++) {
    A2[i] = A2[i+2];
  } // for

  MontgomeryProduct(public_modulus,
                    A2,
                    R_cubed_mod_m,
                    m_prime,
                    A,
                    scratch_space,
                    LEN);

  // A should have the decoded signature.

  comp_sig = compare_n_n(&A[KEYSIZE + 2 - 20],
                         digest, 20);

  if (comp_sig == COMPARE_EQUAL){
    return TRUE;
  } else {
    return FALSE;
  } // if
} // rsa_verify()
```

6 Next Steps

The RSA algorithm is not well suited for this application. It was chosen for the KidCQ prototype because it is very easy

[7]The g_R_cubed_mod_m parameter is some static pre-computation on the public key to help speed up signature verification.
[8]Actually the modulus component of the public key.
[9]the reader is referred to "Handbook of Applied Cryptography", *Menezes et al.* for a complete description of the algorithm.

to implement. However, KidCQ will need to move away from RSA because of the following:

- Currently a 1024-bit RSA key is being used to produce a digital signature that takes up 128 bytes of space in an AX.25 packet. The signature expands further if a base-91 printable version is used in the APRS protocol. This is very large and is not scalable given KidCQ device constraints.
- An RSA key size of 4096-bits would be required to meet the requirements for a production KidCQ system. An RSA signature based on a 4096-bit key would be too large for the AX.25 protocol and current PIC technology. Over a 1200 baud TNC radio link, it is unlikely a complete error free message could be sent given the proposed KidCQ operating environment.
- The RSA signature scheme is classified as a signature scheme with "message recovery". In short this means that an implementation can easily be reversed and used as a form of encipherment. Because of this, anything that implements an RSA based signature scheme is export controlled by the Canadian and US government.

To make activation codes smaller, KidCQ will need to adopt a new digital signature scheme. Elliptic curve cryptography (ECC) is a very promising crypto-system. The ECDSA signature algorithm would also address any export issues, since the algorithm does not have message recovery and can not be used for secrecy.

Crypto-systems aside, the KidCQ prototype needs to be further tested with emphasis on field testing. Improvements must also be made to the design, to get the device as small and power conscious as possible.

7 Sources

1. Brian Neill, VA3BPN. *Direction Finding Abducted Children: Proposal For A New Amateur Radio Emergency Service*, July, 26, 2003, 22nd ARRL and TAPR Digital Communications Conference, ISBN: 0-87259-908-6, ARRL Order Number: 9086.
2. John Hanson, W2FS. *TNC-X: An Expandable Microcontroller-Based Terminal Node Controller*, 22nd ARRL and TAPR Digital Communications Conference, ISBN: 0-87259-908-6. Available at: www.tnc-x.com
3. A. Menezes, P. van Oorschot, S. Vanstone. *Handbook of Applied Cryptography*, CRC Press, Florida, 1997, ISBN: 0-8493-8523-7
4. David Kahn. *The Codebreakers, Revised Edition*, Scribner, NY, 1996, ISBN: 0-684-83130-9
5. Sun Microsystems. *Java Card Platform Specification*, Sun Microsystems, Available at: java.sun.com/products/javacard/specs.html
6. RSA Laboratories. *PKCS #1: RSA Cryptography Standard*, RSA Security, Available at www.rsasecurity.com/rsalabs/

8 Appendix A: KidCQ Protocol Specification

8.1 AX.25 APRS Frame

Refer to APRS specification, section 3.

Field	***Size***	***Value***	***Description***
Flag	1	0x7E	AX.25 Flag
Destination Address	7	APZ525	Experimental APRS version designation. This will need to be changed if system proliferates.
Source Address	7	*callsign*	6 character callsign and optional SSID of operator.
Digipeater Address	7-56	*, KIDCQ	Normal APRS digipeater list, last digipeater address must be KIDCQ.
Control Field	1	0x03	UI Frame.
Protocol ID	1	0xF0	No layer 3 protocol.
Information Field	1-256		APRS user defined information defined below.
FCS	2		Field Check Sequence.
Flag	1	0x7E	AX.25 Flag

8.2 APRS Message Type

Inside the AX.25 Information Field is an APRS User-Defined Data Object. See the APRS specification, section 18.

Field	***Size***	***Value***	***Description***
APRS Data Type ID	1	{	APRS Data Type Identifier.
User ID	1	{	Experimental User ID. Will need to be assigned a unique user id if system proliferates.
KIDCQ Message Type	1	r,a or l	KIDCQ Message types defined below.
KIDCQ Message	Max 253		Defined below.

8.2.1 KIDCQ RSA Printable Message Type (Message Type = r)

Field	*Size*	*Description*
CLD Serial Number	7	5 byte, pseudo base-91 ascii encoded, serial number.
Police Serial Number	4	3 byte, pseudo base-91 ascii encoded, Police smart card serial number that generated the activation code.
RSA Signature	158	128 byte, pseudo base-91 ascii encoded, RSA Digital Signature.

8.2.2 KIDCQ RSA Binary Message Type (Message Type = a)

Field	*Size*	*Description*
CLD Serial Number	5	Serial number of the tracking device.
Police Serial Number	3	Police smart card serial number that generated the activation code.
RSA Signature	128	RSA Digital Signature.

8.2.3 KIDCQ RSA Public Key Message Type (Message Type = l)

Transmitting a digital signature over the air could be mis-construed as data secrecy or encipherment. During experimentation, the public key used to verify the signature may not be widely available and someone who receives a signature message will not be able to verify the signature.

To ensure that KIDCQ messages are not perceived as encrypted messages during experimentation (where the public key is not widely known) it is strongly recommended that the public key is broadcast intermittently with digital signature messages. Adhering to this method will ensure that anyone who hears the KIDCQ digital signature message will be able to verify the signature with the over the air broadcasted public key.

Field	*Size*	*Description*
Exponent	1	RSA Public Key Exponent (integer value of 3 for now).
Modulus	128	RSA Modulus.

8.3 Pseudo Base-91 Ascii Encoding

The nominal method for computing the base-91 encoding of integers, according to chapter 9 of the APRS specification, is to continuously reduce the n-byte integer value modulo 91 to produce a base-91 encode byte stream.

This method works fine for small integer values, however the RAM and processing requirements for big integers is too great for a PIC microcontroller that must decode the base-91 integer encoding. So for the large integers within this message, a slightly less space efficient version of the base-91 encoding is used.

1 base-91 byte can represent a 6 bit integer. Normally a "byte" is an 8 bit integer, so a single base-91 byte is not enough to encode a single integer byte. Using 2 base-91 bytes to represent a single integer byte (similar to Hex encoding) is very wasteful on space, since an n byte integer will expand to n*2 bytes.

It turns out that two (2) base-91 bytes can represent a 13 bit integer. So for larger integers, every 13 bits will be encoded into 2 base-91 bytes. So N bytes will be encoded into ceiling(N * 16 / 13) base-91 bytes.

This will allow the PIC to decode the big integers in small pieces and avoid the time consuming big integer math.

8.3.1 Encoding Example

Encode the 3 byte integer : (hex) 0xF4 0x24 0x00, (integer) 16,000,000.

The binary representation of the integer 16,000,000 is:
111101000010010000000000
pad the left side of the binary value with 0's until size(binary int) mod 13 = 0.
00111101000010010000000000
This gives 2 13-bit integers:
0011110100001 = 1953 and 0010000000000 = 1024
Notice that 1953 * 2 ^ 13 + 1024 = 16,000,000.
And perform base 91 encoding on these 2 integer values, as described by the APRS specification in section 9.
1953 / 91 = 21 remainder 42
1024 / 91 = 11 remainder 23
So following a decimal to hex conversion, the pseudo base-91 integer value will be encoded as:
(21 + 33), (42 + 33), (11 + 33), (23 + 33) = 0x36, 0x4B, 0x2C, 0x38

8.3.2 Decoding Example

Decode the following 4 bytes into an integer value: 0x36, 0x4B, 0x2C, 0x38.

Subtracting the decimal value 33 from each of the above bytes gives the decimal values, 21, 42, 11, 23
Now convert to 2 13-bit integer values:
(21 * 91) + (42) = 1953
(11 * 91) + (23) = 1024
Finally, concatenate the bits to produce a decimal integer:
(1953 * 2 ^ 13) + (1024) = 16,000,000

9 Appendix B: KidCQ RSA Public Key

```
Exponent
0x03

Modulus
0xC2, 0xFB, 0xE3, 0xC1, 0x68, 0x6E, 0x9B, 0x49, 0xB2, 0xB1, 0x3A, 0xDA, 0x87,
0x80, 0xE8, 0x44, 0x22, 0xAB, 0xEE, 0x66, 0x94, 0x7E, 0x9D, 0xC1, 0x25, 0xC9,
0x96, 0xD5, 0xD6, 0xB5, 0xA6, 0x86, 0x0C, 0x39, 0x74, 0x81, 0xB7, 0x4E, 0x6C,
0x62, 0x13, 0x6D, 0xCA, 0xF3, 0xDB, 0x8E, 0xB5, 0x9F, 0x6C, 0x3B, 0xFC, 0x75,
0x70, 0x42, 0xE5, 0x46, 0x4C, 0x3A, 0x95, 0x4F, 0x50, 0xEA, 0x85, 0x11, 0xE5,
0x68, 0xC5, 0x69, 0x8D, 0x6D, 0xD1, 0x26, 0xE2, 0x97, 0xEA, 0x0D, 0x68, 0xEB,
0xF8, 0xEE, 0x89, 0x5A, 0x84, 0x8C, 0x36, 0xCF, 0xA5, 0xB8, 0x0E, 0xD5, 0x43,
0x17, 0xCB, 0x55, 0xE2, 0xEE, 0x77, 0x79, 0xBF, 0x8C, 0x09, 0x79, 0x65, 0xCF,
0xFC, 0xB9, 0x44, 0x3F, 0xA6, 0xD3, 0x2E, 0x41, 0x9C, 0xA0, 0xF1, 0xF4, 0x98,
0x6C, 0xCA, 0x59, 0xE0, 0x53, 0xDE, 0xC1, 0x7A, 0x64, 0xE6, 0xE5
```

10 Appendix C: KidCQ Design Diagrams

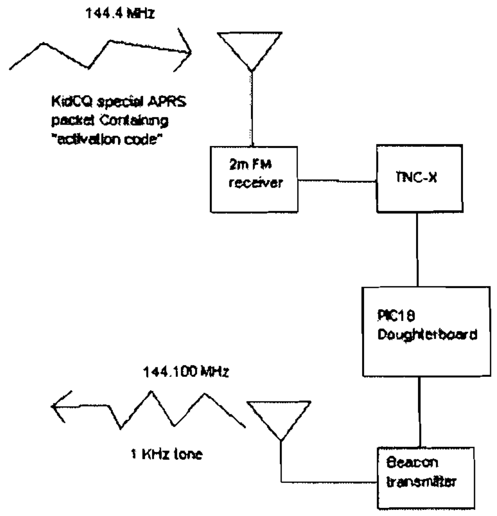


Figure 4: KidCQ Prototype Device Block Diagram

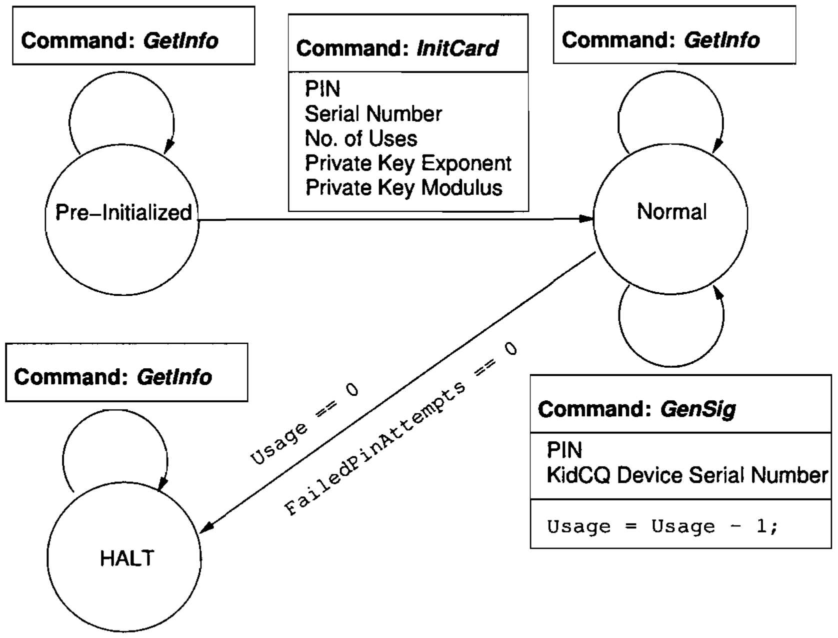


Figure 5: KidCQ Smart Card State Machine

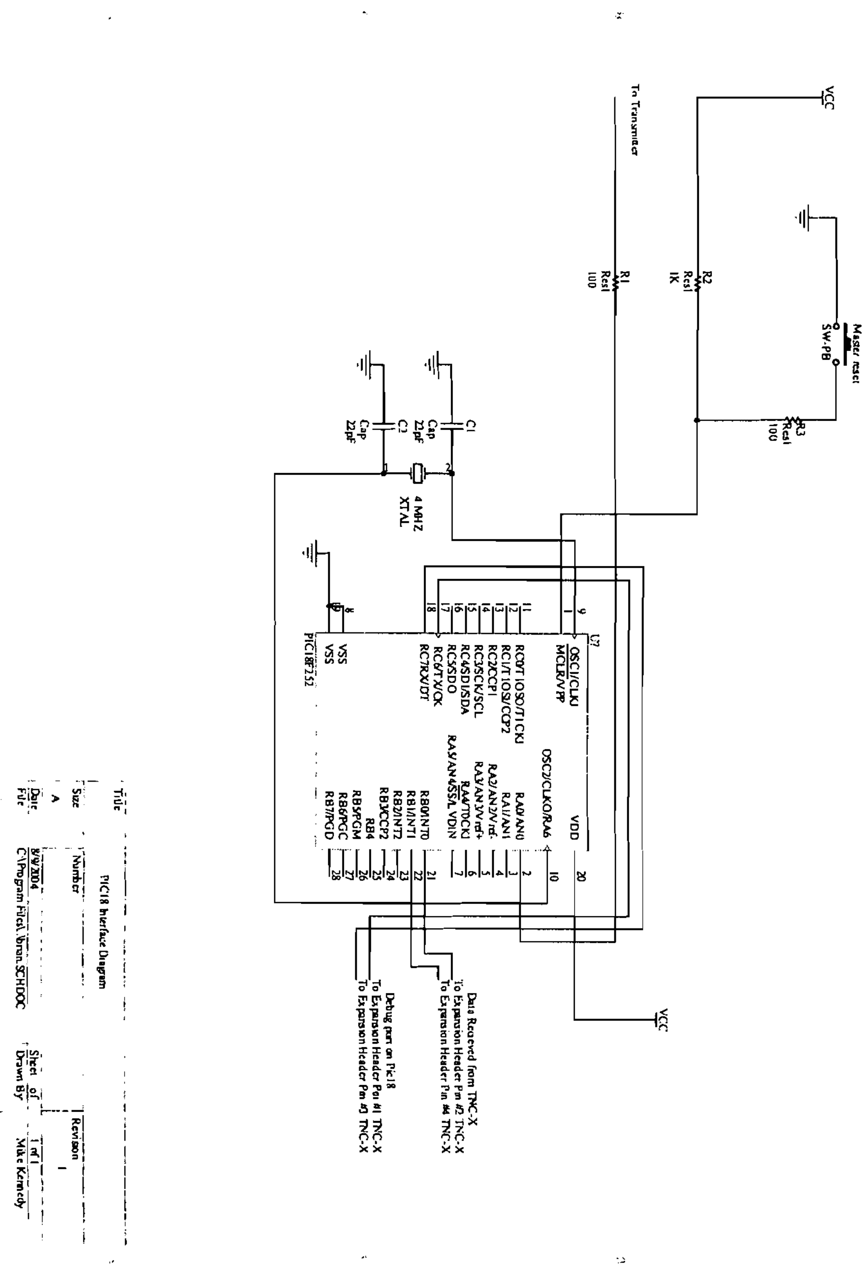


Figure 6: TNC-X KidCQ Daughter Board Design