6.115 KryptoPhone Final Project Report

Your voice for secure telecommunications(tm)

Ben Adida, Kevin Fu, Rodrigo Leroux
{ben,fubob,rodrigo}@mit.edu

December 18, 1997

1 Introduction

In this age of information, individuals must protect their legitimate personal and business transactions from eavesdroppers. Whether they intercept traffic for personal satisfaction or for international espionage, passive eavesdroppers pose a significant problem. For instance, President Clinton recently discovered his personal pager transmissions had been surreptitiously recorded, then placed on the Internet. Newt Gingrich also learned that the traditional cellular phone system is not secure. A couple amateur radio enthusiasts managed to record some politically embarrassing conversations.

Clearly we need some mechanism to ensure the privacy, authentication, and integrity of communication. The KryptoPhone significantly increases security against such attacks on the PTSN. Through the use of strong cryptography, a pair of individuals can be reasonably sure that no one besides the intended parties obtain the conversation. Moreover, each individual can be certain of the other party's identity.

We note that we came across numerous technical difficulties in implementing our system, from lack of chips, to defective chips, to overpriced adaptors. What we were able to build was not quite what we had hoped. We will first report on the theory of our system, and how it was meant to be. We will then explain the limitations we faced, and what our current prototype actually does.

2 Design Model and Goals

2.1 Trust and Threat Models

Before we decide what our goals are, we need to identify what trust and threats are present in our system. This necessitates distinguishing all the parties involved:

- The phone users: the caller, and the recipient
- The adversary/eavesdropper: anyone who wants to listen in on the conversation, or somehow change the normal way of things.

From this we can easily conclude our trust model:

- A phone user trusts his equipment. We will not deal with buggered equipment threats.
- A phone user trusts the voice he/she hears on the other end as being the one he/she recognizes. We will not deal with adversaries who know how to fake someone's voice and personality.
- No trust is put on the security of the phone line, either for authentication or privacy.

Given what we trust, we now address the issues that pose threats:

- An adversary can listen in on the communication that is going on between two given people: we will call this a passive attack.
- An adversary can listen in and modify signals on the communication that is going on between two people. The extreme case of this is a man-in-the-middle attack, where the adversary pretends to be the "other user" to each of the phone users. We will call this an active attack.

2.2 Goals

Now that we know exactly what to model our system on, and what things we need not take into consideration, we can formulate our precise goals for this system.

- Security against passive attacks
- Security against active attacks
- usability over normal phone line
- reasonably low cost (approximately $250)

3 Protocol and High-Level Design

3.1 Basic Cryptographic Needs

At a high level, we know immediately what is needed to enable us to have the security required. First, each phone needs a public/private keypair of some kind to enable the sharing of a secret key. Each phone also needs to be capable of a stream-like cipher encryption scheme (like RC4). As we will see later, each phone box finally needs to be able to perform hashing (using MD5, for example).

For practical purposes, we will be using Diffie-Hellman key exchange (because it's free), and RC4 (or alleged RC4, RC4 is supposedly a trade secret of RSA Data Security, Inc.) as our stream-cipher because it is a byte-oriented encryption scheme. Using a byte-oriented encryption scheme allows us to easily use an 8-bit microprocessor.

3.2 The protocol

The idea of our system is that encryption can be started at any point during a phone conversation:

- Two individuals are speaking on the phone, each has a kryptophone box.
- One of the two initiates the encryption by pressing the '*' key on his/her telephone. This person becomes the initiator, the other becomes the receiver.
- The initiator's box generates a random number x and sends a Diffie-Hellman public key to the receiver: $g^x \pmod{p}$.
- The receiver's box generates a random number y and sends its corresponding Diffie-Hellman public key to the initiator: $g^y \pmod{p}$.
- Each box calculates the shared secret: $g^{xy} \pmod{p}$.
- The secret is split into two keys, one for each channel (initiator to receiver, and receiver to initiator).
- Each key is used as an RC4 stream-cipher key to encrypt the two channels of voice communication.

3.3 Preventing Active Attacks

In addition to the above protocol, we want to prevent against man-in-the-middle attacks where an active attacker interferes with the communication and spoofs being the receiver where the initiator is concerned, and vice-versa.

A man-in-the-middle attack can only be prevented using some form of authentication. Since we cannot have a centralized certification authority (way too complicated), we will use voice authentication to perform this, as specified in the trust model.

What will happen is that when a box receives the other box's public key, it hashes it using MD5, and displays, on an LED display, the last few bytes of the hash. The person seeing this then asks the other person on the phone to read off his/her corresponding key hash. Since voice authentication is trusted, we know that the person speaking on the other end is indeed the rightful person in the communication. If the two numbers match, then the user of the box is sure that the public key he/she received is indeed the public key of the correct user on the other side. A man-in-the-middle attack becomes impossible.

3.4 Handshaking Protocol

Beyond the general description of the cryptographic protocol, we need a protocol that will synchronize the two kryptophones. Any error in synchronization will completely mess up the stream cipher, which depends completely on the position of the byte it is crypting.

To synchronize everything, we will use a handshaking protocol of the following kind:

- When the kryptophone boxes are turned on, they are automatically in receive state.
- When one of the boxes initiates the communication (i.e. when the user presses the '*' key), it sends out an 8 byte pattern, repeatedly, which is the equivalent of a "HELLO"
- When a box in receive mode notices the "HELLO" message on the incoming line, it responds with a "YEAH" message, another 8 byte pattern. This "YEAH" is sent only once.
- Once a box in receive mode has sent a "YEAH" message back, or once a box in initiator mode has received a "YEAH" message back, it starts sending a few identical bytes, called the PREPARE byte. After about 30 of these bytes, It sends a START byte.
- Once the START byte has been sent, encrypted transmission ensues.
- Once at least one PREPARE byte followed by a START byte has been received, encrypted reception ensues.

This ensures perfect synchronization between the boxes. Adding a key exchange in this handshaking protocol is quite simple, using the technique of PREPARE and START bytes to precisely point out the beginning of a particular sequence.

4 Hardware Design

Our hardware is basically a significant extension over Lab #5 of 6.115. We need to sample an analog signal, compress it, encrypt it, and modulate it over a phone line. Conversely, on the receiving end, we need to demodulate the signal, decrypt it, decompress it, and make it into an analog signal again.

4.1 Interface between the kryptophone and the phone

Our first task is to interface this evil primitive communication device we call the telephone.

4.1.1 Filtering

The first important thing that needs to be done is to filter the voice signal we get to make sure that we are only getting data from the appropriate spectrum (300Hz-4Khz).

The voice signal is first filtered by a 3rd Order Butterworth Low Pass Filter at a cutoff frequency of 4Khz since a phone line will not transmit anything above that. After the filtering, the signal is amplified so that it has a range of -3V to +3V.

4.1.2 Sampling

Given the highest frequency being 4Khz (the highest that we want), we need to perform our sampling at 8Khz. At each sample, we will be generating one byte of information, thus we have 64Kbits/sec of information coming through.

Assuming that we have two separate signal lines coming from the phone (a transmit and a receive), we need to:

- digitize the analog signal from the phone's transmit line.
- reconstitute the analog signal to the phone's receive line.

We could use simple A/D and D/A converters for this, like the AD7870 and AD7840. However, since we have plans to use a voice compression chip, we should instead use a μ-law codec: the MITEL MT8960 codec. This chip allows us to have full-duplex through one chip. It spits out PCM-encoded serial signals.

4.1.3 Voice Compression

We cannot afford to simply sample voice at 8khz, and send that much data over a phone line. Currently, the best modems send 56,000 bits/sec, and here we have 64,000 bits per second on each channel! Thus, we use a voice compression chip which directly plugs into the codec, and provides an 8-bit bus for reading from and writing to it using a microprocessor. We were able to obtain samples of the Qualcomm Q4401 vocoder, which, in association with the MT8960, takes an analog voice signal and compresses it down to a fixed rate of 4,800 bits/sec, which is quite an acceptable rate for modulating over a phone line.

4.1.4 Adapting to the 2-wire phone connection

The problem is that we cannot assume, as earlier, that we have separate transmit and receive lines. We need to hook into the two-wire phone connection. For this, we need to build, using transistors, a 2-4 wire converter, also called a hybrid. This hybrid will be built without sidetones (sidetones are a small echo between the transmit and receive lines to prevent callers from screaming into the phone, but we don't need them at this level).

4.1.5 Detecting the pressing of the '*' Key

We want to be able to detect when the user presses the '*' key to begin encryption. For this, we use a DTMF detector, which is able to detect the pressing of any phone key. Any key on the phone is the combination of two frequencies. By combining one of 4 given frequencies with one of 4 other given frequencies, a key is "mapped out" like in a cartesian diagram. The DTMF combined with a number of logic gates easily allows us to output a positive signal on a line to the microprocessor when the '*' key is pressed.

4.2 Processing the signal

Since we will be processing one byte going out and one byte coming in 600 times a second (4800 bits= 600 bytes per second), we can easily use an Intel 8051 to do the entire processing and encrypting of the signal.

4.2.1 The Cryptographic Algorithm

The RC4 algorithm can be coded up in assembly quite simply. As for the key exchange, it would be optimal to code that up in C. However, because the C compiler for the 8051 seems to not be working, we attempted to code up the modular exponentiation math in assembly.

Regardless, the key exchange can take quite a bit of time, but it is done only once per conversation. Each byte, though, needs to be encryptable and decryptable in 1/1200th of a second. For a 16Mhz 8051, this leaves more than 1000 instructions to encrypt one byte, which is more than enough. A very fancy version of RC4 (with extra pins to enable/disable encryption and decryption) can be coded up in assembly in 60 instructions or so. In fact, there would be enough processing power to encrypt speech directly, without speech compression.

4.2.2 Checking the Public Key

To prevent active attacks, we need to be able to display the alleged public key on an LED display. This LED display will have 8 hex digits, the first 4 of which will display the alleged public key of the initiator box, and the last 4 of which will display the alleged public key of the other box. Thus, those two numbers should match between the two boxes, a fact which can be easily verified by both parties once the encrypted mode is entered.

4.3 Interfacing the Kryptophone with the Phone Line

Now that the kryptophone box has been interfaced with the phone, and a digital signal has been obtained and encrypted, this signal needs to be sent to the other telephone box. We can't just send the data bytes on the wire, because we're not at all sure that we'll obtain a wave of the right frequency, that won't be filtered out by the phone lines. What we need to do is modulate the data like modems do.

4.3.1 The UART

The first step is to get the data into serial format. This is done using a simple UART, which takes the usual Chip Select, Read, Write signals, an 8-bit parallel data bus, and interfaces with a serial transmit and receive line. The transfer rate supported by the UART we chose (and by almost any UART these days) allows for beyond 1Mbits/sec, which is obviously more than sufficient.

4.3.2 The Modem

The modem chip we need is one made by Cermetek, which takes simple transmit and receive lines from the UART, and interfaces this mode of transmission with the tip and ring connections of the telephone system. This modem also necessitates a few control signals to be hooked up between the UART and modem chip, independently of the microprocessor. Given each channel at 4,800 bits/sec, we will use a 14,400 baud modem, which gives us ample space for later to add extra resynchronization bits in case the conversation comes out of sync.

At this point, we've fully accomplished the digitization, encryption, transmission, reception, decryption, and reconstitution of the telephone signal.

4.4 Switching from non-encrypted to encrypted

Being able to perform the full encryption process is important, but we need a smooth way to switch from the normal, unencrypted mode, to the fully encrypted, digitized mode. This is done first by detecting that the switchover needs to happen, then by actually performing the switch.

4.4.1 DTMF on send and receive lines

We want the kryptophone to switch over to encrypted mode when the user presses the '*' key, as explained above. We do this, again as explained above, using a DTMF. The idea here is that when the DTMF detects the proper '*' signal, it sets a processor control bit to 1. The processor can then carry out the switchover.

The trick is to have two DTMF's, one on the transmit line, one on the receive line. This way, we know who pressed the '*' key, and we can give each kryptophone box a different role: the initiator and the recipient. This is useful in determining which part of the shared secret will be used for which channel. Thus, we have two DTMF's per box, each hooked up to a different control pin on the microprocessor.

4.4.2 Using Relays to Switch Over

Once a signal for initiation or reception has been received by the microprocessor, the telephone signal needs to be rerouted through the digitization and modulation process. We accomplish this by splitting the line after the hybrid on the phone side, and right before the modem on the phone line side. We then hook up relays on each of the wires involved, all grouped in control so that one control bit from the microprocessor switches over from one mode to the other.

This allows us to keep a normal phone signal coming through when nothing is encrypted, completely bypassing the entire cryptographic circuit, and avoiding any possible conflict with other things on the line (i.e. maybe a computer modem is hooked up to the line trying to browse the web, and we don't want to interfere).

4.5 Mapping the Hardware

Now that we've added all of this hardware, we need to map it into 8051 memory sections. We start with Lab #5, which mapped onto the lower half of the memory, in external RAM, the A/D and D/A. What we are going to do is split up that section of memory even more, to address:

- The Voice Compression Chip
- The UART and Modem
- The LED display

Thus, we will split up the lower half of the memory in external RAM into 4 pieces. We need to add inputs to our PAL, so that the Chip Selects of the various mapped hardware can be switched depending on the first 3 bits of the address. The first bit determines RAM, or other. The other two bits work as follows:

- 00: The Voice Compression Chip
- 01: The UART and Modem
- 10: The LED display
- 11: a future piece of hardware.

5 Obstacles in Practice

While our grandiose plans were quite appealing, we were unable to complete a number of the tasks we set out to perform for this project. Most of these problems were not our fault, but we did our best regardless to get a working system.

5.1 Voice Compression Chip Problems

The voice compression chip (the Qualcomm Q4401) was shipped to us very late in the semester. When it arrived, we realized that the adapter for wire-wrapping the chip (which had 100 pins, 50 of which were labeled No Current) cost $150, and there was no way to get a sample. We then attempted to solder normal wires directly onto the chip pins. We did this first by cutting off all the unnecessary pins to give ourselves more room, then by alternating the remaining pins up and down. Unfortunately, after hours of soldering, one of the power pins fell of the chip under the weight of the wire.

Thus, we were unable to use the voice compression chip, and were forced to return to 64Kbits/sec sampling (8Khz).

5.2 Modem Chip Missing

Another problem we encountered is that Cermetek simply did not get back in touch with us concerning the modem chips they ship. Their evaluation board was extremely expensive, and they would not respond to our inquiries on getting the modem chip alone, without the extra goodies of the evaluation board.

Without the modem, we were forced to give up the actual demo on the phone line.

5.3 Telephone Elements Missing

Further problems came up when we found ourselves unable to build a working hybrid (2-4 wire converter). We got the wiring from Professor Burns at MIT, and attempted to make it work, but we never managed to split the signal correctly onto 4 wires.

5.4 Diffie-Hellman Key Exchange

Given that we were unable to code anything in C for the 8051 because of compiler problems, we attempted to code modular exponentiation in assembly. This required about 1000 lines of assembly to perform 65-byte multiplications, squaring, and modular reductions. Although the key exchange works, we could only use random keys of 1 byte. The running time of the Diffie-Hellman key exchange is directly proportional to the number of 1's in the binary representation of the random key for each party. Moreover, if the most significant 1 bit is bit i, then $O(i)$ multiplications are needed. Thus each party has a random "zeroed" out key, with the 8 LSB's random. This allows at most 255^2 different key pairs (since each party has one key). See *Applied Cryptography* or any crypto book for details. The nice thing about finite fields is that multiplication is much easier. You don't have to worry as much about carry bits.

5.5 Changes

After these problems came up, it became clear that we needed to simplify our project to make it more a proof of concept than an actual working prototype.

We simply decided to use a normal microphone and speaker as interface, and to communicate between our two kits using only the UART, and a serial link. The idea here is that we are simply missing elements from the "ends" of our communication line. Adding in a hybrid and voice compression chip on the front, and a modem chip on the back, our system would perform the encryption system on the phone line.

In our final presentation, we demo'ed a voice encryption system, which proved that the 8051 can handle stream encryption in live-mode, even without compression. We also built the voice filters for the microphones and speakers. Thus, while we were able to plan out an entire wiring for a complete kryptophone, we were only able to put together the central part of it, without the full interface to the real world telephone.

6 Conclusion

This entire project has taught us a lot about the practical aspects of building digital systems. We had grandiose plans, but we were not able to do everything we had in mind, even though we had the whole circuit mapped out. Here are some of the conclusions we've come up with, given this experience.

6.1 Price

While a number of companies will give MIT students free samples of chips without asking any questions, and repeatedly thanking us for our business (WHAT business?), the cost of using advanced chips in prototype machines remains quite high.

The problem is with adapters. Admittedly, a company actually developing a product wouldn't mind paying a few hundred dollars worth in adapters for the prototype, and then

using soldering in the final product. However, adapters are never offered as "free samples", and so we were unable to use the incredibly cool Qualcomm Q4401 chip.

It's also very difficult to actually find modem chips these days that do JUST that: modulation/demodulation. Usually, they seem to ship with more stuff, which is just not useful for us. We're hoping, though, that if this system were to go into production, the cost would be as follows:

- One Voice compression chip: $50
- One Modem chip: $50
- One Codec: $2
- One microprocessor: $10
- One UART: $5
- One EPROM chip: $2
- One RAM chip: $5
- One PAL chip: $2
- Two Hybrids: $10
- Two Phone Connectors: $10
- Two DTMF: $10
- Ten Relays: $30
- Two filters: $10
- Connectors: $10
- LED display: $20

Overall, our system's electronics would thus cost: $226. This is not unreasonable, given that such a product currently on the market costs $1000.

6.2 Telephone Electronics suck

The other important thing we learned is that the electronics of the current telephone system are just annoying. We spent a large amount of time learning about the phone system, because it is basically a 90 year-old system. Oh well, maybe one day all phones will be digital, but then our system won't be nearly as useful, since only a small piece of it will be relevant!