6.115 KryptoPhone Project Proposal

Your voice for secure telecommunications(tm)

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

November 21, 1997

"Most people call this a telephone cord. I call it an antenna." - Robert Morris, Sr.

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 embarrasing 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.

Cryptographically secure voice communication

2 Work Management

Things left to do/assign:

1. check on C compiler for 8051
2. check on linking C with assembly.
3. encryption code that works for two-way on one processor.
4. feasibility of voice compression.

Ben will work on the cryptographic protocols (Diffie-Hellman, ARC4) and investigate how one can compress voice data enough to modulate on a phone line.

Kevin is handling the initial documentation, setting up a code repository, finding a source of truly random bits, and coding a cryptographically-strong pseudo-random number generator.

Rodrigo will analyze how a phone actually works and design an appropriate analog filter, analog switches, etc...

3 Design Criteria

Because this sort of project can get extremely complicated, we want to set a certain number of goals for ourselves, and not try to achieve too much.

1. Security against passive attacks
2. User-to-user authentication
3. Security against active attacks
4. Voice compression to enable actual use of phone line
5. Low cost parts

4 Design & Implementation

4.1 Phone Line Signal

A phone line consists of only two wires, where the incoming and outgoing signals are mixed. We need to work with the two lines separately, so we will use a "hybrid", a piece of equipment which is specifically designed to split the two signals apart. Likewise, the two signals will be reconstructed into one after processing, using another hybrid.

After taking the output signal from the hybrid and before sampling it at the AD converter, we will filter it at 4KHz using a 4th order Butterworth Low Pass Filter. It is not clear at this point if we will actually be able to use phone lines to demonstrate the cryptophone. However, if we do, filtering at 4KHz will effectively separate the actual voice signal from noise since phone lines transmit data between 400 and 3400 Hz.

The cost of the filtering chip is around $2. The manufacturer is Digikey.

4.2 Data Acquisition, Transmission, and Retrieval

Once the signal from the phone line has been appropriately split into outgoing and incoming signals and passed through a reasonably good low-pass analog filter, the data needs to be converted to digital form.

Our filter cut-off is at 4Khz, so to get the full voice quality with our system, we need to sample at 8Khz (Nyquist Theorem). Given that we are using an 8-bit processor, and that 8

bits represent enough granularity for each sample, we will be using an 8-bit A-to-D converter like that AD7870. This very simply gives us one byte per sample, at the rate of 8000 samples per second.

This data then needs to be encrypted in real-time, i.e. one byte must be encrypted in $\frac{1}{8000}$'th of a second. We are left with 8000 bytes per second of encrypted data. This data needs to be modulated over the phone line. We then modulate this stream of data onto the phone line, using a standard modulation/demodulation chip. On the other end of the line, the modulated data is demodulated, then decrypted. The result is then output through a D-to-A converter, and filtered appropriately.

4.3 Cryptographic Protocols

4.3.1 The Symmetric Cipher

To encrypt this type of stream data, a stream cipher like RC4 is the best solution (or Alleged RC4, given that RC4 is a trade secret of RSA Data Security, inc.). RC4 is a symmetric stream cipher which uses a key to generate a random stream of bytes which is then XOR'ed with the input to generate the encrypted output. The decryption is simple: the same random stream is generated using the same key, and XOR'ed again with the encrypted data, which yields the original unencrypted data. RC4 is the perfect algorithm because it necessitates very little RAM and can be coded in very few machine instructions.

RC4 alone, though is not enough, because exchanging the secret RC4 key cannot be done in the clear over the phone line! Also, RC4 keys should never be reused, because two conversations encrypted with the same RC4 key can be XOR'ed together to yield comparison information between the two.

4.3.2 Key Exchange

Thus, we need a public-key cryptosystem to exchange an RC4 key. Recently, the Diffie-Helman algorithm came into the public domain with its patent expiring over the summer of 1997. This is the algorithm we will be using. It works as follows:

- The initiator of the communication selects a random prime p, making sure to also generate g, the generator of Z_p^*.
- The initiator selects a random number x from Z_p^*.
- The initiator sends $(p, g, g^x \pmod p)$.
- The recipient selects a random number y from Z_p^*.
- The recipient sends back $g^y \pmod p$.
- The secret key exchanged is $K = g^{xy}$. This secret can easily be calculated by either of the two participants in the exchange as either $(g^x)^y$ or $(g^y)^x$. An eavesdropper, however, cannot calculate K without solving the Discrete Logarithm Problem, which is considered hard.

Once both parties have exchanged the secret K, they can split that secret into two keys: one for each communication stream (incoming and outgoing). Practically speaking, K will be 1024 bits long, and each RC4 key need only be 128 bits long, so splitting the secret K is no problem.

4.3.3 Preventing Active Attacks

The problem we still have to deal with, however, is active attackers who perform "man-in-the-middle" attacks, acting basically as a relay station between the two honest participants. Such an attack can be mounted as follows:

- The attacker intercepts $(p, g, g^x \pmod p)$.
- The attacker selects a random x', and replaces the first message with $(p, g, g^{x'} \pmod p)$.
- When the recipient sends back $g^y \pmod p$, the attacker once again intercepts that, and generates a random y', sending $g^{y'} \pmod p$ to the initiator instead.
- The result is that the initiator and the attacker share a secret $K_1 = g^{xy'}$, and the attacker and recipient share a secret $K_2 = g^{x'y}$. The two involved parties are not aware of this man-in-the-middle, however. The attacker then acts like a relay station between the two, picking up the entire conversation on the way.

This attack can be prevented by ensuring that you are indeed using your friend's $g^x \pmod p$, and not the attacker. Since you can recognize your friend's voice, you can simply have that friend read off the last few bytes of the public key he sent, and check if that is the public key you indeed used. The check should happen in both directions, so that each party is sure of the security of the line. To make this check even more secure, the public keys g^x and g^y can be hashed through a collision-free hash function before reading off the last bytes of the result. This makes even harder to generate a different key that still passes this test.

4.3.4 Practical Details

Practically speaking, the key exchange using Diffie-Helman will be mostly coded in C, given that modular exponentiation is not the easiest thing to do in assembly language! The RC4 algorithm, however, can easily be coded in assembly, and actually needs to be for maximum efficiency.

4.4 Bootstrapping the Encryption

We want our box to support unencrypted communication until encryption is requested. At that point, an organized key exchange must happen. We distinguish the two boxes in the following way:

- The initiator: starts the key exchange

- The receiver: receives the first key, and generates its key appropriately.

To start the whole encryption process, one of the two individuals presses the '*' key on his or her phone.

Inside the box, there is a frequency detector on the transmit line (after the hybrid splits up the two lines). This frequency detector sends a signal to the processor. This signal indicates to the processor that it is now *the initiator*.

On the other end, the '*' frequencies are also heard, but this time on the receive line. Another frequency detector then signals to the processor that it is now *the receiver*.

A second or so after a processor has been assigned its role, it switches the signal lines so that instead of going directly from the phone to phone line outlet, they go through the processor encryption/decryption process. Both processors then mute the phones so that they can perform the key exchange. The initiator then sends its Diffie-Helman key, to which the receiver replies with its key, and the key exchange happens. Once the key exchange has happened, the phones are unmuted, and encrypted conversation ensues.

This bootstrapping system requires two components:

- a touch-tone frequency detector that detects '*'.
- an analog switch that can control which of two signals is chosen, given a digital bit.

4.5 Circuit Schematic

4.5.1 Circuit Description and Component Selection

The circuit we will be using for our system will be very similar to the circuit we used for lab #5. We will be using an 8051 because the tools are easily available, and because it provides the right amount of power for a reasonable price.

To obtain the signal from the phone line, we need an A-to-D converter like that AD7870. To put the signal back onto the phone line (towards the phone), we'll need a D-to-A converter like the AD7840.

We also need a communication chipset that will take the digital encrypted data and communicate it to the other box. Hopefully, this will be a modem chipset, which will modulate the data onto the phone line. However, if it turns out that voice compression is too difficult to achieve, we will simply use a serial link.

We also need an LED display to show the last few bits of the public key assumed for the other person's box. This allows authentication of the public keys.

Both the communication chipset and LED display need to be mapped to memory addresses. Very much like in lab #5, we will simply split up the external memory lower half of the address space:

- Addresses starting with 000 will access the ADC and DAC.
- Addresses starting with 001 will access the communication chipset.
- Addresses starting with 010 will access the LED display.

Thus, the rest of the connections are exactly like Lab #5, with extra input and control bits to and from the PAL to adjust for the extra memory-mapped components.

4.5.2 Schematic

see next pages.

5 Known Problems (features)

The KryptoPhone does not intend to prevent phone bugging, social engineering, and rubber-hose type methods. The KryptoPhone ensures with a high probability that no one can tap your phone line. One should still take adequate precautions to prevent equipment tampering, signal broadcasting (eg, cordless phones), etc. You should not use this phone to communicate military secrets, unless you work for the other side.

5.1 Voice Compression

With our current sampling granularity and rate, we are generating 64Kbits/sec of data for each direction of conversation. There is no known way to modulate 128Kbits/sec of data for duplex conversation on a normal phone line. Thus, if we want to actually have our box work on a phone line, we need to compress the voice before it is encrypted.

We have a couple options which we are currently investigating:

- a Voice-Compression chip. Qualcomm makes them, for example. This is good because once the chip is there, voice compression is automatically done. However, such chips are expensive, and quite complex (200 pins or so).
- a DSP programmed to perform voice compression. This is a cheaper solution cost-wise, but time-wise demands a lot more investment.

Our communication chipset will depend on whether we manage to solve this issue or not. If we do solve it, we'll use a modem chipset. If we don't, we'll use serial communication.