The Use of Encryption in Embedded Systems

Charles Oram

Cardax
PEC (NZ) Ltd., Marton
New Zealand
E-mail: charleso@pec.co.nz

Abstract

It is increasingly common to find embedded systems that must communicate securely with other devices or provide secure storage of private information – such as smart cards or devices that can connect to the Internet. This has increased the need to implement secure and efficient encryption schemes in embedded systems – a task that is complicated by the minimal memory and processor resources typically available. This paper discusses some of the challenges involved in implementing encryption in embedded systems, and how these challenges can be met in practice, taking examples from the implementation of an access control and security monitoring system.

***Keywords**: embedded systems, encryption, security.*

1. Applications of encryption in embedded systems

Encryption may be used in an embedded system for a number of reasons; some of the possible uses of encryption in embedded systems are described below.

1.1 Secure communications channels.

In many systems it is important that communications links are secure, such that it is not possible for a potential intruder to inject messages into the system or observe sensitive information. For example EFTPOS systems use a secure communications channel to communicate with the bank's host computer over the EFTPOS network. An increasing number of devices are appearing that connect to open networks, or the Internet. In many cases these devices must be capable of exchanging data securely with a remote system such as a personal computer (PC).

1.2 Authentication of devices or data

With remote devices communicating on an open network it may be necessary to authenticate the devices to ensure that you are only sent commands and data from a genuine device. It is sometimes necessary to prove that stored information is genuine, i.e. has not been modified and was acquired at a specific time and place. For example, an image taken by a security camera may be used as court evidence, so it must be possible to prove that the image was captured at a certain time and has not been digitally altered.

1.3 Securing data

Devices that store valuable data need to encrypt that data to protect it. For example, Smart cards rely on encryption to store digital cash or personal details.

1.4 Intellectual property protection

In some cases it may be necessary to protect the software in an embedded system from being read or copied. One way of protecting the software is to store it in an encrypted form. This may be to protect the manufacturer's intellectual property (for example a unique algorithm implemented in software) or to protect the integrity of the system (for example an EFTPOS PINpad may encrypt its software to prevent the software being studied in an attempt to find weaknesses in the implementation). One solution to this problem is to use a microcontroller with on-chip encrypted program storage, such as the Dallas DS5002FP Secure Microprocessor Chip.

1.5 Software licensing

Software piracy has traditionally not been seen as a large threat to embedded software developers since the software usually only runs on the manufacturer's specific hardware and is purchased as one unit with the software and hardware included. Today, however, it is not uncommon for manufacturers to provide software upgrades or additional features at an extra cost. Many embedded systems either have socketed EPROMs, or support software downloading into Flash memory or battery backed memory, making it simple for a customer to make copies of the software. Secure software licensing schemes attempt to make it impossible for software to be copied from one system and run on other unlicensed systems. Most schemes use some form of encryption to prevent the licensing scheme from being defeated.

2. The challenges of implementing encryption in embedded systems

Implementing encryption in embedded systems presents a number of challenges, most of which are due to the minimal processor and memory resources typically available to embedded systems.

2.1 Speed

When encrypting data for a communications channel (for example), the processor must be able to encrypt the data fast enough to keep up with the channel. Embedded systems tend to use cheap microcontrollers to keep the cost and complexity down making fast encryption a challenge.

Strong encryption schemes, such as public key encryption, tend to be processor intensive. This means that the encryption scheme implemented must often be a compromise between speed, strength and cost. Schemes for speeding up public key encryption often require a trade-off between speed and memory usage. For example, using addition chaining in exponentiation can reduce the number of multiplications required at the expense of having to store a precomputed table in memory [1].

2.2 Memory

Implementing encryption in an embedded system will inevitably increase the amount of memory required by the system. Even with simple encryption it may be necessary to buffer both the unencrypted message and the encrypted message as you generate it, doubling the amount of memory required. The long integer mathematics involved in some encryption schemes (particularly public key encryption) can also use a considerable amount of memory.

2.3 Key exchange and authentication

How do you start up the encryption scheme with secure keys, and how can you guarantee that the system communicating at the end of a link is the device it says it is? Embedded systems often run stand-alone with no operator to type in a password, so authenticating a device is not simple. Once exchanged, encryption keys must be stored securely. In some cases the device can generate the keys randomly and store them without ever needing to reveal them outside of the device.

2.4 Random number generation

Many encryption schemes rely on using a good random number generator and are significantly weakened if random numbers are not used. Some microcontrollers provide built in hardware random number generators (e.g. Dallas DS5002FP and Motorola MSC0501).

2.5 Is it good enough?

How do you gauge whether the encryption scheme you have implemented is satisfactory for the application? This is a challenge for any software system that uses encryption, not just embedded systems. However, the restricted resources available to the typical embedded system make this more of a challenge in embedded systems.

Most practical encryption schemes rely on difficult mathematics and the fact that it takes a long time (or a huge amount of computing resources) to solve the mathematical problems necessary to break the encryption. This means that most schemes can be broken given enough time and/or computing resources. Strong encryption schemes require more computing resources and development time to implement them, so the end result is always a compromise between encryption strength, product cost and time to market.

Consider the following points when assessing whether the encryption you are using is adequate:

1. Compare the cost involved in defeating your system against the cost of the property or data that it protects. For example, if I need a $100,000 computer to defeat a system that is protecting data or property worth $20,000 then it may not be worth my while (unless I can borrow the $100,000 computer!).

2. Compare the cost of defeating the encryption versus the cost of defeating the system in other ways. For example why bother spending hundreds of thousands of dollars on computers and equipment to defeat an access control system when you could bribe the security guard for a fraction of the cost?

3. Consider the lifetime of the encrypted data. When encrypting communications messages the data may only need to remain secure for a few minutes or seconds (especially if encryption keys are changed regularly). The encryption scheme must keep the data secure for longer than the data's lifetime. Also consider the lifetime of product – technology is moving fast, so what is a secure encryption scheme today may not be secure tomorrow.

4. Be clear on why you want to implement encryption – is it an essential part of the product (such as in a smart card), or is it just to have another feature for your marketing staff to advertise?

5. Consider the cost to your organisation if the encryption is defeated. You may do better to implement a weak encryption scheme and admit that it is weak and comes with no guarantees rather than implement a strong scheme that people rely on.

6. Beware of *snake oil* [2]. Stick to established encryption algorithms and protocols. Cryptanalysts have spent many hours analysing encryption schemes such as Data Encryption Standard (DES) [4] and RSA [5], so their strength is well known. Don't believe that you can roll your own scheme that is better.

7. Don't rely on your encryption protocol being proprietary and secret [3].

8. Use large keys [3].

3. Case studies

The following sections describe how some of the above challenges have been met in the implementation of a security and access control system. The system comprises of the following devices:

- A head-end PC that is used to configure and monitor the system.

- A number of controllers that connect to the head-end PC over an Ethernet network using TCP/IP. The controller has an Intel 386EX processor with 4MB of RAM and 8MB of Flash memory.

- A number of field devices, including card readers and I/O boards, that connect to the controllers over a fast RS-485 link. These field devices mostly use a Dallas DS87C520 microcontroller.

- A digital camera (one of the field devices) that also connects to a controller over the fast RS-485 link. The camera uses a Motorola DSP56303 Digital Signal Processor (DSP).

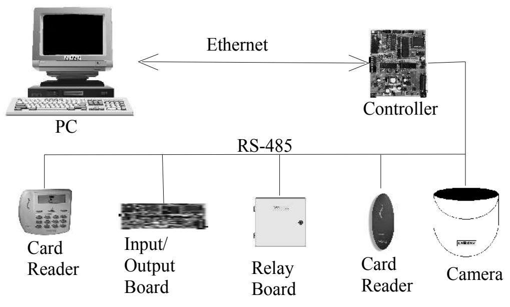

Figure 1: Security and access control system.

3.1 Encrypted TCP/IP communications for a networked controller

3.1.1 The requirements

All communications between controllers and between controllers and the head-end PC must be encrypted to protect against –

- Unauthorised viewing of private information (such as the details of a cardholder),
- Unauthorised injection of commands, e.g. a request to unlock a door,
- Unauthorised viewing of system activity, e.g. a cardholder access event, since this reveals an individual's location.

All communications between controllers and between controllers and the head-end PC must be authenticated to ensure that only authorised controllers and PCs are exchanging commands.

3.1.2 The challenges

Speed - The encryption and authentication of the communications channels must not unduly affect the performance of the system. In particular, a controller must be capable of reporting events to the head-end PC at a rate of 10 events/second.

Key exchange and authentication – It is important to exchange keys securely and to know when a controller receives an "unlock door" message that it has come from a genuine source.

3.1.3 How the challenges were met

The controller encrypts all data with a private session key. An authentication protocol allows the session key to be exchanged and ensures that only authorised controllers and head-end PCs exchange information. The encryption algorithm used is Blowfish [6], which was chosen as a good compromise between speed and security. DES was initially chosen, but replaced with Blowfish because it is faster. A publicly available C implementation of Blowfish was used.

Key exchange when a new controller is installed, is only allowed when a DIP switch on the controller is set to the *insecure/initialise* position, and when the controller is set to the initialising state at the PC. This allows the initial keys to be exchanged at a point when the installer can guarantee that both the PC and the controller involved are genuine. For higher security this initialisation could be done with the PC and controller connected via a direct serial connection.

3.2 Encrypted RS-485 communications for a microcontroller based field device

3.1.1 The requirements

All communications between the field devices and the controller, over the RS-485 link, must be encrypted to protect against –

- Unauthorised viewing of private information, such as a Personal Identification Number (PIN) entered at a card reader.
- Unauthorised injection of commands, e.g. a request to switch on a relay that unlocks a door.

The communications protocol used on the RS-485 link is a polled master-slave protocol where each device is polled ten times a second. The link runs at 187.5kbps, and a field device must only reply to polls that that are correctly encrypted. A very short reply timeout is used to ensure maximum use of the communications link.

3.1.2 The challenges

Speed - The major challenge is that the field device must decrypt the received poll as it is received. The device is not allowed to respond to a poll unless the poll is correctly encrypted, so it must decrypt the poll before deciding whether to send out the reply. The very short reply timeout means that the device cannot wait until it has the complete poll to decrypt it. When a poll is being received a new byte arrives every 43 microseconds, and a significant amount of this time is used by the communications interrupt in reading the byte from the communications chip.

Memory – Most of the field devices have only 1k Bytes of RAM.

Random number generation – A random number is generated to use as the encryption key, so a good random number generator is important.

3.1.3 How the challenges were met

A simple stream cipher was used where a pseudo random sequence is XORed with the data. The starting point of the pseudo random sequence is changed with every poll and the parameters of the pseudo random sequence are changed regularly.

The a pseudo random sequence is generated using an 8 bit Linear Congruential Generator [3] as follows -

$$x = (M*y + C) \text{ MOD } Z \qquad (1)$$

where x is the next byte of the pseudo random sequence, y is the value of the current byte in the pseudo random sequence and Z is 256 (to simplify the arithmetic). M, C and Z must be mutually prime (i.e. no common factors).

The bytes are encrypted or decrypted in the communications interrupt service routine as they are sent or received. This means that the complete poll is decrypted within microseconds of receiving the last byte. It is not strictly necessary to encrypt the reply as it is being sent, since this could be done before the poll is received, but doing so simplifies the protocol and the implementation.

Speed – The encryption algorithm can be implemented in 6 instructions on an 8051 microcontroller, taking only 2.3 microseconds on a 24MHz Dallas DS87C520 and adding approximately 6% to the communications interrupt service routine. The following code example shows what is required –

```
; Encrypt the byte in R1
mov     A, crypt_y
mov     B, crypt_M
mul     AB
add     A, crypt_C
mov     crypt_y, A
xrl     A, R1
```

Memory – Because a stream cipher is used and the encryption is performed as each byte is sent or received the encryption scheme uses only about 3 extra bytes of memory (required to store the previous value in the pseudo random sequence and the parameters currently being used for the sequence).

Random number generation – The "encryption key" used in each poll or reply to start off the pseudo random sequence is a random number generated by the field device. This number can be generated in a number of ways, depending on the components available on the field device. Most of the devices simply use one of the free running timers on the microcontroller to obtain a pseudo random number. Care must be taken with this approach because if the timer is read at exactly the same time after initialising the timer it will always provide the same number. Some of the field devices use an analogue to digital converter (ADC) to monitor power supply or input signals. The values read from the ADC will have a degree of randomness based on –

- The tolerances of the components on the board, particularly those used in generating the reference voltage.
- The current state of the input being monitored. In the case of an Input/Output (I/O) board this will also vary with the length of the cable to the sensor and the tolerance of the balancing resistors used with the sensor.
- The tolerance of the ADC.

This means that one (or several) of the ADC channels may be used as a source of a random number or as a seed for a random number generator.

3.3 Digital image authentication and tamper protection on a DSP

3.1.1 The requirements

The Cardax digital camera captures, compresses and stores images at 4 frames per second (fps). When an alarm occurs the controller can request stored and future images from the camera; these images are then transmitted to the head-end PC for storing on hard disk so that a record of what happened immediately before and after the alarm is recorded. The images are stored as binary data in a file and are therefore susceptible to tampering. For the images to be used as court evidence they must have some form of tamper protection and authentication. We call this an Image Authentication Watermark (IAW). The IAW proves that –

- The image has not been digitally altered.
- The image was taken at a specific time, by a specific camera connected to a specific system.

When viewing the images on the head-end PC it must be possible to verify the IAW of an image.

3.1.2 The challenges

Speed - An image must be compressed and watermarked at the rate of 4 fps. Compressing the image takes 150 ms, leaving 100 ms to apply the IAW.

3.1.3 How we plan to meet the challenge

Work on implementing the IAW in the camera is not yet completed, so this section discusses how we plan to implement the scheme.

Two options are being considered; both options involve encrypting information about the image (camera sequence number, time of day etc.) and dispersing this information throughout the image. Each camera has a unique sequence number provided by a Dallas DS2401 Silicon Serial Number chip.

The less preferred option uses private key encryption to encrypt the information. The weakness of this scheme is that private key is needed to verify the IAW, and so must be stored (encrypted) on the head-end PC. If the private key is revealed it could be used to remove the IAW, modify the image and reapply the IAW. The advantage of this scheme is that private key encryption will be fast enough.

The preferred option uses public key encryption to generate a digital signature of the information. This scheme is better because the IAW can be verified using only the public key, which can be freely revealed because it cannot be used to apply a new watermark. Unfortunately this scheme requires a considerable amount of processing to calculate the IAW.

To assess whether the public key scheme can be used we need to estimate whether the DSP has the processing power to calculate the public key signature in less than 100ms. The processing requires performing multiplication and exponentiation of large (528 bit) integers using modular arithmetic. Calculating public key signature, for the scheme we are using, requires approximately 332000 multiplications. A multiply takes 1 cycle (12.5ns, if the data is in internal memory) on an 80MHz Motorola DSP56300, so 332000 multiplications would take 4.15ms.

In comparison, a 100MHz Pentium takes about 17ms to calculate the public key signature. Considering that it should be possible to optimise the implementation to suit the architecture of the DSP it seems likely that the DSP should be able to perform the calculation in around 40ms.

The core of a DSP is the MAC (Multiply Accumulate) instruction. This feature, and the fact that the DSP56300 has two 48-bit accumulators, can be used to implement fast multiplication of large integers [7]. There are a variety of methods for efficiently implementing multiplication and exponentiation [1] with smart card applications promoting research in this area [8] – this gives us confidence that we can implement the scheme to meet the timing constraints.

4. Conclusions and future work

Successful use of encryption in embedded systems can be achieved if consideration is given to working within the resource constraints of embedded systems, and adequate solutions are chosen carefully.

The work on implementing the IAW on the digital camera is not yet complete, and it has yet to be proven whether the system is capable of generating an IAW using public key cryptography. An optimised implementation of the digital signature algorithm will be developed and timed to ensure that it takes much less than 100ms.

Microprocessors and microcontrollers (such as the Analog Devices ADSP-2141L and Motorola MSC0501) with on-chip hardware support for encryption are beginning to appear, driven by the need for products such as smart cards and encrypting routers. These types of chips will allow developers to use more sophisticated encryption in embedded systems and free them from some of the constraints discussed in this paper.

5. Acknowledgements

The author wishes to thank Peter Burgess, Wolfgang Zdrenka and Felix Collins of PEC (NZ) Ltd. for their assistance.

6. References

[1] A. Menezes, P. Oorschot and S. Vanstone, Handbook of Applied Cryptography, Boca Raton: CRC Press, 1996.

[2] C. Curtin, The Snake Oil FAQ, http://210.15.255.14/snake-oil-faq.html

[3] B.Schneier, Applied Cryptography, 2nd ed. New York: Wiley, 1996.

[4] National Bureau of Standards, Data Encryption Standard, U.S. Department of Commerce, FIPS Publication 46, Jan 1977.

[5] R.L. Rivest, A. Shamir, and L.M. Adleman. “A method for obtaining digital signatures and public-key cryptosystems”, Communications of the ACM, 21(2): pp.120-126, February 1978.

[6] The Blowfish Encryption Algorithm, http://www.counterpane.com/blowfish.html

[7] M. Wiener, “Processor method of multiplying large numbers”, United States Patent number 5,121,431, June 1992

[8] J. Dhem, “Design of an efficient public-key cryptography library for RISC-based smart cards”, PhD thesis, Belgium: Universite Catholike de Louvein, May 1998