Attacks on and Countermeasures for USB Hardware Token Devices[*]

Kingpin

@stake, Inc.
196 Broadway, Cambridge, MA 02139, USA.
http://www.atstake.com
E-mail: kingpin@atstake.com

Abstract. This paper presents the methods used to attempt access to private data stored in Universal Serial Bus (USB) hardware tokens without having legitimate credentials. We look at the current state-of-the-art products of the commercial world. Our research is based on an approach of using only common, off-the-shelf tools, yet we still succeed in defeating the security features and gaining access to private data. The focus is on three major attack categories: Mechanical, Electrical, and Software. We also examine other areas that may be susceptible to attack. Countermeasures and design changes that will enhance the security of such devices are proposed.

Academic papers describing the design of secure hardware devices have existed for over two decades [1, 2, 3]. Applying these approaches into real-world products, however, requires an understanding of the threat model and security envelope of the product being designed. Improper implementation can lead to avenues of attack as described in this paper.

Keywords: security, attack methods, countermeasures, analysis, hardware token, USB, secure hardware design

1 Introduction

This paper documents investigations into the mechanical, electrical, and software design of multiple vendors' USB hardware tokens (Figure 1). The main focus of the analysis is to access a legitimate user's private data on the key without having the proper credentials. We strive to educate both users and vendors about the need to implement forward-thinking security measures into their products and how to analyze a product to compare its perceived security to its actual security. It in no way describes all possible attack methods.

USB hardware tokens are a new breed of portable devices that are being used in security and public key application spaces and are beginning to attract attention by the computer security industry. They perform as authentication devices and store private keys, passwords, or electronic certificates in a

[*] This paper has been published by Reykjavik University in the *Proceedings of the Fifth Nordic Workshop on Secure IT Systems Encouraging Co-operation*, Reykjavik, Iceland, October 12-13, 2000, pp 35-57, ISBN 99799483-0-2.

hardware token the size of a house key. USB keys make use of two-factor authentication in order to grant access to the private data stored within the key. Using the legitimate user's Personal Identification Number (PIN) or password, access to the key's data will be granted.

Section 2 examines attacks related to the mechanical design of the products. This involves manipulation of the physical device housing in order to gain access to the circuit board and other device internals. Section 3 details the invasive electrical attacks which require access to the device's electronic circuitry. Section 4 details the non-invasive software attacks in which one could examine the communication channels between the USB device and host computer. In each section, design changes and preventative measures that will enhance the security and reduce the risk of unauthorized attacks are discussed.

The following devices have been analyzed:

- **Aladdin Knowledge Systems' eToken[1] 3.3.3.x.** [4]
- **Rainbow Technologies' iKey[2] 1000.** [5]
- **Rainbow Technologies' iKey 2000.** [6]

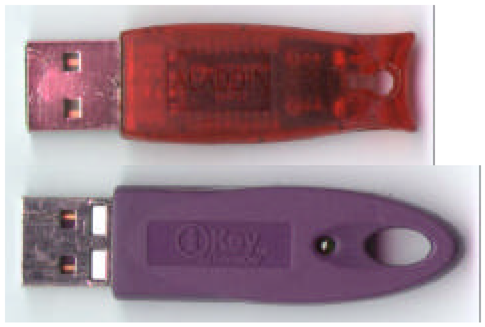


Figure 1: Aladdin Knowledge Systems' eToken 3.3.3.x, top, and Rainbow Technologies' iKey 1000 and 2000, bottom.

2 Mechanical Attack

The main goal of mechanical attacks is to gain access to the product internals. Tamper-proofing features are often designed into products to prevent or detect invasive attacks [1, 2, 3].

To further the electrical attacks (Section 3), physical, invasive access to the USB key circuit board is required. In all devices that were examined, no obvious attempts at tamper-proofing were evident and it was possible to open

[1] eToken™ is a trademark of Aladdin Knowledge Systems Ltd.

[2] iKey™ is a trademark of Rainbow Technologies, Inc. Rainbow Technologies has stated that iKey 1000 devices created after November 1999 have been modified to mitigate some of the attacks described in this paper. We have not verified this claim.

the physical housing and gain access to the printed circuit board without any signs of forced entry.

2.1 Case Study of eToken

The eToken device uses glue to keep the two pieces of the plastic housing together making disassembly moderately difficult. However, the glue can be weakened by heating the housing and careful prying will open the device.

The complete disassembly and reassembly process can be performed in under 30 minutes (Figure 2). A highly practiced attacker could complete the process in under 10 minutes. The only tools required for the attack are a hobby knife, heat gun, glue, and pliers.

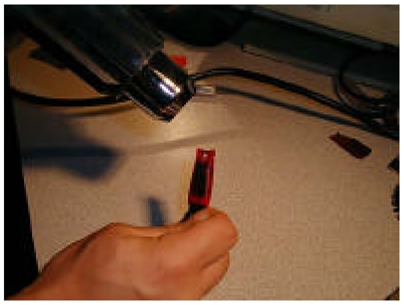


Figure 2a: Heating the key's tail to soften the glue.

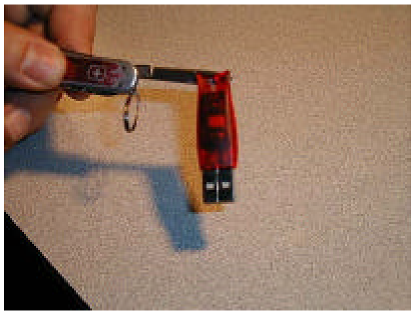


Figure 2b: Using a knife to pry open the tail of the housing. This gives an entry point to pry open the side of the housing.

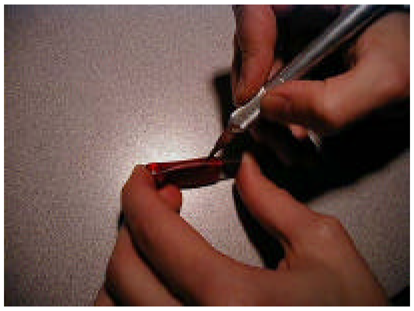


Figure 2c: After more heating, scoring and prying of the side.

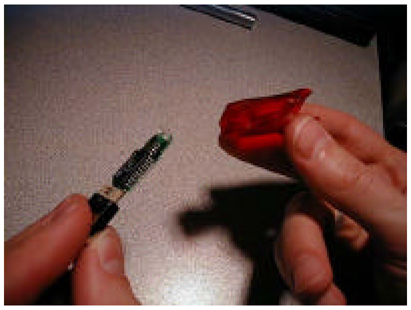


Figure 2d: Successful penetration of the housing and easy removal of the key's circuit board.

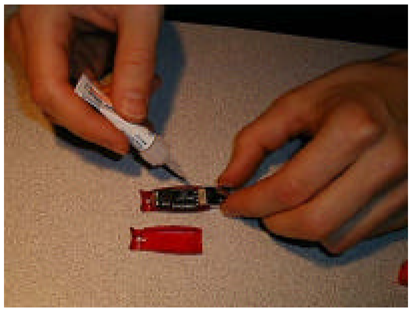


Figure 2e: After the electrical attack is complete, applying glue to both sides of plastic housing.

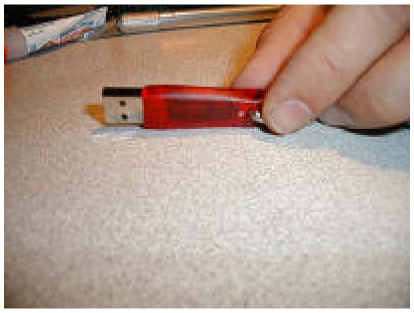


Figure 2f: Top view of reassembled key. No evidence of tampering is visible.

2.2 Case Study of iKey 1000, 2000

The iKey devices use "socket and post"-type mechanical features press-fit into each other to keep the two pieces of the plastic housing together (Figure 3). The metal housing of the USB connector assists by serving as a clamp. Removing the metal USB connector housing and prying carefully with a hobby knife was sufficient to open the device.

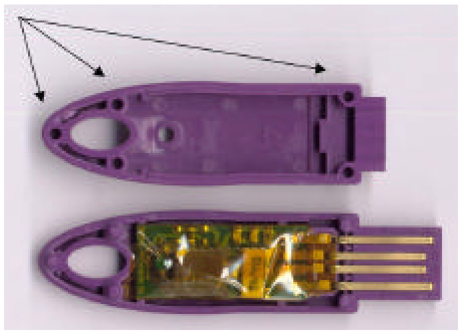


Figure 3: iKey plastic device housing with arrows denoting "socket and post" features.

The opening of both iKey devices is simple and can be performed in under 30 seconds with household tools (Figure 4). Since no cutting or scoring is required to loosen the housing, the chance of visible damage to the housing is minimal.

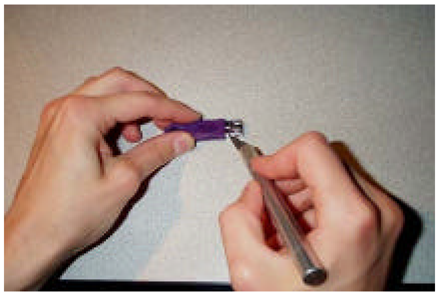


Figure 4a: Lifting the four tabs of the metal USB housing.

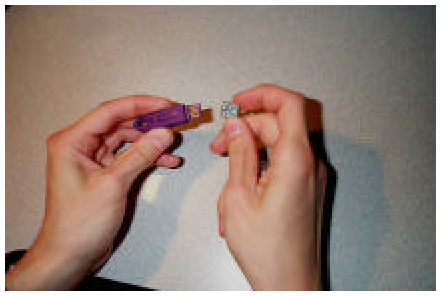


Figure 4b: Removal of the metal USB housing.

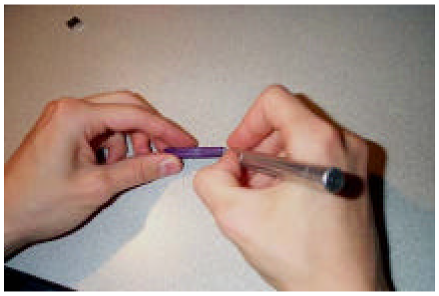


Figure 4c: Prying of the plastic housing.

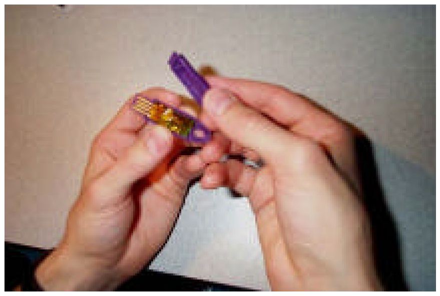


Figure 4d: Successful opening of the housing.

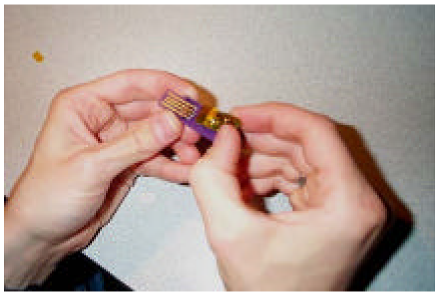


Figure 4e: Removing the key's circuit board from the 4 "fingers" of the USB connector.

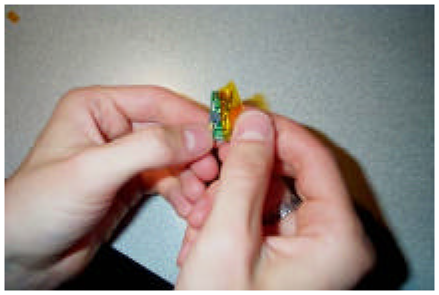


Figure 4f: Removal of tape wrapped around the circuit board. This may be a tamper-detection attempt, but the lack of tape is not visible when the device is properly closed.

2.3 Mechanical Device Variances Between eToken and iKey

"Cost versus risk" management plays a large role in security product design. Depending on what subsystems the vendor is trying to protect, various security features will be designed into the product. The investigation of the two USB key vendors yields the following matrix:

Device	Difficulty To Penetrate Housing	Protection of Internal Circuitry
eToken 3.3.3.x	Moderate	None
iKey 1000	Easy	Moderate (Epoxy)
iKey 2000	Easy	Moderate (Chip-on-Board)

The two vendors took different routes in addressing mechanical security. The eToken has a mechanical design which makes opening the device moderately difficult, while the iKey has no physical protection. However, once the eToken was opened, its circuitry was completely unprotected making probing and analysis trivial, while the iKey uses epoxy encapsulation and obfuscates part numbers to deter attackers.

2.4 Mechanical Design Recommendations

The design of secure hardware devices is a topic in itself. However, a few key points can be made related directly to the mechanical design of portable devices that are especially prone to tampering and attack such as USB hardware tokens:

- **Strong, High-Temperature Glues** with a very high softening/melting point increase the security of the physical housing and serve as a tamper-evidence feature. Many glues used by today's product manufacturers soften under direct heat, which aids in housing disassembly.
- **Mechanical Features** such as snap-fits and one-piece designs increase the security of the physical housing. They not only strengthen the design by serving as reinforcements or structural supports, but could also make the housings difficult to re-open after manufacturing has been completed. A wide variety of mechanical features can be put into the design, depending on the product type and the goal of the protection.
- **Obfuscation of Part Numbers** makes reverse engineering of the circuitry more difficult, but does not eliminate the possibility.

High-temperature glues and mechanical features are recommended for plastic housings that "snap fit" together. With the eToken, heating the key evenly softened the glue enough to gently cut open. The use of high-temperature glue would have increased the security here. There were two mechanical features holding the two plastic pieces together. Even if those are broken during disassembly, they are only visible from the inside of the housing and no anomalies are seen when the device is glued back together.

3 Electrical Attack

The electrical attacks mounted against the USB keys require physical access to the device's circuit boards. The primary goal was to attempt to access the private data, which is supposed to be protected by the legitimate user's PIN number or password, without detection by the legitimate user. External memory components, which are non-volatile storage areas that can be read and written with low-cost or homebrew device programmers, were targeted for attack.

The electrical design of both USB keys is standard and simple, consisting of a microprocessor with USB support, external memory, and "glue" circuitry. Currently, there are few microprocessors available that have internal USB support, which leads to many similarities in the electrical designs of USB devices.

A design flaw common to the USB keys examined is the improper storage of password values, which can allow the extraction of all data, including private information, from the key [7, 8]. Changing the password value which is stored in an external Serial Electrically Erasable Programmable Read-Only Memory (EEPROM) will allow access to the device and will allow an attacker to extract all private information from the key. Changing the password back to its original value after attack will prevent any detection by the legitimate user.

Serial EEPROMs are extremely common in the engineering industry and require minimal circuitry to read and write. They are also notoriously insecure and often do not provide any type of security features. Due to the nature of Serial EEPROMs, it is possible to attach a device programmer to the device, while it is still soldered onto the circuit board, and read and write to it at will. Given these known weaknesses, it behooves vendors to take steps in properly restricting access to Serial EEPROMs when employed in security-related devices.

The experiments described in this paper were carried out using the Needham's Electronics' EMP-30 [9] which has a retail price of $995, although a homebrew device programmer could be built with a handful of components for under $10. Other device programmers are available from a number of companies, ranging in cost from $25 to $1000. The prices are noted here in an effort to gauge the level of perceived threat.

3.1 Case Study of eToken

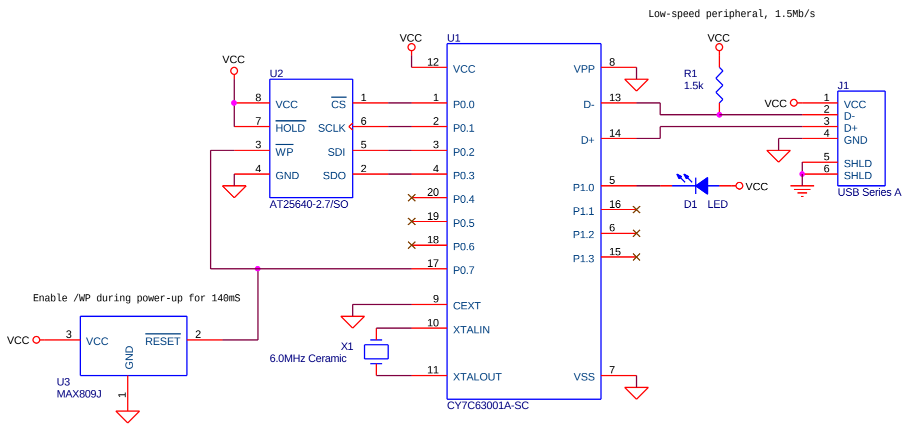


Figure 5: Electrical schematic of the eToken 3.3.3.x PCB version 4.3a

Because the Cypress CY7C63001 microprocessor [10] isn't protected by any encapsulation (Figure 6), commercial and homebrew techniques could be used to remove the top of the integrated circuit (IC) housing and gain direct access to the die, thus allowing a determined attacker to extract the firmware of the device [11, 12]. This type of attack requires a sizeable initial investment of effort and equipment, thus reducing the likelihood that a less determined attacker will take advantage of the situation. It was not performed for this paper. Having access to the firmware of the USB key often proves to be extremely useful. Analysis of the code can determine the exact operation of the device, programming errors, and any hidden features or backdoors - any of which could be used to launch further attacks.

The first attack attempted was to read the firmware programmed into the microprocessor. This was thwarted by the security bit being set in the device. Using the security bit, a common feature in microprocessors, the device will prevent a user from either reading or writing the device firmware.

The 64-bit unique serial number associated with each eToken device is hard-coded into the Cypress microprocessor firmware. This is a good design feature, as it makes exact device cloning difficult, since the firmware is write-only and protected inside the microprocessor.

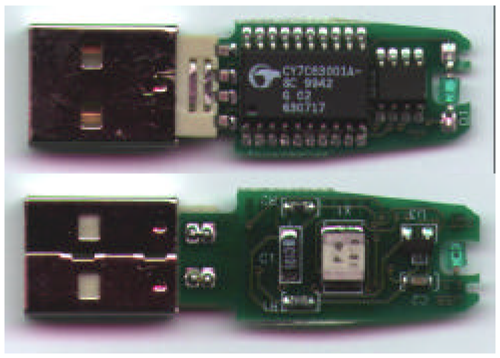


Figure 6: eToken 3.3.3.x PCB version 4.3a, top: front, bottom: back.

All data on the eToken USB key is stored in external memory. The 8KB version of the eToken uses an Atmel [13] 25640 Serial EEPROM. If requested by the customer, Aladdin Knowledge Systems can provide them with eToken devices with memory sizes up to 64KB. To read the contents of the Serial EEPROM, we simply attached the leads to a device programmer. This was done using a homebrew cable with an 8-pin Dual In-line Package (DIP) footprint on one side, to connect to the device programmer, and a 16-pin Small-Outline Integrated Circuit (SOIC) clip on the other side, to connect to the EEPROM in-circuit on the printed circuit board (PCB).

The memory map (Figure 7) of the Serial EEPROM was determined by modifying and reconfiguring the eToken data and viewing the content changes in the EEPROM.

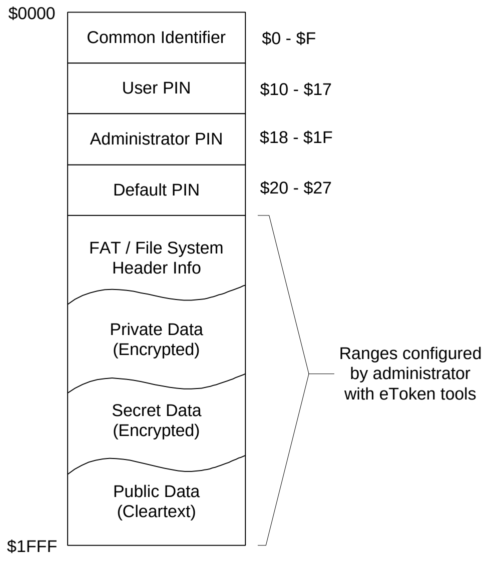
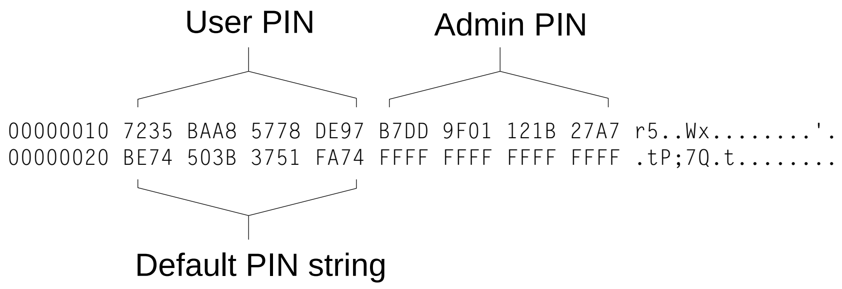


Figure 7: eToken Atmel 25640 Serial EEPROM Memory Map

During analysis of the external Serial EEPROM, it was found that the legitimate user's PIN can be reset back to the default PIN by simply copying a particular 8-byte string from one area of the unprotected external memory to another. This is explained in detail shortly. If necessary, the legitimate user's original PIN can be copied back into the external memory after the attack and no evidence of tampering will be apparent.

There are two PIN numbers associated with each eToken device, allowing either User or Administrator access. User access has complete control of the eToken file system, while Administrator is allowed to initialize the key, but not access private data. The User and Administrator PINs, private data, and secret data are encrypted before being stored in the EEPROM. The public data is stored in plaintext and can be easily read by viewing the buffer of the Serial EEPROM.

The 8-byte strings which determine the User and Administrator PINs are stored at location 0x10 and 0x18, respectively (Figure 8). By copying the 8-byte string stored at 0x20 into either of those areas, we return either the User or Administrator PIN to its default state (Figure 9). The 8-byte Default PIN string (an encrypted string representing the ASCII version of the default PIN) is unique for each eToken.



```
00000010 7235 BAA8 5778 DE97 B7DD 9F01 121B 27A7 r5..Wx........'.
00000020 BE74 503B 3751 FA74 FFFF FFFF FFFF FFFF .tP;7Q.t........
```

Figure 8: Initial memory dump, with the User PIN and Administrator PIN set to unknown values.

```
00000010 BE74 503B 3751 FA74 B7DD 9F01 121B 27A7 .tP;7Q.t......'.
00000020 BE74 503B 3751 FA74 FFFF FFFF FFFF FFFF .tP;7Q.t........
```

Figure 9: Memory dump, after modification, with User PIN now set to default.

Once the modified buffer is programmed back into the Serial EEPROM, the attacker can login to the eToken using the default PIN and make use of the legitimate user's credentials.

The ASCII version of the default PIN is 0xFFFFFFFFFFFFFFFF, which is 8 bytes of 0xFF, a non-printable character [14]. To enter the default PIN at a dialog box on a Windows platform, hold the "Alt" key while typing "0255". Release the "Alt" key between characters. Repeat this 8 times.

A proof-of-concept tool [7] was developed to demonstrate the following functions:

- **Search USB ports for eToken.**
- **Retrieve and display configuration data** for the inserted key.
- **Login as User using the default PIN** of 0xFFFFFFFFFFFFFFFF
- **Retrieve all public and private data** and export the directory hierarchy to DOS.

The tool expects that the eToken User PIN has been reset to the default state, as described earlier in this section. A portion of the console output is shown below:

```
tokenId = 00 00 00 00 00 00 a6 23
slotid = 5
isConfigured = 1
verMajor = 3
verMinor = 27
color = 0
fsSize = 8088
publicSize = 3796
privateSize = 2576
secretSize = 512
freePublicSize = 2784
freePrivateSize = 2446
freeSecretSize = 496
secretGranularity = 16
fat = 10
maxfat = 100
maxAdmin = 255
maxUser = 255

Attempting eToken User login with Default PIN...Success!

dir = 3f00
file = a000
file = 1234
file = 6666
dir = feed
dir = beef
file = beef
dir = dead
file = beef
dir = face
```

3.2 Case Study of iKey 1000

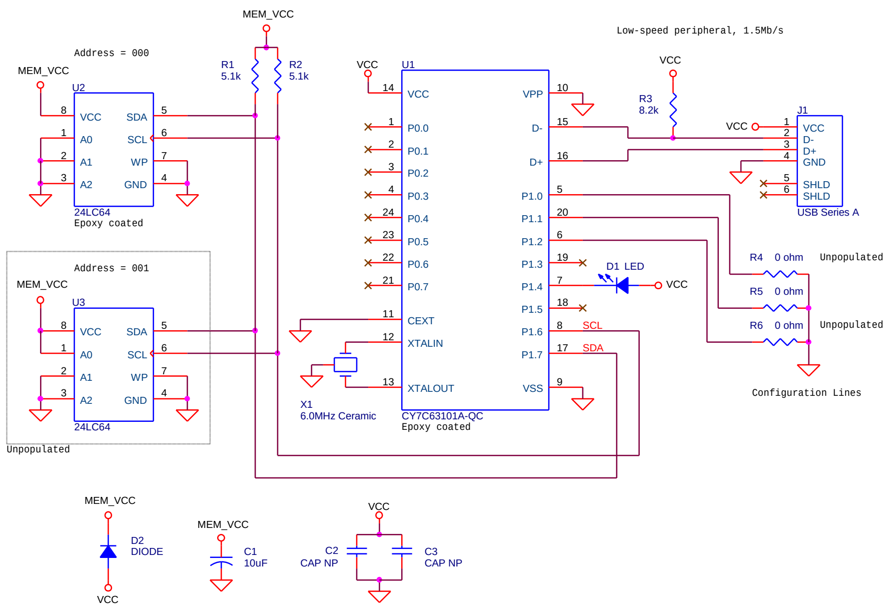


Figure 70: Electrical schematic of the Rainbow iKey 1000 PCB 106160-003

The iKey (Figure 11) has epoxy conformal coatings over all of the IC's on the PCB. This increases the difficulty of an electrical attack and is a common deterrent which should be employed in most situations. A determined attacker might still remove this coating by using chemicals or by prying and scraping with a knife, both of which are visually evident. As will soon be shown, it is essential to understand the perimeter of security and how such deterrents take or fail to take into account what they are protecting.

All data is stored on a Microchip [15] 24LC64 Serial EEPROM, which is covered by the epoxy coating. This particular version of the iKey has 8KB of external memory. However, the printed circuit board allows for an expansion to 128KB. Because of this, there is an unpopulated area for the memory, shown on the back of the circuit board. This design flaw is used to access the "protected" Serial EEPROM. This will now be explained in more detail.

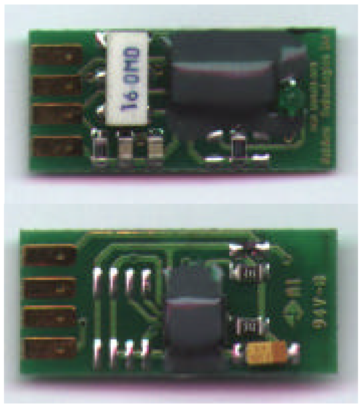


Figure 11: iKey 1000 PCB 106160-003, top: front, bottom: back.

The Microchip 24LC64 Serial EEPROM devices use the Inter-Integrated Circuit (I^2C) bus protocol for a minimal number of connections to transfer data between itself and the host:

- **Power, VCC.** Common to all devices attached to the bus.
- **Ground, GND.** Common to all devices attached to the bus.
- **Serial Data, SDA.** Bi-directional pin used to transfer addresses and data into and data out of the device. Common to all devices attached to the bus.
- **Serial Clock, SCL.** Used to synchronize the data transfer to and from the device. Common to all devices attached to the bus.
- **Write Protect, WP.** Used to protect the contents of the external memory. If pulled high, one is prevented from writing to the memory. Read operations are not affected.
- **Address Select Lines, A2, A1, A0.** Used for multiple device operation to allow up to 8 devices (2^3) on the I^2C bus. The host processor sends the address of the external memory device it wants to communicate with along with the command to the I^2C bus. Whichever Serial EEPROM is configured for the same address is selected for use.

By attaching probes or soldering small leads to the unpopulated memory footprint, the power, ground, clock, and data lines of the I^2C bus can be accessed. Even though the external Serial EEPROM being targeted is physically coated and its actual pins are not accessible, the contents can now be read by attaching the leads to a device programmer. While attaching probes to the memory is more difficult when the tamper-proofing features are correctly implemented, in this case there is a clean avenue of communications available over the I^2C bus which is free of any preventative measures. To remedy this problem, all unpopulated component areas on the PCB should be covered in epoxy or removed to prevent probing.

The 64-bit unique serial number associated with each iKey device appears to be stored in the external EEPROM, whereas the eToken stores its unique serial number in read-only memory internal to the microprocessor. Storage in external EEPROM makes it possible to change the serial number, essentially removing its uniqueness. If the serial number is used in a company's implementation, this could lead to increased channels of attack.

The iKey allows administrator access using the Master Key (MKEY) password. Administrator access to the iKey, normally used for initialization and configuration, will allow all private information stored on the key to be accessed.

The MKEY is an administrative password that must be known by the trusted person or program that will initialize and configure the iKey. The MKEY password is an ASCII string up to 256 characters in length. The default factory setting is "rainbow" [16]. The ASCII string is MD5-hashed [17], encoded with a proprietary algorithm, and stored in external memory.

Analysis of the external EEPROM led to the discovery that only the upper 8-bytes of the MD5 hash, hereby referred to as the 'hashed MKEY', are encoded and stored in external memory with the scheme described in this paper. The resultant 8-byte encoded value is hereby referred to as the 'obfuscated MKEY'. Figure 12 shows the steps taken to generate the obfuscated MKEY from the ASCII-string MKEY password.

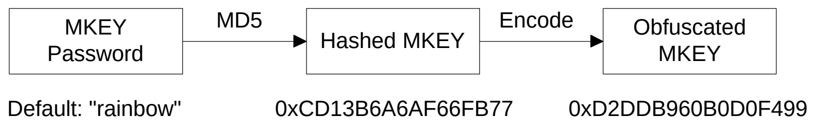


Figure 12: MKEY generation, default settings shown below each box.

All PC applications that use the iKey will generate the hashed MKEY locally before sending it to the iKey device to login. In order to login to the iKey device, the iKey Application Programming Interface (API) requires the 8-byte hashed MKEY, not the MKEY password that created it. Administrator access to the iKey can be gained in two ways:

- **Determine the hashed MKEY** from the obfuscated MKEY value which is stored in the external EEPROM.
- **Encode a new obfuscated MKEY** using a new MKEY password string and store it in the external EEPROM.

The iKey encoding scheme was determined by setting the hashed MKEY to a known value and observing the resultant obfuscated MKEY, which is located at address 0x8. After several iterations, it was evident that the scheme is a series of XORs and additions. Let *A* be the 8-byte hashed value and *B* be the 8-byte obfuscated value. Let A_i be the *i*th byte of *A*.

```
                                Byte # 1 2  3 4  5 6  7 8
A, Hashed MKEY value, md5("rainbow") = CD13 B6A6 AF66 FB77
B, Obfuscated MKEY value in EEPROM   = D2DD B960 B0D0 F499

               $B_1$ = $A_1$ XOR 0x1F
               $B_2$ = $A_2$ XOR ($A_1$ + 0x01)
               $B_3$ = $A_3$ XOR 0x0F
               $B_4$ = $A_4$ XOR ($A_3$ + 0x10)
               $B_5$ = $A_5$ XOR 0x1F
               $B_6$ = $A_6$ XOR ($A_5$ + 0x07)
               $B_7$ = $A_7$ XOR 0x0F
               $B_8$ = $A_8$ XOR ($A_7$ + 0xF3)

Example:       0xD2 = 0xCD XOR 0x1F
               0xDD = 0x13 XOR (0xCD + 0x01)
               0xB9 = 0xB6 XOR 0x0F
               0x60 = 0xA6 XOR (0xB6 + 0x10)
               0xB0 = 0xAF XOR 0x1F
               0xD0 = 0x66 XOR (0xAF + 0x07)
               0xF4 = 0xFB XOR 0x0F
               0x99 = 0x77 XOR (0xFB + 0xF3)
```

Setting the hashed MKEY to 0x0000000000000000 gave the necessary information to determine the encoding scheme. Bytes 1, 3, 5, and 7 are simply XORs with constant values and bytes 2, 4, 6, and 8 are XORs with constant values added to bytes of the hashed MKEY, as described above.

```
               Byte # 1 2  3 4  5 6  7 8
A, MKEY             = 0000 0000 0000 0000
B, EEPROM MKEY      = 1F01 0F10 1F07 0FF3
```

Once the obfuscated MKEY has been changed to a known value or the hashed MKEY has been determined, the attacker can login as administrator to the iKey device and access all of the legitimate user's data. The whole attack as described above can be completed in less than 2 minutes.

A proof-of-concept tool [8] was developed to demonstrate the following functions:

- **Retrieve and display configuration data** for the inserted iKey.
- **Convert obfuscated MKEY** back into hashed MKEY.
- **Login as Administrator** using hashed MKEY.
- **Retrieve all public and private data** and export the directory hierarchy to DOS.

The tool expects the 8-byte obfuscated MKEY on the command-line, which is obtained from reading the external Serial EEPROM with the use of a device programmer as described earlier in this section. A portion of the console output is shown below:

```
C:\>ispy D2DDB960B0D0F499

Magic = 5242544B
DeviceHandle = 80
ClientHandle = 205408
Flags = 20000000
library_version = 2
driver_version = 256
ver_major = 0
ver_minor = 7
prod_code = 54
config = 0
header_size = 8
modulus_size = 0
mem_size = 8168 (bytes)
capabilities = 11
SerialNumber = 0123466A00000249
CheckSum = FAD1
HwInfo = FFFF
MaxPinRetries = 5
CurPinCounter = 5
CreateAccess = 0
DeleteAccess = 0

Obfuscated MKEY = D2 DD B9 60 B0 D0 F4 99   [...`....]
Actual MKEY     = CD 13 B6 A6 AF 66 FB 77   [.....f.w]

Attempting iKey Administrator login...

VerifyMasterKey: SUCCESS

dir  = 00000000
file = 00000001
dir  = 000000C1
file = 000000C1
file = 0000BEEF
dir  = 0000FEED
```

3.3 Electrical Design Variations Between iKey 1000 and 2000

The iKey 2000 was briefly examined to determine the variances between it and the iKey 1000. The manufacturing processes of iKey 1000 and 2000 are similar. However, there are slight changes to the iKey 2000 PCB (Figure 13):

- **Microprocessor Package.** The package of the microprocessor has been changed from a Quad Small Outline Package (QSOP) to Small-Outline Integrated Circuit (SOIC) footprint. The part number has been scratched off the top of the package, making it difficult to identify. However, a part of the Cypress logo is still visible on the package, so it can safely be assumed that the iKey 2000 still uses the Cypress CY7C63101A device, which was also used in the iKey 1000. This 24-pin device is available in both QSOP and SOIC footprints.
- **USB Connector.** The metal fingers that serve as the USB connector are soldered onto the PCB on one side. This might be done for a number of reasons, possibly due to slippage of the PCB underneath the fingers or to simplify the manufacturing process.
- **Conformal Coating.** The conformal coating has been removed from the microprocessor, which makes access to the pins much easier. Commercial and homebrew techniques could be used to remove the top of the IC housing and gain direct access to the die, thus allowing a determined attacker to extract the firmware of the device [11, 12].
- **External Memory.** The package type of the external memory has been changed. The black, square box on the front side of the circuit board is the new package, which is the silicon die encapsulated with epoxy, also known as chip-on-board (COB). The gold pad to the right of the memory is an area for memory expansion. There are only 8 pads on the footprint, so it can safely be assumed that the memory remains a Serial EEPROM. Although the new footprint is harder to probe, the same design flaw exists as the iKey 1000, in which the pads for the unused memory expansion area are still accessible.

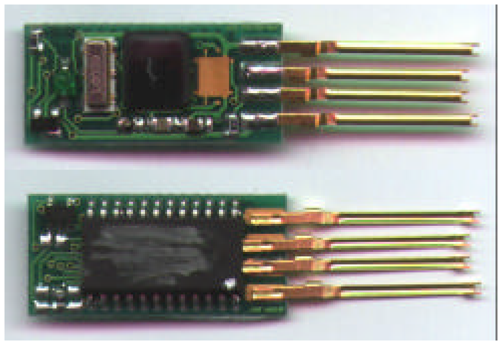


Figure 13: iKey 2000 PCB 106420-004, top: front, bottom: back.

The pinout of the external memory footprint (Figure 14) was determined by simple probing of the circuit board and using the iKey 1000 schematic as a reference. The pinout appears to be similar to that of the Microchip 24LC64 device which was used in the iKey 1000.

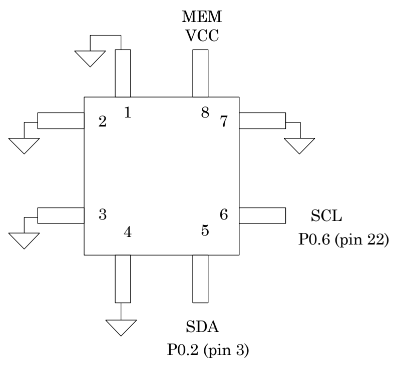


Figure 14: iKey 2000 unpopulated memory pinout

Three of the pins connected to GND in Figure 14 are most likely the Address Select Lines, A2, A1, A0, of the Serial EEPROM. Whichever Serial EEPROM is configured for that address (by hardwiring the pins to a logic high or low) is selected for use. It appears that the address of the unpopulated device is 000. If this is so, the actual encapsulated EEPROM could have an address ranging from 001 – 111 (binary). The external memory attack described in Section 3.2 has not been attempted on the iKey 2000, though it is believed it would be successful in reading the contents of the device.

3.4 Electrical Design Recommendations

Developers of security-based products, not limited to token devices of the nature described in this paper, should consider the following electrical features for design and manufacture:

- **Conformal Coatings,** such as epoxy, help protect critical components from probing and tampering. Coatings, when implemented correctly and unless they are easily removed or dissolved by chemicals, serve as a good deterrent to many attackers. As a benefit to the designer, common attacks using sulfuric acid to dissolve the epoxy coating will also dissolve the wire-bonds of an exposed die, thus rendering the device unusable.
- **Microprocessors with Internal Non-Volatile Memory Storage** will deter the casual attacker by requiring advanced techniques, such as de-lidding and microscopic inspection of the IC die, to determine the data stored in the memory [11, 12].
- **Non-Standard or Hard-To-Probe Package Types** for integrated circuits, such as ball-grid-array (BGA) or chip-on-board (COB) help deter the casual attacker, since the pins of the IC are either hidden or difficult to access.

The major flaw in the iKey design is that the external Serial EEPROM, which was meant to be protected by the epoxy encapsulation, is still left open to attack using the unused memory upgrade footprint to access the protected pins. If a memory upgrade is not present, these pads should be removed or covered in epoxy to help prevent attack. On the iKey 2000, the pins can also be accessed from the unprotected pins of the SOIC-packaged Cypress microprocessor.

A temporary fix to the electrical attacks, although it does not remedy the core problem, is to be very aware of the physical security and location of the key at all times. The owner of the key should not leave the key unattended or loan it to a potentially untrustworthy colleague. If the key is unattended for any amount of time, the data could be compromised with the methods described in this paper.

4 Software Attack

Software attacks are considered non-invasive attacks in which the device is not harmed or physically tampered with. Non-invasive and software attacks often, but not always, make use of the normal operating conditions of the device and are aimed to find flaws in the implementation of the software or firmware in a product. Once the attack is designed and successful, the results are reproducible from one device to another.

The attacks chosen for the USB key investigation fell into two distinct areas:

- **Examine the communication channels** between the USB device and host computer, using custom device drivers and commercial USB protocol analyzers, and look for undocumented commands and problems with handling intentionally erroneous and mis-structured commands.
- **Analyze and determine the possibility to brute-force a password** which will give access to the USB key device (i.e., the Administrator's MKEY value or the legitimate user's password or PIN).

The source code and header files included with the vendor-provided software development kits (SDK) contain a lot of interesting information about the design and structure of the device and API in question. For example, the PC software included in the eToken SDK prints Windows debugging messages. These messages contain bits and pieces of the Serial EEPROM contents of the key and may be leaking secret or private information.

Not all of the attacks described in this section have been completed. Rather, they are indicative of attacks which might yield interesting information about the design of the USB keys.

4.1 USB Protocol Analyzers

A number of commercial tools exist to aid in the designing and debugging of USB devices. These tools, ranging in rental cost up to $3290 per month and retail price of up to $25K, allow detailed analysis, trace, and storage of protocol traffic. Some models allow the generation of traffic onto the USB bus which can contain both legal packets per the USB standard [18] and illegal packets, thus enabling stress/limit testing of USB-based designs, and observation of design behavior under faulty bus conditions.

4.1.1 Undocumented Commands

It is possible that these USB keys have undocumented command sets. During the development phase, it is common to need access to the Serial EEPROM data on a regular basis. The use of software commands to allow this would avoid the need of additional device programmer hardware and resources.

Currently, with the electrical attacks described in Section 3, it is possible to obtain the entire contents of the device by physically opening it and reading its contents with a device programmer. It would be useful to find a command that would dump the entire contents of the Serial EEPROM, used for storage of all public and private data, back through the USB port and bypass any private-memory restrictions. By searching through any undocumented USB commands, it may allow one to obtain all possible information and data from the USB key without physical tampering. The search was not performed on any of the devices, but the attack methodology is as follows:

1. **Analyze typical data transactions** between the host PC and the USB key. This will allow us to see how the commands and data are structured in the USB packet.
2. **Send custom, legal USB packets** making use of the traffic generation features of the USB protocol analyzer. The custom packets will be in the form that was obtained in Action 1, replacing the command data each time. By linearly advancing through the entire command keyspace (i.e., 0x0000 to 0xFFFF if the command structure is 2 bytes), we will be able to detect any hidden commands.
3. **Monitor the data** being transmitted from the USB key. Look for the response in which most or all of the contents of the Serial EEPROM are being transmitted back to the host PC, or any response that isn't defined in the standard API or command-set of the device.

4.1.2 Illegal USB Packets

By sending incorrect and known erroneous USB packets to the USB key, it may leak information such as the contents of protected memory areas. This experiment is similar to Section 4.1.1, except the packets generated will be deliberately illegal USB packets not conforming to the USB standard.

1. **Send illegally-structured USB packets** making use of the traffic generation features of the USB protocol analyzer. Changing any of the

fields in the USB packet could potentially lead to unintended leakage of information.

2. **Monitor the data** being transmitted from the USB key. Look for interesting responses and information that aren't defined in the standard API or command-set of the device.

4.2 MKEY Timing Attack with iKey 1000

The API function responsible for MKEY authentication, *RnbTkn_VerifyMasterKey*, takes an 8-byte *char* array and encodes and compares this against the obfuscated MKEY stored in the iKey. There are no counters or limits designed to prevent brute-force attacks on the MKEY value. However, there are 2^{64} possible values, making typical brute-force methods infeasible. Because the Cypress CY7C63000-family of microprocessor is an 8-bit device, meaning all registers and operation codes are handled in 8-bit chunks, it is theorized that timing attacks could aid in determining the MKEY value in shorter time.

When an 8-bit processor needs to compare two 64-bit numbers (defined herein as A_i and B_i, where i is each byte ranging from 1 to 8), it is achieved by first comparing A_1 with B_1. If they match, A_2 and B_2 will be compared, and so on until the entire value has been compared (Figure 15).

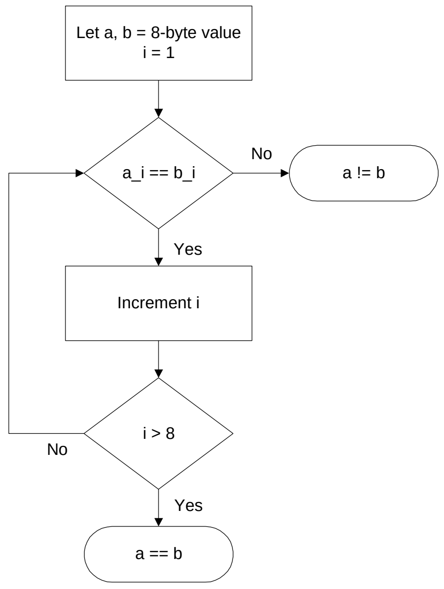


Figure 15: Comparing two 64-bit values on an 8-bit microprocessor

Using this process, it is possible to use time measurements to determine how close the guessed MKEY value is to the correct MKEY value. The more bytes that match, the longer the compare routine will take. If the routine returns quickly, it can be assumed that the bytes being compared do not match. This greatly reduces the amount of time necessary for a brute-force attack, since the 8-byte values are being compared on a byte-by-byte basis and the correct value of one byte will be successfully guessed before moving on to the next.

As it happens, the latency times of the device drivers on the Windows operating system are too large to get accurate timing measurements from the USB key. It takes too long for the USB command to reach the key and for its response to get back to the Windows application. Because of this, we developed custom device drivers and kernel module for use with the Linux operating system. These custom components allow for a more directed control of the device thus reducing the time delay between the sending of the USB command and the reception of the command by the USB key. It may be possible to further reduce the latency by using the traffic generation on a USB protocol analyzer, which would allow commands to be sent directly to the key without relying on any drivers or operating system software.

However, this attack is more complicated in that a *RnbTkn_GetResults* API function must be called to retrieve the results of the last operation (*RnbTkn_VerifyMasterKey*, in the case of this attack). Due to the device driver latency and the need to call a second function to retrieve the compare result, it is difficult to measure how long a compare operation really takes to complete. Timing attacks are well known and have been used against other cryptographic systems [19].

4.3 Software Design Recommendations

In relation to the attacks described in this section, the following programming practices should be considered:

- **Remove all undocumented commands, debug symbols and development functions.** All functionality not used or needed in the production unit should be completely removed from the firmware.
- **Protect against malformed and illegal USB packets.** The software is in control of interpreting the USB device requests and responding correctly to them. Make sure the device does not leak critical information or enter an unintentional state if a deliberately incorrect USB packet is sent to it. Verification of proper handling can be verified by using a USB protocol analyzer to generate intentionally bad packets.
- **Design each routine to take a constant amount of time.** Any critical function calls (such as *RnbTkn_ VerifyMasterKey* described in the previous section) should take a constant amount of time to complete regardless of

the result of the operation. This will prevent timing attacks from being successful in determining passwords or other critical information.

5 Conclusions

Users must be aware that at today's level of delivered products, private data can be accessed from USB keys without having legitimate credentials. If a user loses their USB key, all data should be considered to have been potentially compromised and proper action should be taken.

This paper described and detailed a number of practical and theoretical attacks related to the mechanical, electrical, and software aspects of the USB keys. These attacks are not meant only for USB keys and could be expanded upon and attempted on other products. There are flaws in the existing USB hardware tokens on the market today, and users must recognize the security risks and benefits of each tool before it is recommended and implemented into their infrastructure. Some of these flaws can be worked around, but only after the weaknesses have been identified. It is important for designers of hardware devices, especially security products, to fully understand the threat model of their particular product before implementing a solution.

Acknowledgements

Thanks to DilDog and Brian Carrier (software attack ideas and cryptanalysis), and Dan Geer, Mudge, @stake Research Labs and Nybor (editing and peer review).

References

1. Clark, Andrew J., "Physical Protection of Cryptographic Devices", *Eurocrypt: Advances in Cryptography*, April 1987, pp. 83-93.
2. Chaum, D., "Design Concepts for Tamper Responding Systems", *Crypto 1983*, pp. 387-392.
3. Weingart, S.H., White, S.R., Arnold, W.C., Double, G.P., "An Evaluation System for the Physical Security of Computing Systems", *Sixth Annual Computer Security Applications Conference 1990*, pp. 232-243.
4. Aladdin Knowledge Systems eToken Web Page, `http://www.ealaddin.com/etoken`.
5. Rainbow Technologies iKey 1000 Web Page, `http://ikey.rainbow.com`.
6. Rainbow Technologies iKey 2000 Web Page, `http://www.rainbow.com/ikey2000/index.html`.
7. Kingpin, @stake Security Advisory, "eToken Private Information Extraction and Physical Attack", `http://www.atstake.com/research/advisories/2000/etoken-piepa.txt`.
8. Kingpin, @stake Security Advisory, "iKey 1000 Administrator Access and Data Compromise", `http://www.atstake.com/research/advisories/2000/ikey-admin.txt`.
9. Needham's Electronics Device Programmer Web Page, `http://www.needhams.com`.

10. Cypress Semiconductor CY7C63000-family Web Page, `http://www.cypress.com/cypress/prodgate/usb/cy7c63000a.html`.
11. F. Beck, "Integrated Circuit Failure Analysis – A Guide to Preparation Techniques", John Wiley & Sons, 1998.
12. O. Kömmerling and M. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," *USENIX Workshop on Smartcard Technology Proceedings*, May 1999.
13. Atmel Web Page, `http://www.atmel.com`.
14. Aladdin Knowledge Systems eToken Developer's Guide, November 1999, pg. 36.
15. Microchip Technology Web Page, `http://www.microchip.com`.
16. Rainbow Technologies iKey PowerTools Plus Manual, May 1999, pg. 3.
17. MD5 Message-Digest Algorithm RFC-1321, `ftp://ftp.isi.edu/in-notes/rfc1321.txt`.
18. Universal Serial Bus Specification Revision 1.1, `http://www.usb.org/developers/data/usbspec.zip`.
19. P. Kocher, Cryptography Research, Inc., "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems", `http://www.cryptography.com/timingattack`.

Appendix A Additional Resources

1. USB Implementers Forum Web Page, , `http://www.usb.org`.
2. B & G International IC Decapsulation Products Web Page, `http://www.bgintl.com/dcap.html`.
3. CATC USB Bus & Protocol Analyzers Web Page, `http://www.catc.com/products.html#USBDevTools`.