A Historical Look at Hardware Token Compromises

Black Hat USA 2004 Briefings

Wednesday, July 28, 4:45pm - 6:00pm

Joe Grand

Grand Idea Studio, Inc.

`joe@grandideastudio.com`

Agenda

- Goals
- Attacks on USB Authentication Tokens
  - Aladdin Knowledge Systems eToken 3.3.3.x
  - Rainbow Technologies iKey 1000
  - Brief look at newer versions
- Attacks on iButton
  - Dallas Semiconductor DS1991

Goals

- Defeat security mechanisms
  - Access to data stored on the devices
  - Forging a user's identity to gain access to a system
- Understand <u>classes</u> of problems
- Examine possible workarounds/fixes
- Education by demonstration
- Learn from history

Authentication Tokens

- Used to provide identity in order to gain access to an asset
  - How do you prove you are who you say you are?
- Typically used in combination with a password
  - Two-factor
  - Something you know and something you have
- Common security-related uses
  - Private data storage (credentials, crypto keys, certs, passwords)
  - One-time-password generation

Hardware Tokens: USB

- Aladdin Knowledge Systems eToken 3.3.3.x

- Rainbow Technologies iKey 1000

- Note: Both vendors claim that the tokens I had were "prototypes"
- Research performed May-July 2000

Hardware Tokens: USB 2

- Analysis of three areas:
  - Mechanical
  - Electrical
  - Software/Firmware

USB: Mechanical

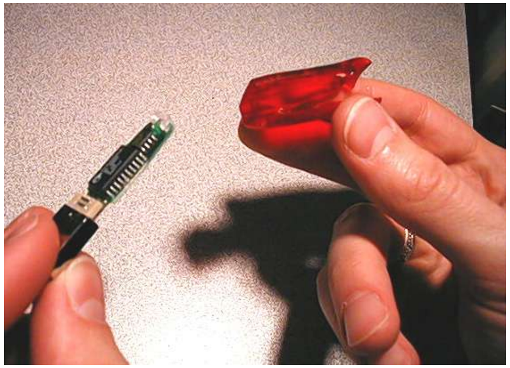
- Goal is to get access to internal circuitry
- Can succeed with no visible evidence of tampering
- Can open physical packages using standard tools

Device	Difficulty To Open	Protection of Circuitry?
eToken 3.3.3.x	Moderate	None
iKey 1000	Easy	Moderate (Epoxy)

USB: Mechanical
Aladdin eToken 3.3.3.x

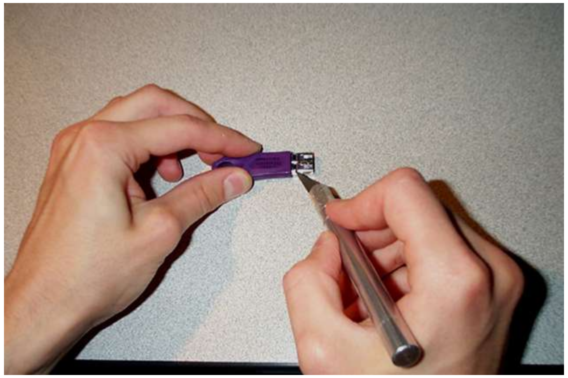
- Glue around housing, can soften with heat gun
- Split one side with X-ACTO knife
- Requires marginal amount of care
- After an attack, can simply glue to re-seal housing

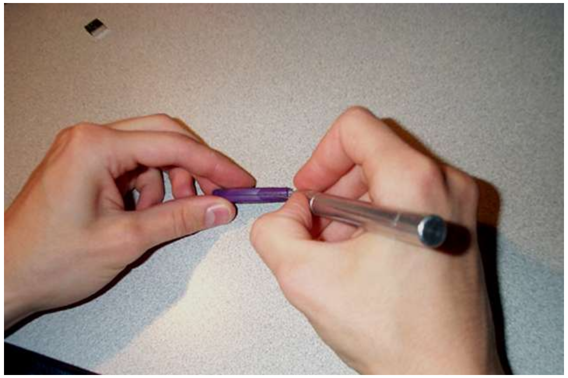


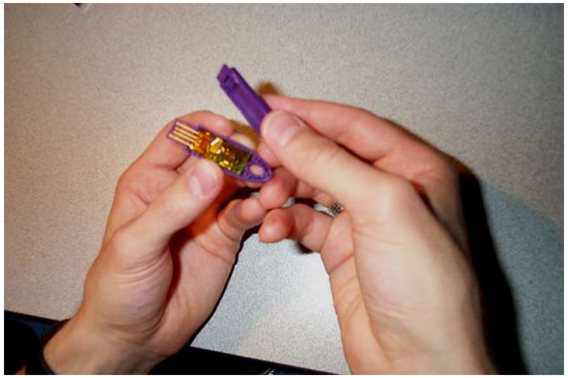


USB: Mechanical
Rainbow iKey 1000

- No glue
- Extremely easy to open with X-ACTO knife
- Under 30 seconds with no visible damage

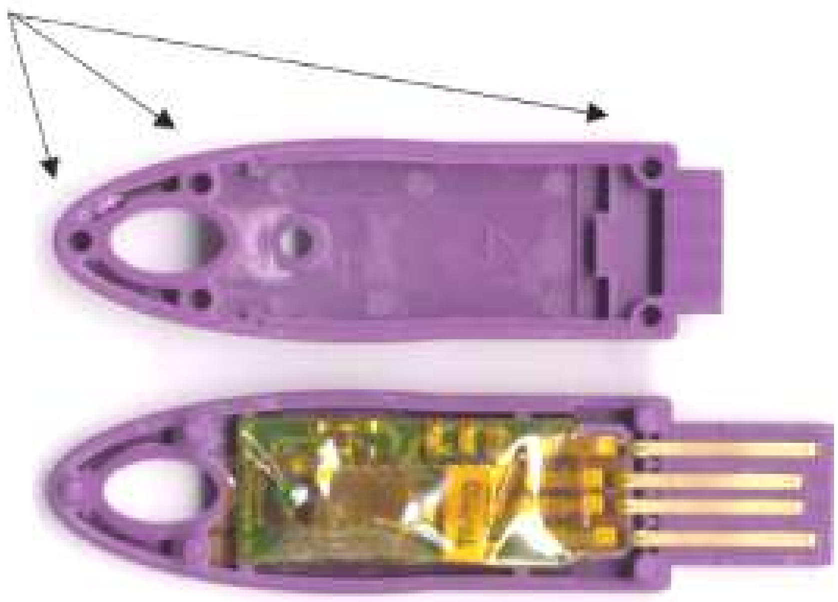






USB: Mechanical 2
Rainbow iKey 1000

- Mechanical features hold housing together
  - Socket & post
  - Metal housing of USB connector serves as a clamp



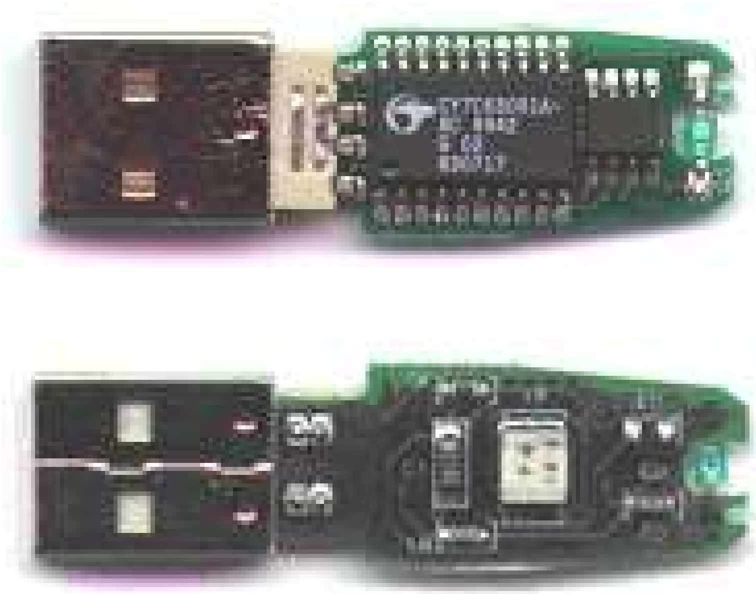
USB: Mechanical Recommendations

- Prevent easy opening using sealed/molded housing
  - Ultrasonic welding or high-temperature glue
  - If done properly, will require destruction of device to open it
  - Consider service issues (if a legitimate user can open device, so can attacker)
- Add tamper mechanisms (epoxy encapsulate)
- Obfuscate part numbers

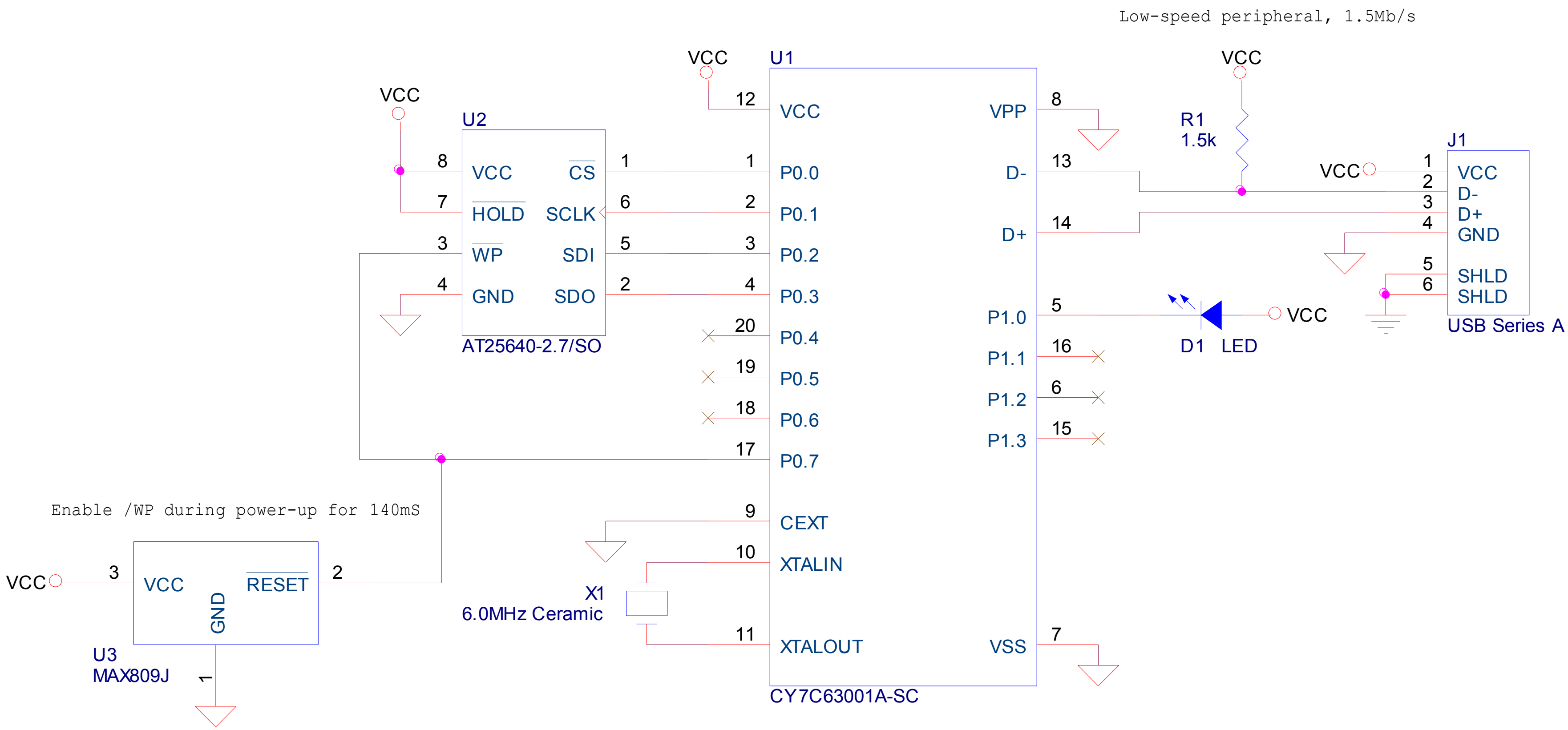
USB: Electrical

- With access to circuitry, we can now reverse engineer and look for weaknesses
- Similar design of all products – led to same vectors of attack
- Improper protection of external memory
  - Most memory is notoriously insecure
  - Serial EEPROMs can be read in-circuit
- Use low-cost device programmer to retrieve data
- Weak encoding algorithms used to protect the PINs

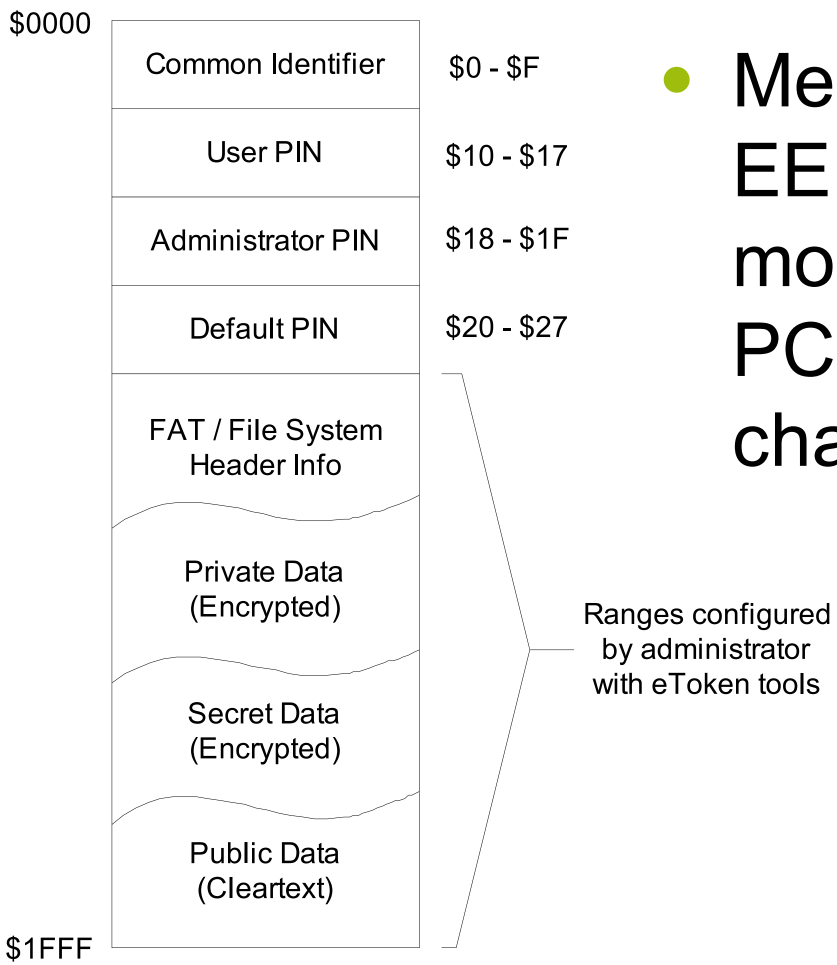
USB: Electrical
Aladdin eToken 3.3.3.x



USB: Electrical 2
Aladdin eToken 3.3.3.x



USB: Electrical 3
Aladdin eToken 3.3.3.x



- Memory map of Serial EEPROM obtained by modifying eToken data on PC and viewing content changes in EEPROM

USB: Electrical 4
Aladdin eToken 3.3.3.x

User PIN   Admin PIN

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

Default PIN string

Initial memory dump, User and Admin PINs 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........
```

Memory dump, after modification, with User PIN now set to default

USB: Electrical 5
Aladdin eToken 3.3.3.x

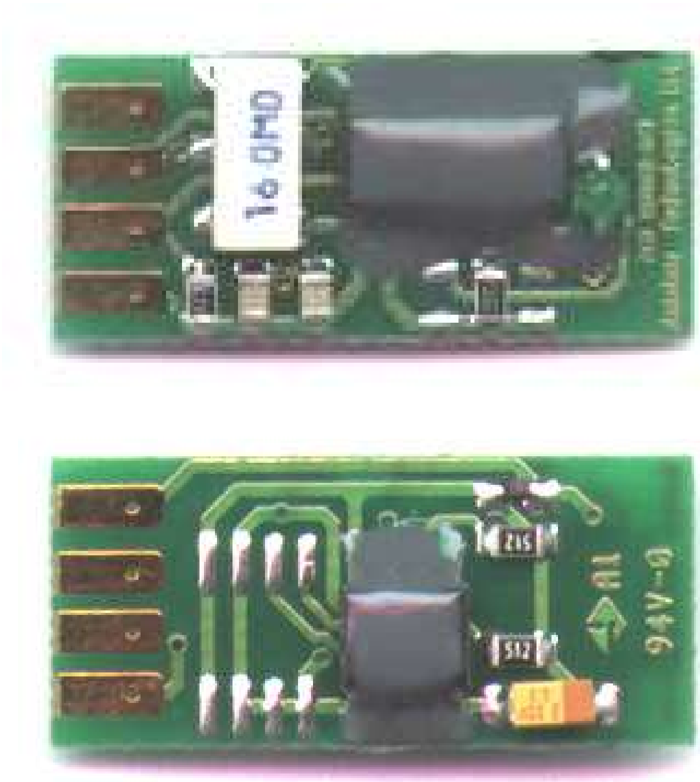
- Demo: "Heimlich" (requires old eToken SDK 1.0)
  - Search USB ports for eToken
  - Retrieve and display configuration data for the inserted key
  - Login as User using the default PIN of **`0xFFFFFFFFFFFFFFFF`**
  - Retrieve all public and private data and export the directory hierarchy to DOS
- Tool expects that eToken User PIN has been reset to default state (using device programmer)

USB: Electrical 6
Aladdin eToken 3.3.3.x

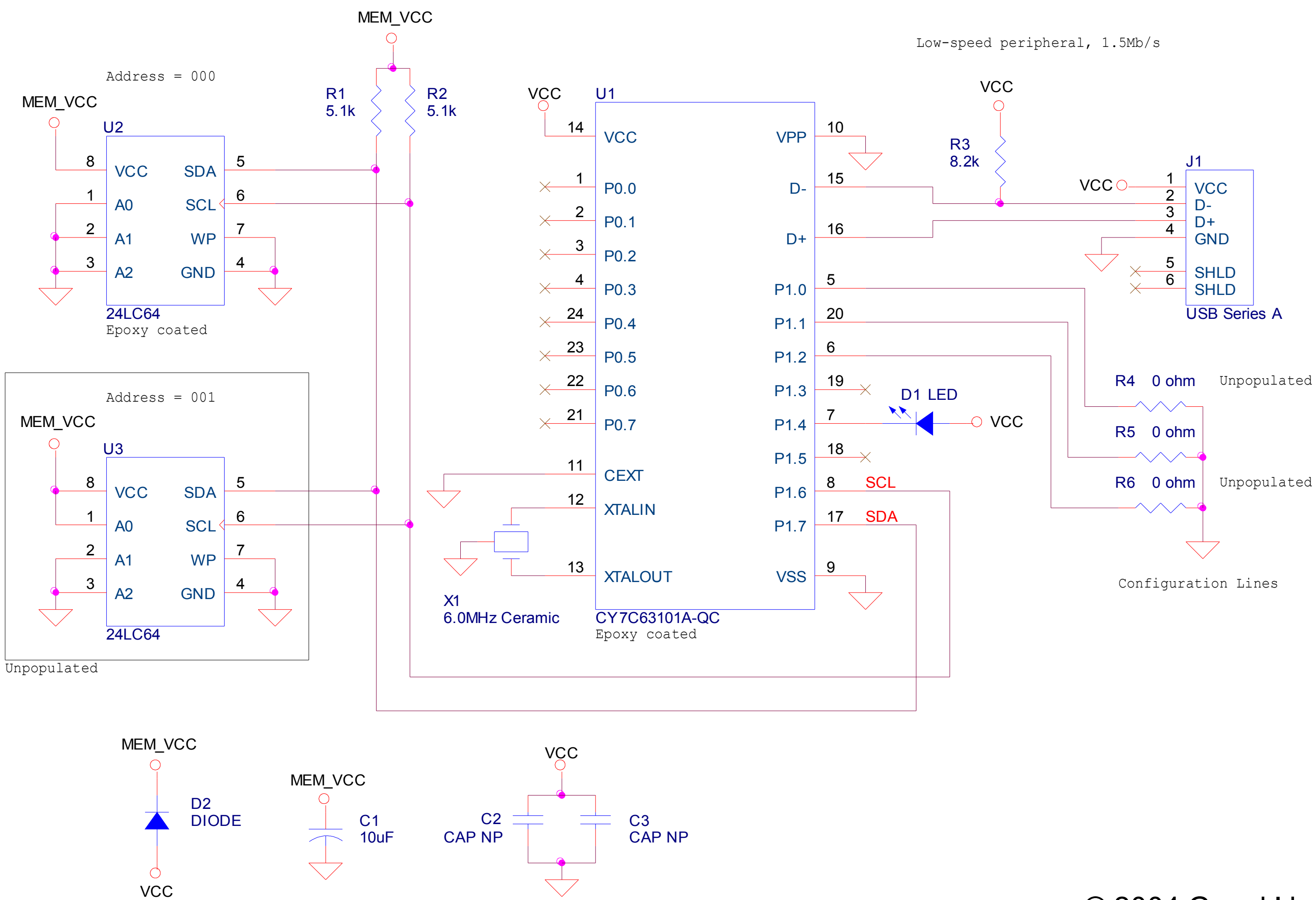
```
eToken found on Slot 5

tokenId = 000000000000a623
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
```

```
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

Heimlich maneuver complete.
```

USB: Electrical
Rainbow iKey 1000



USB: Electrical 2
Rainbow iKey 1000



USB: Electrical 3
Rainbow iKey 1000

- Can attach probes to the unpopulated footprint and read the "encapsulated" EEPROM
  - 24LC64 uses I^2C bus (serial clock and data)
- 64-bit "unique" serial number of each device stored in EEPROM
  - Can be changed, removing its uniqueness

USB: Electrical 4
Rainbow iKey 1000

- MKEY (Master Key) serves as administrative password (gives full access to device)
  - 256 character ASCII, default = "rainbow"
  - Hashed MKEY stored at address 0x8

MKEY Password $\xrightarrow{\text{MD5}}$ Hashed MKEY $\xrightarrow{\text{Encode}}$ Obfuscated MKEY

MKEY Password	Hashed MKEY	Obfuscated MKEY
Default: "rainbow"	0xCD13B6A6AF66FB77	0xD2DDB960B0D0F499

USB: Electrical 5
Rainbow iKey 1000

```
                                          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) ...
```

USB: Electrical 6
Rainbow iKey 1000

- Determined encoding by setting hashed MKEY to known value:

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

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)

USB: Electrical 7
Rainbow iKey 1000

- All PC applications convert password to hashed MKEY locally before sending it to key
  - iKey API requires the 8-byte hashed MKEY
  - Do not need to know the actual password to access device, just the hash
- Administrator access can be gained in 2 ways:
  - Determine the hashed MKEY from the obfuscated MKEY value which is stored in the EEPROM
  - Encode a new obfuscated MKEY using a new password string and store it in the EEPROM

USB: Electrical 8
Rainbow iKey 1000

- Demo: "iSpy"
  - Retrieve and display configuration data for the 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
- Tool expects that obfuscated MKEY has been read from the Serial EEPROM (using device programmer)

USB: Electrical 9
Rainbow iKey 1000

```
OpenDevice: SUCCESS

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

Obfusc. MKEY = D2DDB960B0D0F499
Actual MKEY  = CD13B6A6AF66FB77

VerifyMasterKey: SUCCESS

dir  = 00000000
file = 0000BEEF
dir  = 0000FEED
```

USB: Electrical Recommendations

- Use microprocessors with internal memory
- Make sensitive components difficult to access
  - Ex.: Microprocessor, ROM, RAM, or programmable logic
- Cover critical components in epoxy encapsulation/conformal coatings
  - Prevents moisture, dust, corrosion, probing
  - Difficult, but not impossible, to remove with solvents or Dremel tool (and wooden skewer as a "bit")

USB: Electrical 2 Recommendations

- Non-standard or hard-to-probe package types
  - Chip-on-Board (COB)
  - Ball-Grid-Array (BGA)
- Remove identifiers and markings from ICs
  - Known as "De-marking" or "Black topping"
  - Use stainless steel brush, small sander, micro-bead blast, laser etcher, or third party

USB: Software

- Defined as non-invasive, no physical tampering of device
- Two primary goals:
  - Examine the communication channels between USB device and host computer
  - Analyze and determine the possibility to brute-force a password
- Inconclusive based on our attacks, could be expanded

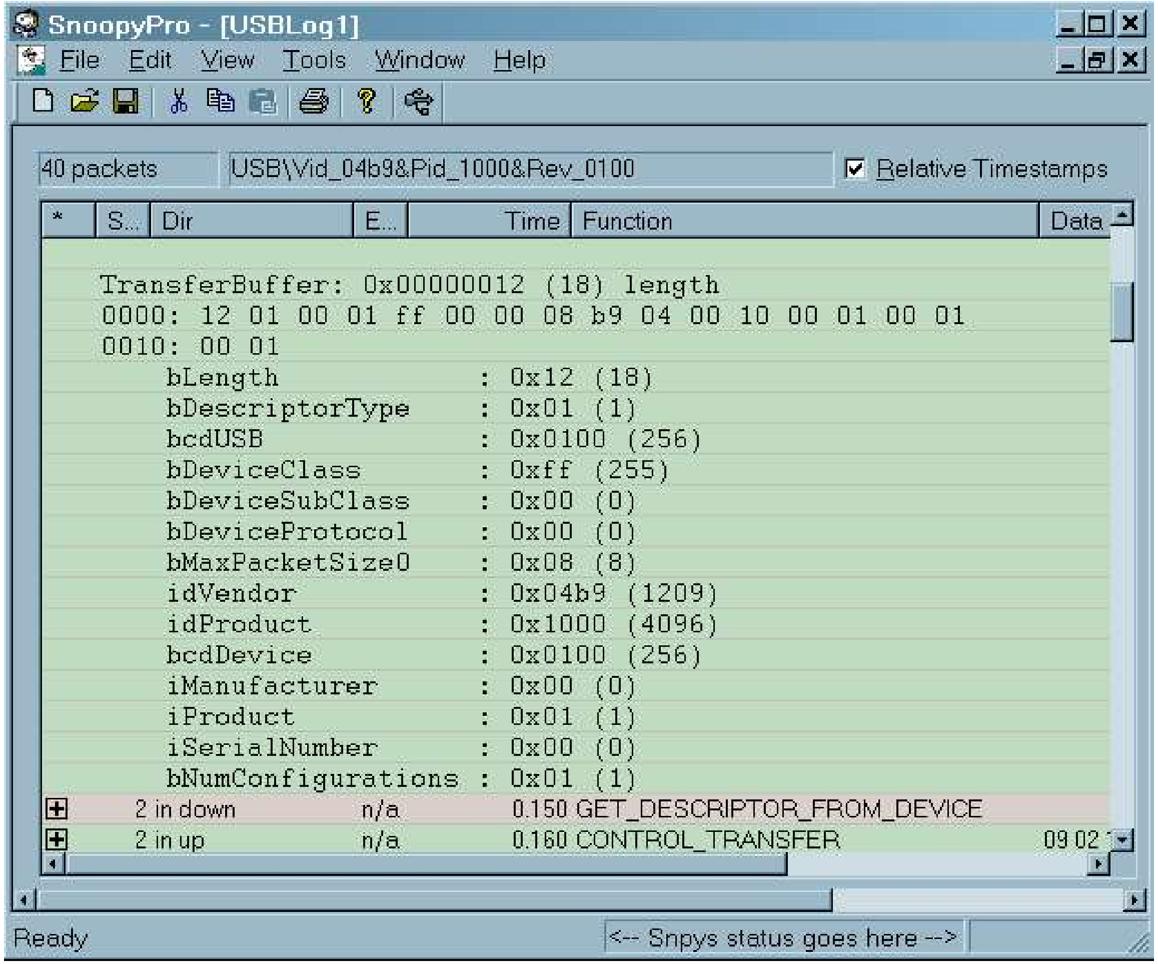
USB: Software Communication Channels

- Look for undocumented commands/debug functionality
- Check for improper handling of intentionally illegal packets
- Attack process:
  - Analyze typical data transactions
  - Send commands outside of regular keyspace **OR**
  - Send illegally-structured USB packets
  - Monitor the data on the bus

USB: Software 2 Communication Channels

- Could use hardware or software USB protocol analyzer for additional investigations
  - HW: CATC, USBee, Jungo USB Tracker
  - SW: SnoopyPro (aka USB Snoopy), SourceUSB

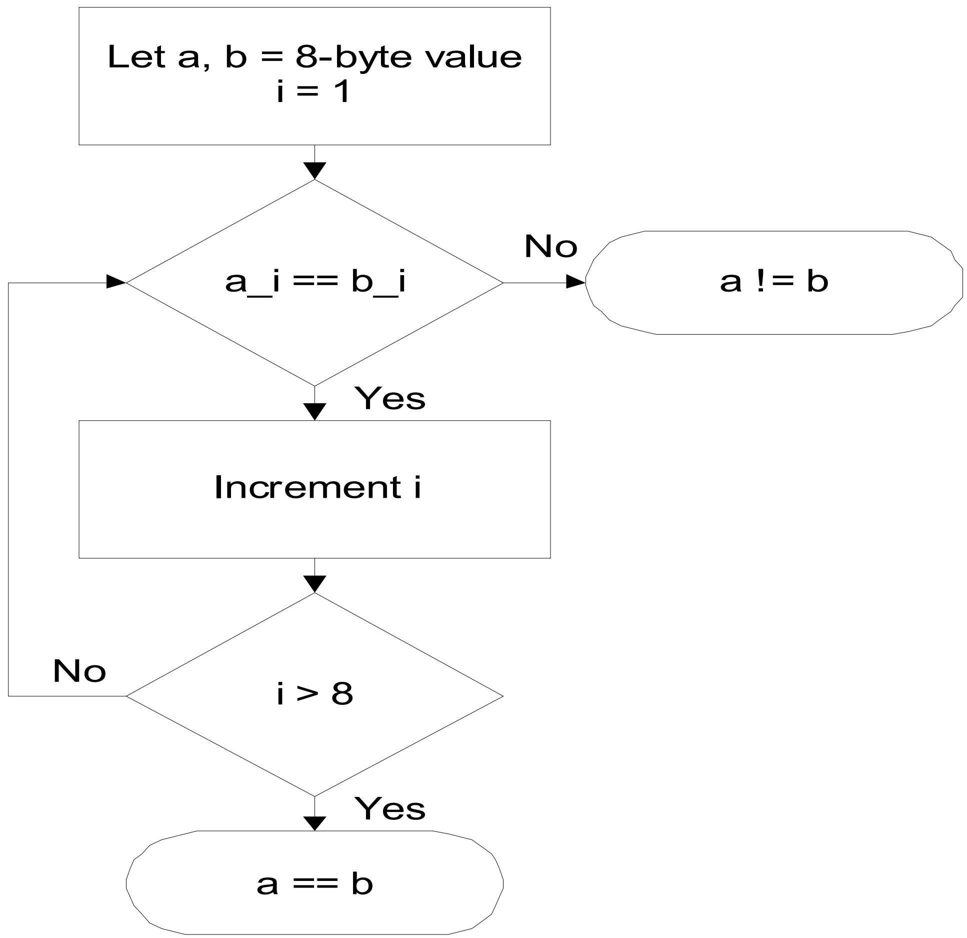
USB: Software 3
Communication Channels



USB: Software
Rainbow iKey 1000

- Timing attack to brute-force MKEY value
- No counters for invalid MKEY attempts (though counter exists for invalid user attempts)
- Brute-force of 64-bit MKEY value not feasible
- Take advantage of how a "compare" function works on an 8-bit processor
  - Longer time for more matching bytes
- Driver latency prevents accurate measurements
  - Maybe better using Linux or custom USB host?

USB: Software 2
Rainbow iKey 1000



USB: Software Recommendations

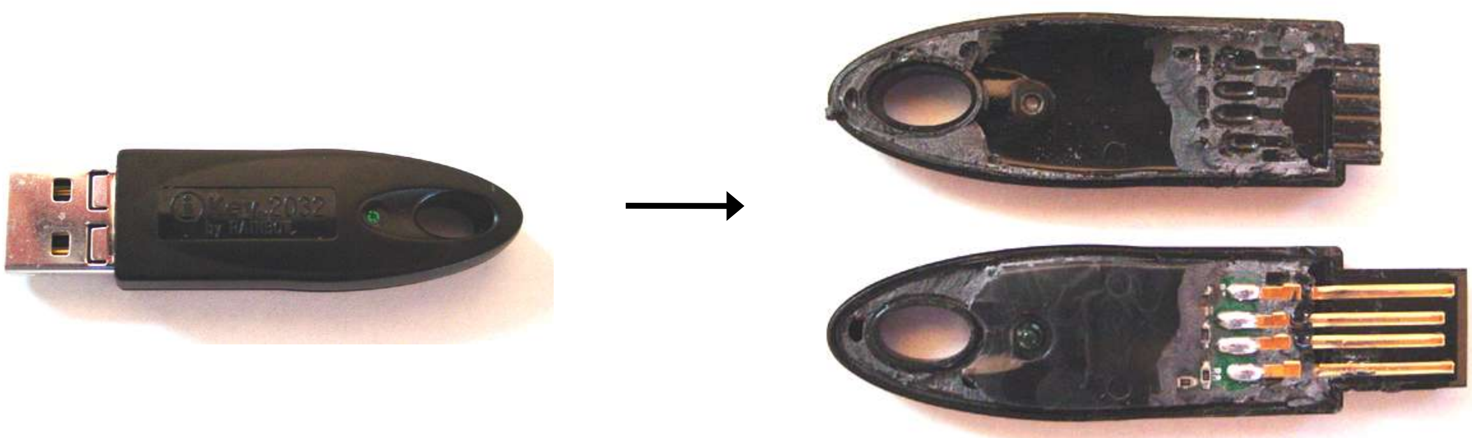
- Remove all:
  - Undocumented commands/functionality
  - Development routines
  - Debug symbols
- Protect against malformed, illegal packets
  - Intentionally sent by attacker to cause fault
- Design each routine to take a constant amount of time

USB: New Token Technologies

- Quick evaluation of some newer versions of USB tokens
  - Rainbow iKey 2032
  - Authenex A-Key
- Hypothesized attacks and weaknesses
- In general, devices are tougher to open and access circuitry
- No known public research performed on any of these devices

USB: New Token Technologies
Rainbow iKey 2032

- Black two-piece plastic housing
- Potted with encapsulate (cracked on opening)
- Encapsulate softens with heat gun

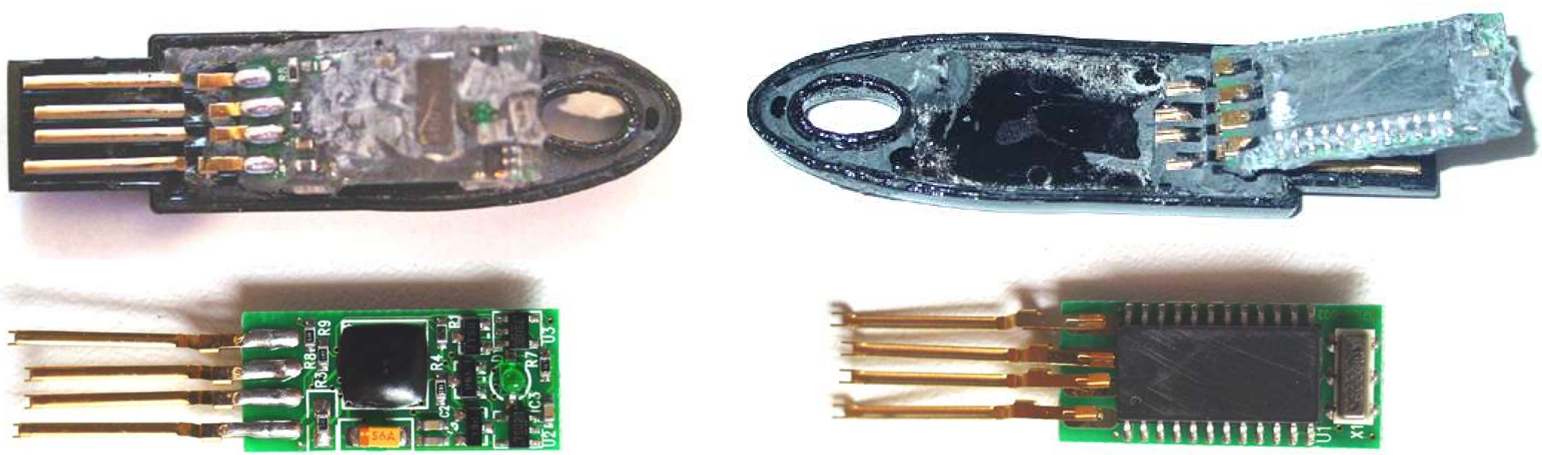


USB: New Token Technologies 2 Rainbow iKey 2032

- Can access all pins of processor (24-pin SOIC)
- Probe known connections (USB) to guess at device pinout
  - Likely Cypress CY7C63000A or CY7C63743
  - Aladdin data sheet mentions Philips 5032 Secure Smartcard Controller
- Can monitor I/O pins for interface between processors and/or memory
- Specific attacks against Philips 5032

USB: New Token Technologies 3
Rainbow iKey 2032

- Obtained an earlier, non-encapsulated version
- Can compare features/components
- Similar parts, slightly different layout

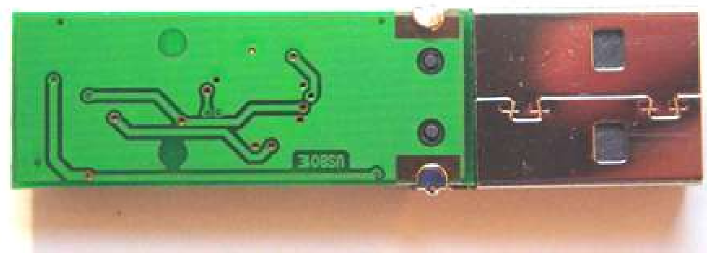


USB: New Token Technologies Authenex A-Key

- Black sealed two-piece plastic housing
- Removed plastic with Dremel tool along seam
- Circuitry completely unprotected inside

USB: New Token Technologies 2
Authenex A-Key

- Chip-on-Board (COB) with 48MHz oscillator & voltage regulators?
- 16kB Flash memory on-board
- User password: 6-63 ASCII characters stored in Flash
- Could remove epoxy and analyze die



Hardware Tokens: iButton

- Dallas Semiconductor (now part of Maxim)
- Meant to replace barcodes, RFID tags, magnetic stripes, proximity and smart cards
- Physical features: Stainless steel, waterproof, rugged, wearable, tamper responsive
- Many varieties: Real-time clock, temperature sensor, data storage, cryptographic, Java

Hardware Tokens: iButton 2

- 1-wire Interface
  - Actually, 2 wires (clock/data and ground)
  - Parasitically-powered
  - 16kbps (standard) and 142kbps (overdrive)
- Unique 64-bit ID (non-secret) for each device

iButton: DS1991 MultiKey

- 1,152 bits of non-volatile memory split into three 384-bit (48-byte) containers known as “subkeys”
- Each subkey is protected by an independent 8-byte password
- Only the correct password will grant access to the data stored within each subkey area and return the 48-bytes
- Commonly used for cashless transactions (e.g., parking meters, public transportation) and access control

iButton: DS1991 MultiKey 2

- Incorrect password will return 48-bytes of "random" data
- Marketing literature* claims:
  - "False passwords written to the DS1991 will automatically invoke a random number generator (contained in the iButton) that replies with false responses. This eliminates attempts to break security by pattern association. Conventional protection devices do not support this feature."
- “Random” data turns out to be not random at all

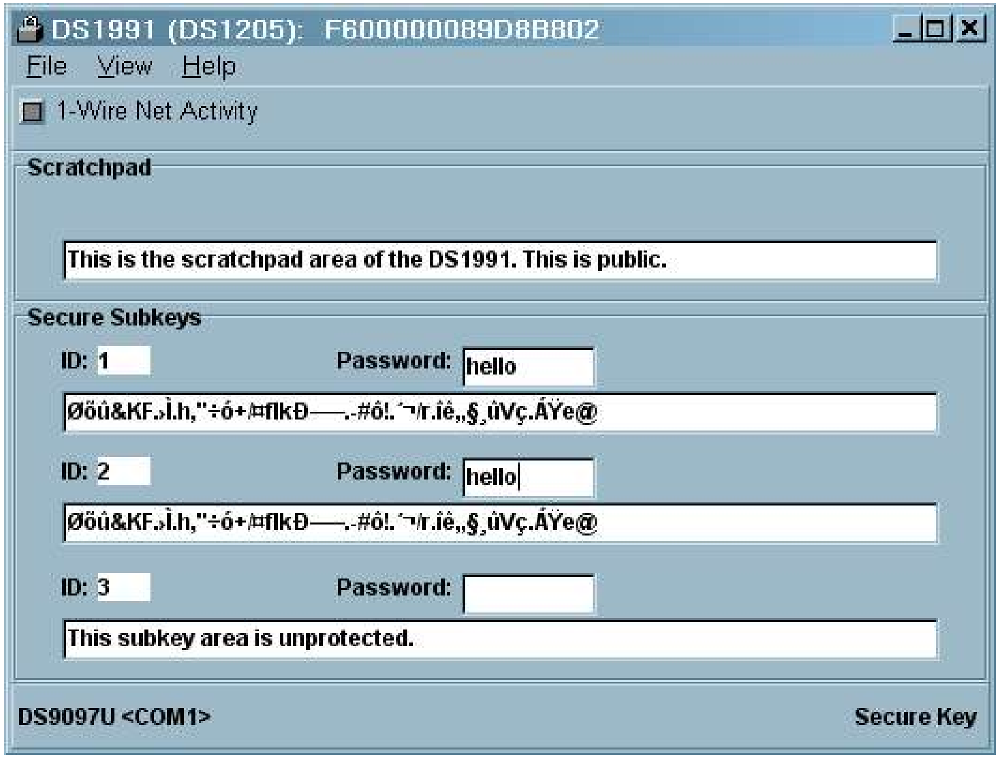
* **`www.ibutton.com/software/softauth/feature.html`**

iButton: DS1991 MultiKey 3

- Based on input password and 12kB constant block
  - Constant for all DS1991 devices
- Can precompute the 48-byte return value expected for an incorrect password
- If return value does not match, must be the correct password and subkey data

iButton: DS1991 MultiKey 4

- Initial experiments with iButton Viewer (part of free iButton-TMEX SDK) showed that "random" response is based on input password



iButton: DS1991 MultiKey 5

- For any given character (256 possibilities), a unique 48-byte response is returned from iButton
- Created application to set each single-byte password and monitor serial port for response
- Trial and error to determine how response was generated for longer length passwords

iButton: DS1991 MultiKey 6

```
A[8] = password (padded with 0x20 if < 8 bytes)
B[256][48] = constant block
C[48] = response (initialized to 0x00)

for (j = 0; j < 8; ++j) // For each character in passwd
{
  for (m = 0; m < 48; ++m) // For each byte response
  {
     if (m + j < 48) // Catch overflow above 48-bytes
     {
       k = A_j; // Perform a look-up into constant block
                // based on the jth byte of the password

       C_(m + j) ^= B_k;  // XOR the response with value
                           // of the constant block
                           // (shifted by j bytes)
     }
  }
}
```

iButton: DS1991 MultiKey 7

```
Let A = "hello   " = 68 65 6C 6C 6F 20 20 20

B_68 ('h') = D8 F6 57 6C AD DD CF 47 ...
B_65 ('e') = 03 08 DD C1 18 26 36 CF ...
B_6C ('l') = A4 33 51 D2 20 55 32 34 ...
B_6C ('l') = A4 33 51 D2 20 55 32 34 ...
B_6F ('o') = 45 E0 D3 62 45 F3 33 11 ...
B_20 (' ') = E0 2B 36 F0 6D 44 EC 9F ...
B_20 (' ') = E0 2B 36 F0 6D 44 EC 9F ...
B_20 (' ') = E0 2B 36 F0 6D 44 EC 9F ...

             D8 F6 57 6C AD DD CF 47 ...   \
                03 08 DD C1 18 26 36 ...   |
                   A4 33 51 D2 20 55 ...   |
                      A4 33 51 D2 20 ...   | XOR
                         45 E0 D3 62 ...   |
                            E0 2B 36 ...   |
                               E0 2B ...   |
                                  E0 ...   /
         C = D8 F5 FB 26 4B 46 03 9B ...
```

iButton: DS1991 MultiKey 8

- Demo: "DS1991" (boring name, sorry)
  - Looks on default COM port for DS1991
  - Given a dictionary/word file as input, calculates the expected 48-byte response returned on an incorrect password attempt
  - Attempts to read subkey area #1 using password. If correct, the protected subkey data is displayed
  - Otherwise, repeat process with the next password in the file

iButton: DS1991 MultiKey 9

```
Searching for a DS1991...

Serial ROM ID: F600000089D8B802

####

Password: 55 55 55 55 55 55 55 55   [UUUUUUUU]

Subkey Data:

53 65 63 72 65 74 20 69   [Secret i]
6E 66 6F 72 6D 61 74 69   [nformati]
6F 4E 21 40 23 20 20 20   [oN!@#   ]
20 20 20 20 20 20 20 20   [        ]
20 20 20 20 20 20 20 20   [        ]
20 20 20 20 20 20 20 20   [        ]
```

iButton: DS1991 MultiKey Recommendations

- Employ hard-to-guess passwords
  - No dictionary words, mix upper and lower case, add numbers and punctuation, etc.
- Encryption/additional obfuscation of the actual password at the application level
- Do not use a constant subkey password between all devices in an infrastructure
  - This way, if one password is discovered, won't affect others in the system

Conclusions

- Securely designing hardware is a hard problem
- Older devices have simplistic and common problems
  - "Security through obscurity" does NOT work
  - Private data is accessible on all examined devices without legitimate credentials
- Be aware of physical location

Conclusions 2

- Newer devices more difficult to attack
  - Changes threat vector - lunchtime attack likely not possible
  - Stealing key to access data with no time constraints still likely
  - Improper implementation of cryptography could leave device open
- Nothing is ever 100% secure
  - Can only attempt to make products sufficiently secure
- Learn from mistakes
  - Study history and previous attacks

Resources & Tools: USB

- Aladdin Knowledge Systems, eToken Web page, **`www.ealaddin.com/etoken`**
- SafeNet, iKey Web page, **`www.safenet-inc.com/products/ikey`**
- J. Grand (Kingpin), "Attacks on and Countermeasures for USB Hardware Token Devices," *Proceedings of the Fifth Nordic Workshop on Secure IT Systems*, 2000, **`www.grandideastudio.com/files/security/tokens/usb_hardware_token.pdf`**
- J. Grand (Kingpin), "eToken Private Information Extraction and Physical Attack," May 2000, **`www.grandideastudio.com/files/security/tokens/etoken_usb_advisory.txt`**
- Heimlich, **`www.grandideastudio.com/files/security/tokens/heimlich.zip`**
- J. Grand (Kingpin), "iKey 1000 Administrator Access and Data Compromise," July 2000, **`www.grandideastudio.com/files/security/tokens/ikey_1000_usb_advisory.txt`**
- iSpy, **`www.grandideastudio.com/files/security/tokens/ispy.zip`**

Resources & Tools: USB 2

- SnoopyPro: USB Sniffer for Windows, **`http://sourceforge.net/projects/usbsnoop`**
- Philips Semiconductor, "Security Target First Evaluation of Philips P8WE5032 Secure 8-bit Smart Card Controller," **`www.bsi.bund.de/zertifiz/zert/reporte/0153b.pdf`**

Resources & Tools: iButton

- Dallas Semiconductor/Maxim Integrated Products, iButton Web page, **`www.ibutton.com`**
- J. Grand (Kingpin), "DS1991 MultiKey iButton Dictionary Attack Vulnerability," January 2001, **`www.grandideastudio.com/files/security/tokens/ds1991_ibutton_advisory.txt`**
- DS1991 iButton Dictionary Attack Tool, **`www.grandideastudio.com/files/security/tokens/ds1991_attack.zip`**
- Dallas Semiconductor, The Java-powered-ibutton Archives, **`http://lists.dalsemi.com/maillists/java-powered-ibutton`**
- The Code Project: A Basic iButton Interface, **`www.codeproject.com/samples/ibuttoninterface.asp`**

Resources & Tools: Other

- O. Kömmerling and M. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," *USENIX Workshop on Smartcard Technology*, 1999, **www.cl.cam. ac.uk/~mgk25/sc99-tamper.pdf**
- D. Chaum, “Design Concepts for Tamper Responding Systems,” *Advances in Cryptology: Proceedings of Crypto '83*, 1984.
- A.J. Clark, “Physical Protection of Cryptographic Devices,” *Advances in Cryptology: EUROCRYPT '87*, 1988.
- J. Grand, "Practical Secure Hardware Design for Embedded Systems," *Proceedings of the 2004 Embedded Systems Conference*, 2004, **www.grandideastudio.com/files/security/hardware/ practical_secure_hardware_design.pdf**
- J. Grand, "Authentication Tokens: Balancing the Security Risks with Business Requirements," **www.grandideastudio.com/files/security/tokens/ authentication_tokens.pdf**

Thanks!

Grand Idea Studio, Inc.

`http://www.grandideastudio.com`

`joe@grandideastudio.com`