Faster PwninG Assured

- FPGAs
  - Quick Intro
- coWPAtty
  - WPA Overview
  - Precomputing tables
  - Performance
- Airbase
  - jc-aircrack
  - jc-wepcrack
  - pico-wepcrack
  - Performance
- CheapCrack
FPGAs

- Quick Intro
  - Chip with a ton of general purpose logic
    - ANDs, ORs, XORs
    - FlipFlops (Registers)
    - BlockRAM (Cache)
    - DSP48’s (ALUs)
    - DCMs (Clock Multipliers)
FPGAs

- Virtex-4 LX25
FPGAs

- Virtex-4 LX25
  - IOBs (448)
FPGAs

- Virtex-4 LX25
  - IOBs
  - Slices (10,752)
FPGAs

- Virtex-4 LX25
  - IOBs
  - Slices
  - DCMs (8)
FPGAs

- Virtex-4 LX25
  - IOBs
  - Slices
  - DCMs
  - BlockRAMs (72)
FPGAs

- Virtex-4 LX25
  - IOBs
  - Slices
  - DCMs
  - BlockRAMs
  - DSP48s (48)
FPGAs

- Virtex-4 LX25
  - IOBs
  - Slices
  - DCMs
  - BlockRAMs
  - DSP48s
  - Programmable Routing Matrix (~18 layers)
Introduction to WPA

- WiFi Protected Access

Diagram:

1. MK (Master Key) derives Supplicant (WN)
2. PMK (Pairwise Master Key) derives Authenticator (AP)
3. PMK moves key to Authenticator Server (AS)
4. PTK (Pairwise Transient Key) uses PMK and 4-way handshake
5. GTK (Group Temporal Key) uses 4-way group handshake

Keys:
- Master Key (MK)
- Pairwise Master Key (PMK)
- Pairwise Transient Key (PTK)
- Key Confirmation Key (KCK)
- Key Encryption Key (KEK)
- Temporal Key 1 (TK1)
- Temporal Key 2 (TK2)

2006 © The OpenCiphers Project
Introduction to WPA

- **PSK**
  - MK is your passphrase
  - It’s run through PBKDF2 to generate the PMK
Introduction to WPA

- **PSK**
  - MK is your passphrase
  - It’s run through PBKDF2 to generate the PMK
Introduction to WPA

- **PSK**
  - MK is your passphrase
  - It’s run through **PBKDF2** to generate the PMK
**Introduction to WPA**

- **PBKDF2**

  ```c
  unsigned char hash[32];

  t = sha1_hmac(MK, SSID, 1);
  for(i = 1; i < 4096; i++)
      t = sha1_hmac(MK, t);
  memcpy(hash, &t, 20);

  t = sha1_hmac(MK, SSID, 1);
  for(i = 1; i < 4096; i++)
      t = sha1_hmac(MK, t);
  memcpy(hash + 20, &t, 12);
  ```
**Introduction to WPA**

- **sha1_hmac**

```plaintext
sha1(MK ^ 0x5c, sha1(MK ^ 0x36, t));

sha1init(ctx);
ctx = sha1update(ctx, MK ^ 0x36);
ctx = sha1update(ctx, t);
`innersha1_ctx` = sha1final(ctx);

sha1init(ctx);
ctx = sha1update(ctx, MK ^ 0x5c);
ctx = sha1update(ctx, `innersha1_ctx`);
`outersha1_ctx` = sha1final(ctx);
```
Introduction to WPA

- sha1_hmac

```plaintext
sha1(MK ^ 0x5c, sha1(MK ^ 0x36, t));

sha1init(ctx);
ctx = sha1update(ctx, MK ^ 0x36);
ctx = sha1update(ctx, t);
inner_sha1_ctx = sha1final(ctx);

sha1init(ctx);
ctx = sha1update(ctx, MK ^ 0x5c);
ctx = sha1update(ctx, inner_sha1_ctx);
outer_sha1_ctx = sha1final(ctx);
```

You can cache some of the state to reduce the number of required SHA1’s.
Introduction to WPA

- For every possible PMK compute PTK and see if it matches the handshake captured on the network
FPGA coWPAtty

- Uses 8 SHA-1 Cores
- Uses BlockRAM to buffer the words fed to the cores
- As long as the machine is able to supply words fast enough, the SHA-1 cores will be utilized fully
FPGA coWPAtty

Computer

0
1
2
3
4
5
6
7
8
9
...
247
248
249
250
251
252
253
254
255
FPGA coWPAtty
FPGA coWPAtty

Computer

<table>
<thead>
<tr>
<th></th>
<th>SHA-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>247</td>
<td></td>
</tr>
<tr>
<td>248</td>
<td></td>
</tr>
<tr>
<td>249</td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
</tr>
<tr>
<td>251</td>
<td></td>
</tr>
<tr>
<td>252</td>
<td></td>
</tr>
<tr>
<td>253</td>
<td></td>
</tr>
<tr>
<td>254</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td></td>
</tr>
</tbody>
</table>
FPGA coWPAtty
FPGA coWPAtty

Computer

```
0
1
2
3
4
5
6
7
8
9
...
247
248
249
250
251
252
253
254
255
```

SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
### FPGA coWPAtty

![Diagram of FPGA coWPAtty](image)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>...</th>
<th>254</th>
<th>255</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
<td>SHA-1</td>
</tr>
</tbody>
</table>

**Diagram Description:**
- The diagram illustrates the FPGA coWPAtty setup.
- A computer connects to a table with entries from 0 to 255.
- Each entry is connected to a SHA-1 hash function.

---

Black Hat Briefings 2006 - August 3rd, 2006

2006 © The OpenCiphers Project
FPGA coWPAtty
FPGA coWPAtty

Computer

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
...
254
255

SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
FPGA coWPAtty
## Performance Comparison

<table>
<thead>
<tr>
<th>PC</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cowpatty</strong></td>
<td><strong>Cowpatty</strong></td>
</tr>
<tr>
<td>800MHz P3</td>
<td>LX25</td>
</tr>
<tr>
<td>~25/sec</td>
<td>~430/sec</td>
</tr>
<tr>
<td>3.6GHz P4</td>
<td>15 Cluster</td>
</tr>
<tr>
<td>~60/sec</td>
<td>~6,500/sec</td>
</tr>
<tr>
<td>AMD Opteron</td>
<td>FX60</td>
</tr>
<tr>
<td>~70/sec</td>
<td>~1,000/sec</td>
</tr>
<tr>
<td>2.16GHz IntelDuo</td>
<td></td>
</tr>
<tr>
<td>~70/sec</td>
<td></td>
</tr>
</tbody>
</table>

| Aircrack                |                         |
| 3.6GHz P4               |                         |
| ~100/sec                |                         |
Results

- Decided to compute hash tables for a 1,000,000 passphrase wordlist for the top 1,000 SSIDs

“That million word list that I fed you incorporated a 430,000 word list from Mark Burnett and Kevin Mitnick (of all people) and was made up of actual harvested passwords acquired through some google hacking. They are passwords that people have actually used. I padded it out to 1 million by adding things like websters dictionary, and other such lists, and then stripped the short word (<8 chars.) out of it.”
Results

- Took RenderMan 1 month to compute on his cluster
- Found out that his wordlist had return characters at the end of every line
- (after computing for a month)
- He sent me an email asking for help
- A 15 card cluster did it in 2 days ;-)

2006 © The OpenCiphers Project
FPGA coWPAtty

+ + = ?
Demo
Mac OS-X coWPAtty???

- /System/Library/PrivateFrameworks/Apple80211.framework/Versions/Current/Resources/airport

airport AirPort v.427.2 (427.2.0)

Supported arguments:
<snip a whole bunch of semi-normal iwconfig-like features>
-P<arg> --psk=<arg> Create PSK from specified passphrase and SSID.
The following additional arguments must be specified with this command:
--ssid=<arg> Specify SSID when
#!/usr/bin/perl

open(INFILE,"dictionary.txt");
my $start = time;
my $count = 0;
foreach (<INFILE>) {
    chop($_);
    $cmd = "airport --psk=$_ --ssid=linksys >> pmks.txt";
    system $cmd;
    $count++;
}

$elapsed = time - $start;
$perform = $count / $elapsed;
print "$count passphrases tested in $elapsed seconds: ";
print "$perform passphrases/second\n";

beetles-computer:~/Downloads beetle$ ./ghetto_pmk.pl
2253 passphrases tested in 217 seconds: 10.3824884792627 passphrases/second
Airbase
Airbase
Airbase
jc-aircrack
jc-aircrack

jc-aircrack version 2.2
Net: 00 14 bf 3a 6c ef
Tried 0 x keys
Evaluated 6656 IVs. Buffer 0% full. (0 / 166)

KEY FOUND:

[00] 11 22 33 44 55 66 77 88 99 AA BB CC

[------------------] Attack: [num found][weight]------------------]
0: [2690](5) 1: [53](3) 2: [0](13) 3: [0](11) 4: [0](4)
5: [7](4) 6: [245](11) 7: [0](11) 8: [0](4)
9: [0](15) 10: [0](5) 11: [0](5) 12: [3](13)
13: [0](4) 14: [0](4) 15: [382](4)

[-------------------] No new data in 0 searches-------------------]
jc-wepcrack
jc-wepcrack, no pico
Accelerating brute forcing
Current arch
Future arch
Pico client details
Airbase
pico-wepcrack

- FPGA Core
  - Uses 32/48 custom RC4 cores
  - Uses BlockRAM for S-Boxes
  - Will try every key between a start and end
pico-wepcrack

RC4:

```c
for(i = 0; i < 256; i++) // Initialization
    S[i] = i;

for(i = j = 0; i < 256; i++) { // KSA
    j += S[i] + K[i];
    Swap(S[i], S[j]);
}

for(i = 1, j = 0; ; i++) { // PRGA
    j += S[i];
    Swap(S[i], S[j]);
    PRGA[i - 1] = S[S[i] + S[j]];
}
```
**RC4:**

```c
for(i = 0; i < 256; i++) // Initialization
    S[i] = i;

for(i = j = 0; i < 256; i++) { // KSA
    j += S[i] + K[i]; // K is input
    Swap(S[i], S[j]);
}

for(i = 1, j = 0; ; i++) { // PRGA
    j += S[i];
    Swap(S[i], S[j]);
    PRGA[i - 1] = S[S[i] + S[j]]; // PRGA is output
}
```
pico-wepcrack
**pico-wepcrack**

- **Computer**
  - Start: 0
  - End: 16
  - PRGA: bling

- **RC4 Keygen**
  - PRGA: w00t!
  - PRGA: arg?
  - PRGA: samy is my hero
pico-wepcrack
pico-wepcrack

Computer

<table>
<thead>
<tr>
<th>Start: 0</th>
<th>End: 16</th>
<th>PRGA: bling</th>
</tr>
</thead>
</table>

RC4 Keygen

- PRGA: bling
- PRGA: meow
- PRGA: yo

RC4

RC4

RC4

RC4
pico-wepcrack

- **RC4:**

```c
for(i = 0; i < 256; i++) // Initialization
    S[i] = i; // S-Box must be reset

for(i = j = 0; i < 256; i++) { // KSA
    j += S[i] + K[i];
    Swap(S[i], S[j]);
}

for(i = 1, j = 0; ; i++) { // PRGA
    j += S[i];
    Swap(S[i], S[j]);
    PRGA[i - 1] = S[S[i] + S[j]];
}
```
pico-wepcrack
pico-wepcrack
pico-wepcrack
pico-wepcrack
pico-wepcrack
pico-wepcrack

Computer

S

Read: 2 -> XX13

RC4

S

Read: 57 -> XX7A

RC4

S

Read: 3C -> XX25

RC4
pico-wepcrack

Computer

S
Write: 02 -> 0213
RC4

S
Write: 57 -> 577A
RC4

S
Write: 3C -> 3C25
RC4
pico-wepcrack

00: 0073
01: 019B
02: 0296
03: 03c2
04: 0431
05: 05df
06: 0609
07: 078c

RC4:

for(i = 0; i < 256; i++) // Init
    S[i] = i; // S-Box must be reset

for(i = j = 0; i < 256; i++) {
    j += S[i] + K[i];
    Swap(S[i], S[j]);
}

for(i = 1, j = 0; ; i++) {
    j += S[i];
    Swap(S[i], S[j]);
    PRGA[i - 1] = S[S[i] + S[j]];
}

.....
pico-wepcrack

Computer

S → Read: 2 -> 13XX → RC4

S → Read: 57 -> 7AXX → RC4

S → Read: 3C -> 25XX → RC4
pico-wepcrack

Computer

S  Write: 02 -> 1302  RC4

S  Write: 57 -> 7A57  RC4

S  Write: 3C -> 253C  RC4
pico-wepcrack
Demo
# Performance Comparison

## PC
- jc-wepcrack
  - 1.25GHz G4  ~150,000/sec
  - 3.6GHz P4  ~300,000/sec

## FPGA
- pico-wepcrack
  - LX25  ~9,000,000/sec
  - 15 Cluster  ~135,000,000/sec
  - FX60  ~18,000,000/sec
Everyone remembers DeepCrack, right?

- In 1998, EFF built a dedicated hardware DES cracking machine
- Cost: ~$210,000 USD, budgeted for ~$250,000
- In 1998 dollars -- today that’s ~$243,341.90, budgeted for ~$289,692.74
- Let’s just round up and say it would cost ~$250,000, budgeted for ~$300,000
- But, you’d get more for your money, due to Moore’s Law and advances in the technology in intervening eight years
CheapCrack

- The hardware consisted of:
  - 1536 custom ASICs
  - 64 chips per board, 12 boards per chassis, 2 chassis
  - Each chip has 24 “search units”
  - With this total configuration and implementation, they could search 92,160,000,000 keys/second
  - The average search time: 4.524 days

- EFF documented the whole thing in a book
  - Included VHDL source code for the chip designs
  - Included circuit board diagrams
  - Included source code to OCR the book’s content
CheapCrack

- We started asking the question: “what have the last eight years of advances in Moore’s Law bought us at the low end of custom hardware?”
- That is, programmable hardware -- FPGAs
- Can we do as well as EFF did in 1998 with:
  - Less money: just $10,000?
  - Cheaper hardware: COTS FPGAs?
  - Really cheap hardware: Low-cost COTS FPGAs already built into hobbyist developer kits?
  - Shoestring parallelism: getting a bunch of boards to run our design in parallel with cheap interconnect?
CheapCrack

- **Answer:**
  - We’re not sure yet
  - But we think so

- **Why?**
  - The design used in DeepCrack was pretty simple
  - We might be able to do better
  - FPGAs are still slower than custom ASICs…
  - But today’s FPGAs might rival circa 1998 custom ASIC designs in some cases

- **When will we know?**
  - This is a sneak peek, we plan to release at ToorCon
Conclusion

- Get an FPGA and start cracking!
- Make use of your hardware to break crypto
- Add cool ASCII Matrix FX when you can :-)
- Choose bad passwords (please!)
- Use DES (pretty please!)
Hardware Used

- Pico E-12
  - Compact Flash
  - 64 MB Flash
  - 128 MB SDRAM
  - Gigabit Ethernet
  - Optional 450MHz PowerPC 405
Hardware Used

- Pico E-12 Super Cluster
  - 15 - E-12’s
  - 2 - 2.8GHz Pentium 4’s
  - 2 - 120GB HDD
  - 2 - DVD-RW
  - 550 Watt Power Supply
Hardware Used

- Digilent S3BOARDs
  - Xilinx Spartan-3 XC3S1000 FPGA
  - 1 million gate equivalent
  - 2 MB Platform Flash
  - 1 MB SRAM
  - Lots of simple I/O
  - ~$150/board
  - digilentinc.com
Greetz

- Johny Cache (airbase/jc-wepcrack/jc-aircrack)
- Josh Wright (cowpatty)
- RenderMan (pmk hashtable monkey)
- Beetle (ghettopmk and CheapCrack funding!)
- Viewers Like You!
Questions?

- We’ll give you a free set of hash tables!

- David Hulton
  - dhulton@openciphers.org
  - http://www.openciphers.org
  - http://www.picocomputing.com

- Dan Moniz
  - dnm@pobox.com
  - http://pobox.com/~dnm/
  - http://hundrad.org/