ShmooCon

Moose? I don't need no stinkin moose

Hacking the Airwaves with FPGAs

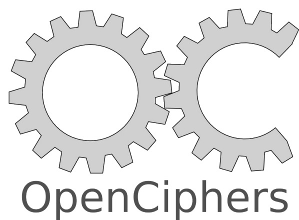
ShmooCon 2007

David Hulton <david@toorcon.org>

Chairman, ToorCon

Security R&D, Pico Computing, Inc.

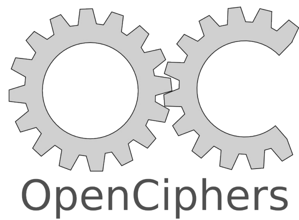
Researcher, The OpenCiphers Project



Overview

- FPGAs – Quick Intro
- Cracking
  - pico-wepcrack – WEP (Brute Force)
  - jc-aircrack – WEP (FMS) *NEW*
  - coWPAtty – WPA
  - VileFault – Mac OS-X FileVault *NEW*
  - btpincrack – Bluetooth Authentication *NEW*
  - Works in Progress *NEW*
- Conclusions

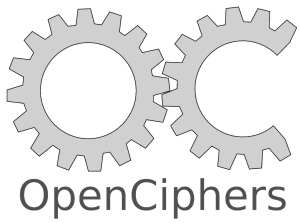
2007 © The OpenCiphers Project



FPGAs

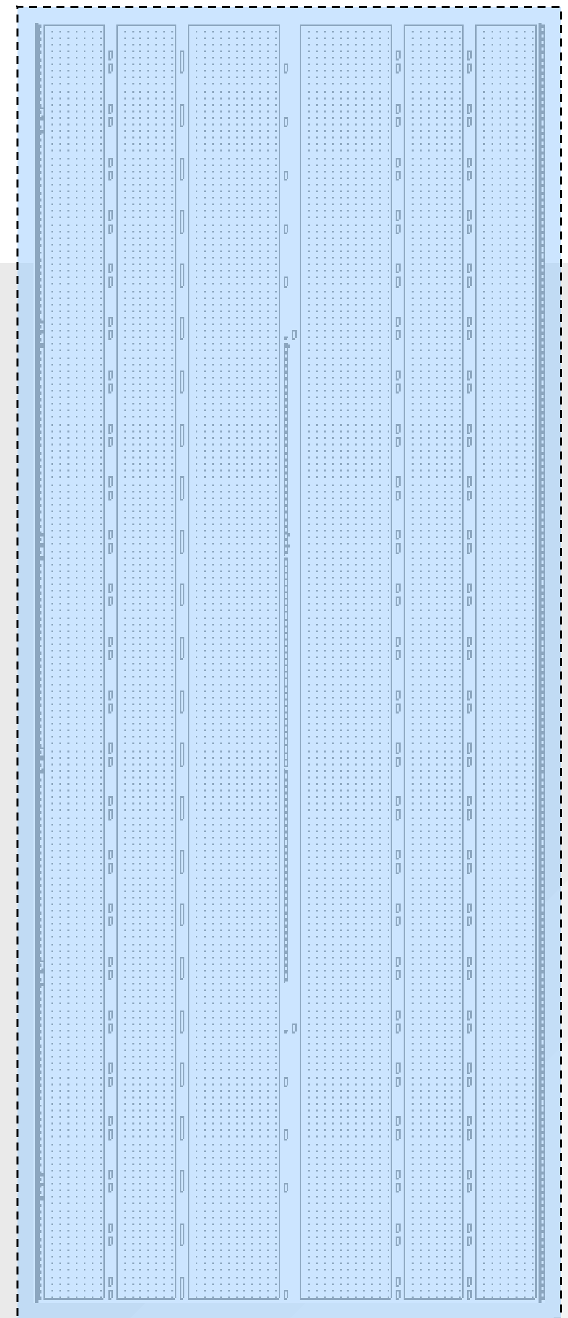
- Quick Intro
  - Chip with a ton of general purpose logic
    - ANDs, ORs, XORs
    - FlipFlops (Registers)
    - BlockRAM (Cache)
    - DSP48's (ALUs)
    - DCMs (Clock Multipliers)

2007 © The OpenCiphers Project

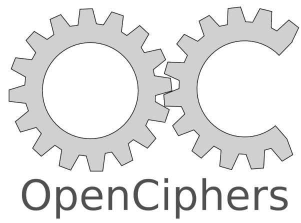


FPGAs

- Virtex-4 LX25

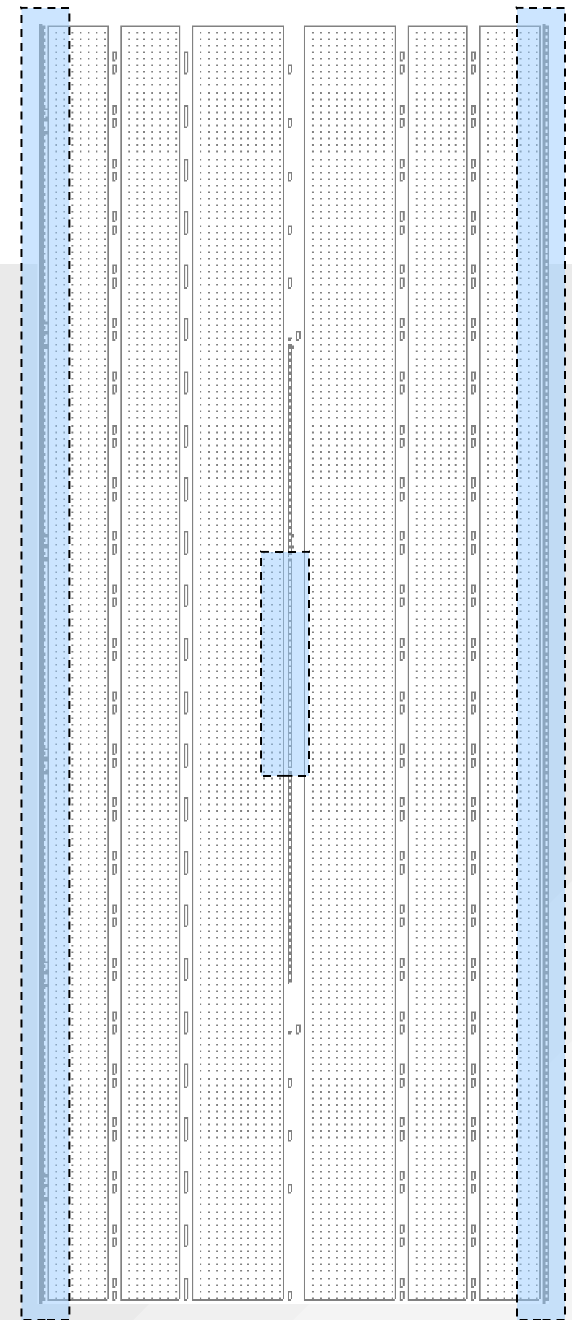


2007 © The OpenCiphers Project

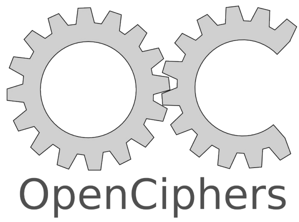


FPGAs

- Virtex-4 LX25
  - IOBs (448)

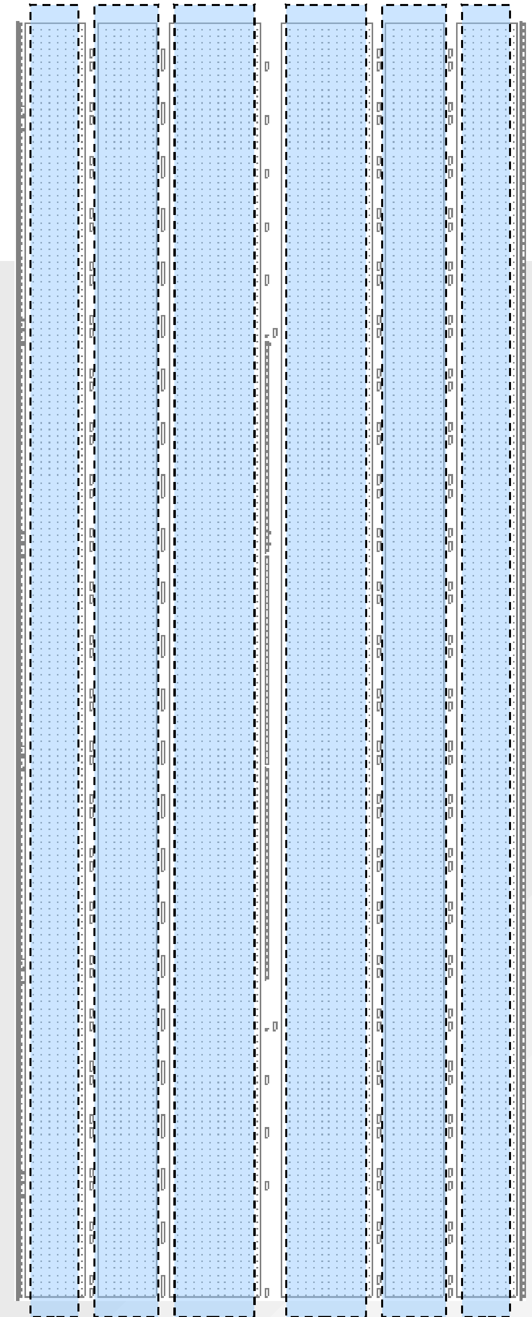


2007 © The OpenCiphers Project

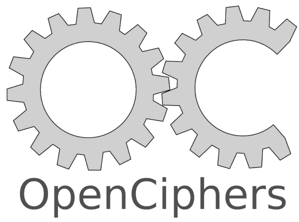


FPGAs

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

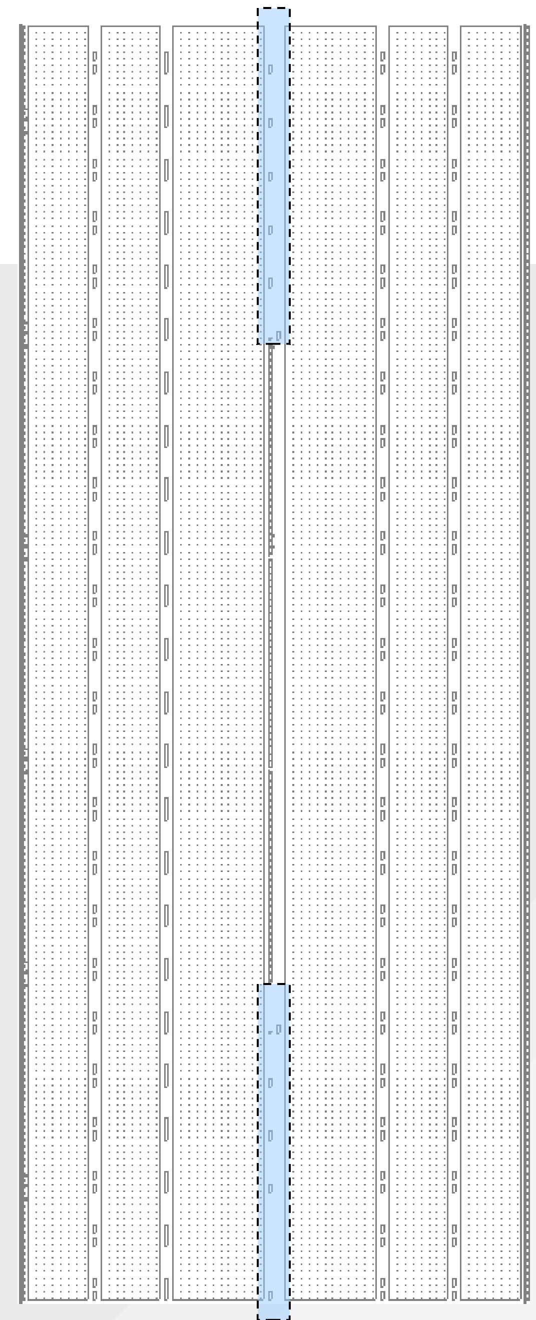


2007 © The OpenCiphers Project

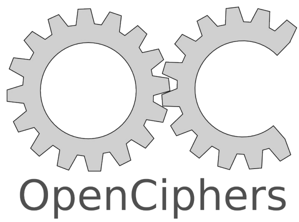


FPGAs

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

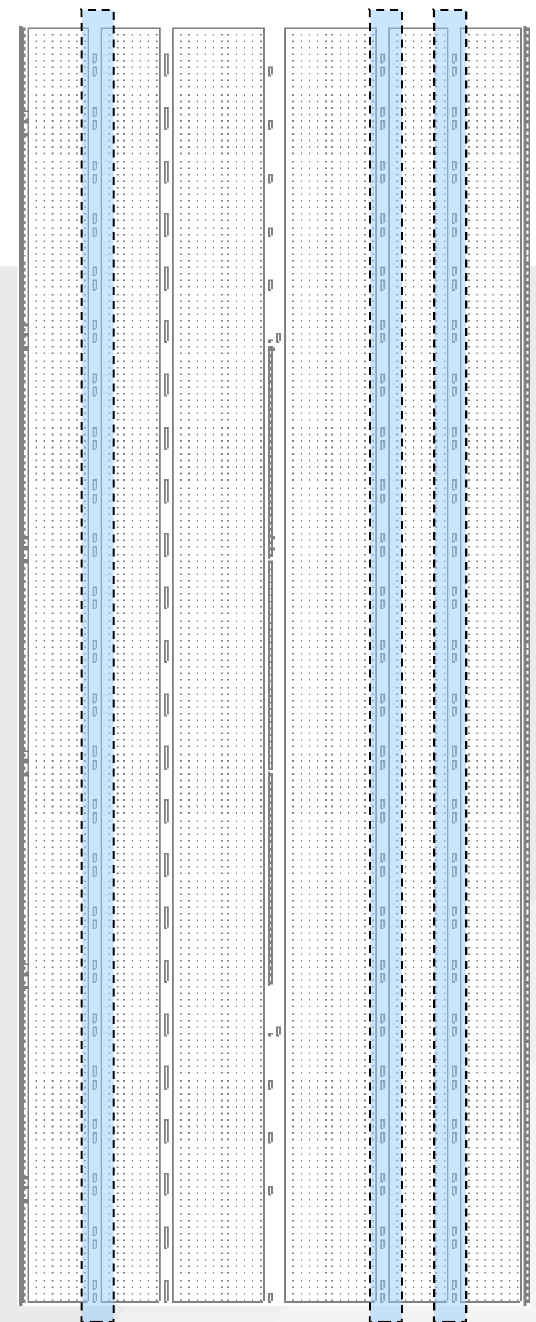


2007 © The OpenCiphers Project

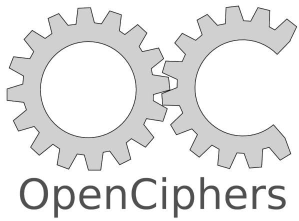


FPGAs

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

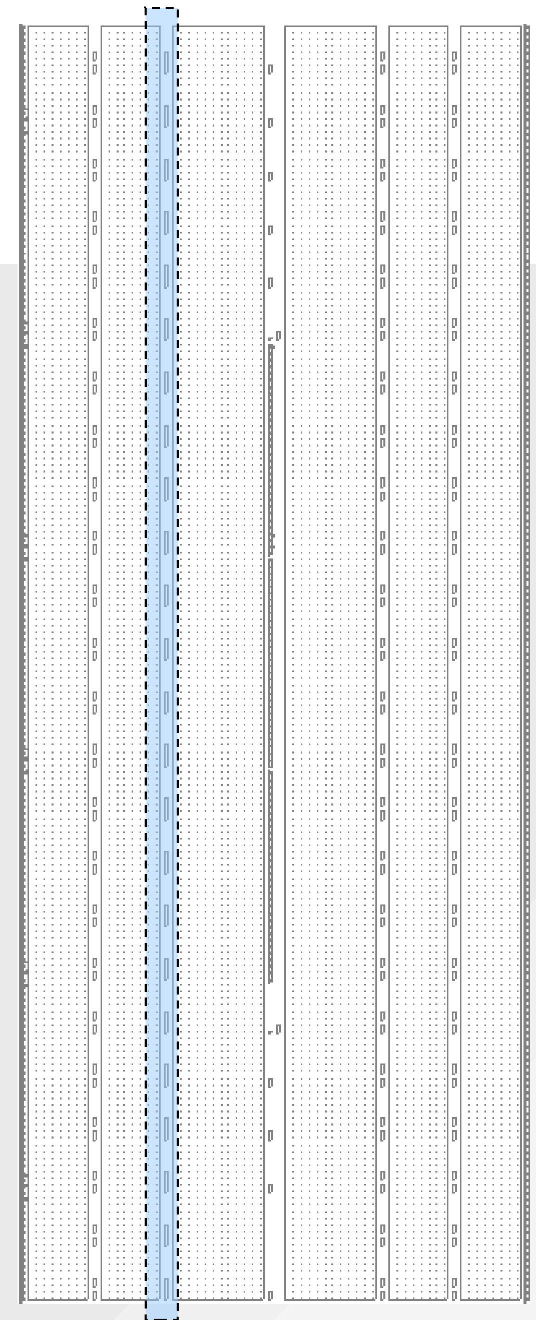


2007 © The OpenCiphers Project

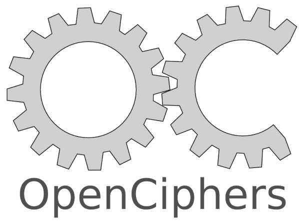


FPGAs

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

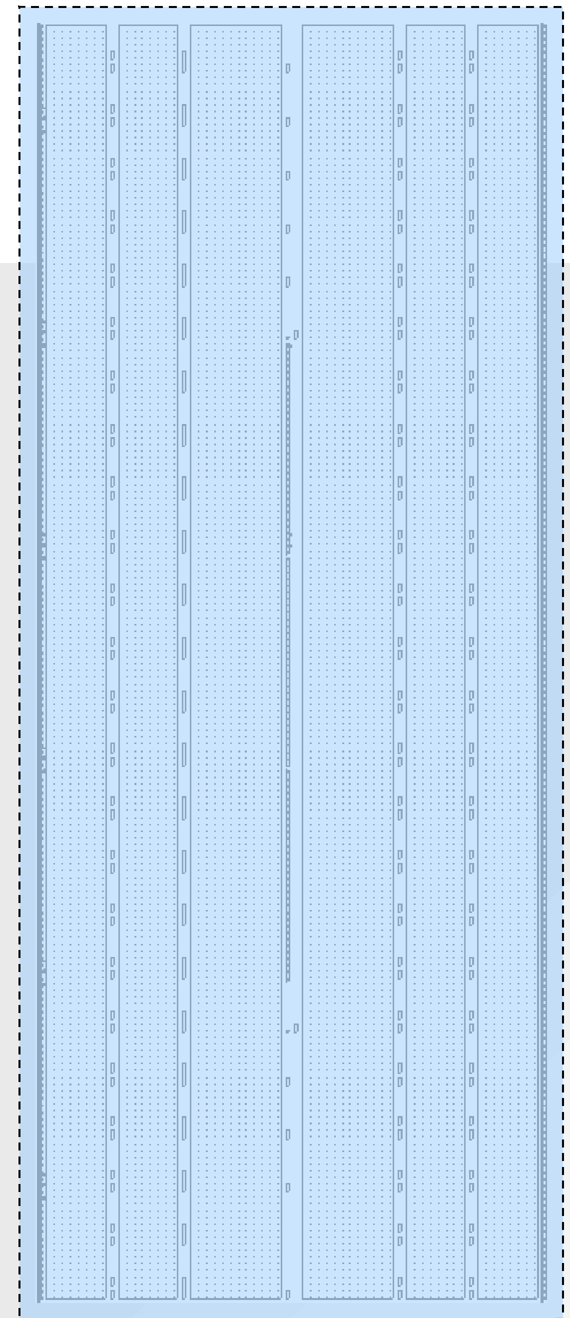


2007 © The OpenCiphers Project

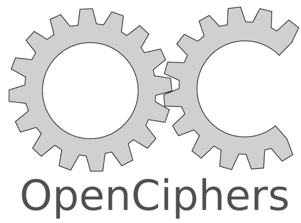


FPGAs

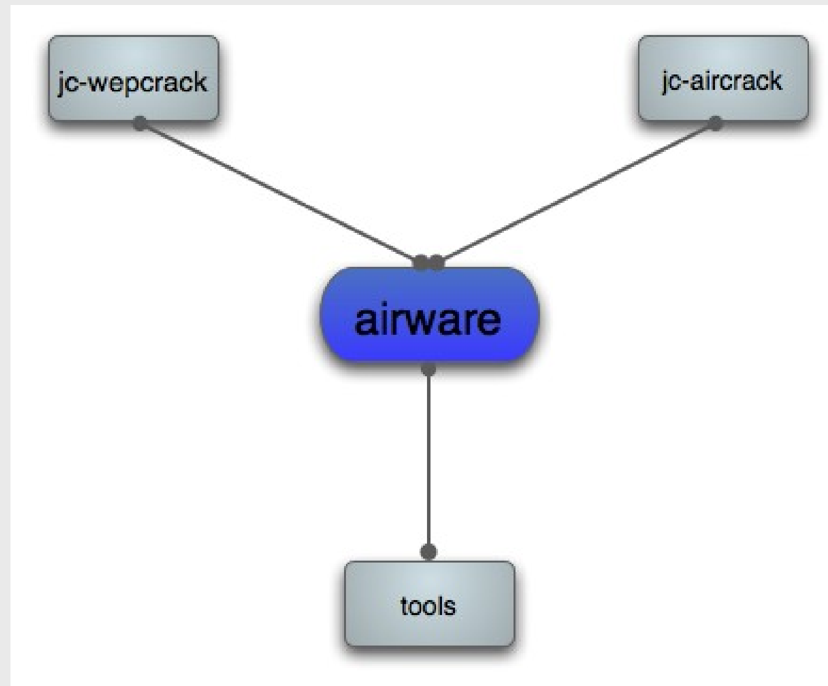
- Virtex-4 LX25
  - IOBs
  - Slices
  - DCMs
  - BlockRAMs
  - DSP48s
  - Programmable Routing Matrix (~18 layers)



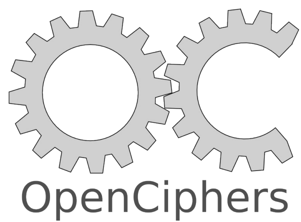
2007 © The OpenCiphers Project



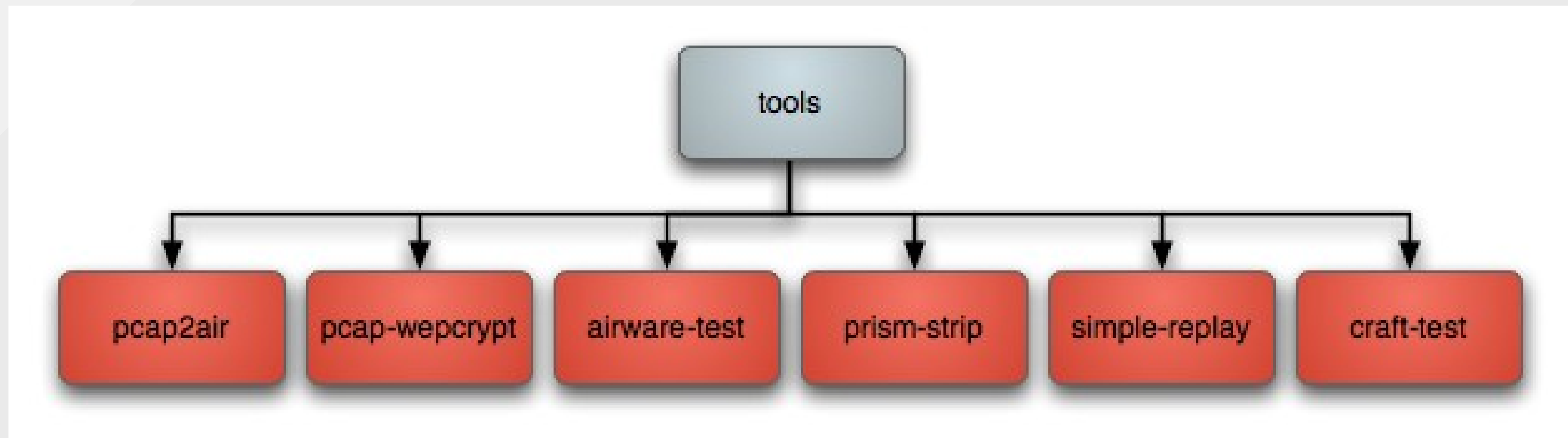
Airbase



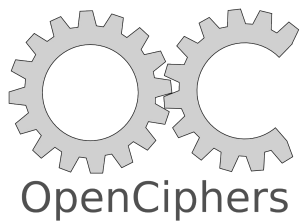
2007 © The OpenCiphers Project



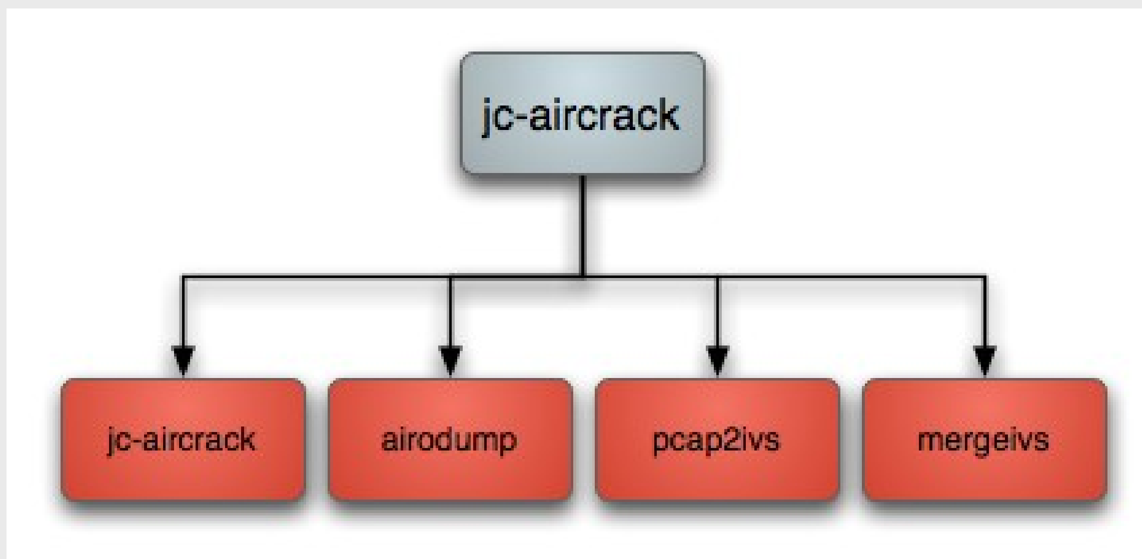
Airbase



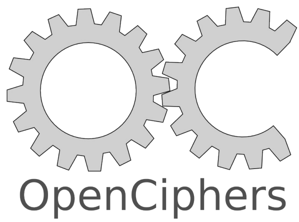
2007 © The OpenCiphers Project



jc-aircrack



2007 © The OpenCiphers Project

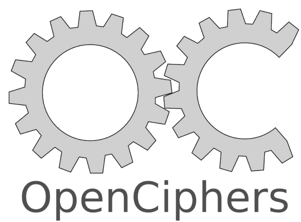


jc-aircrack

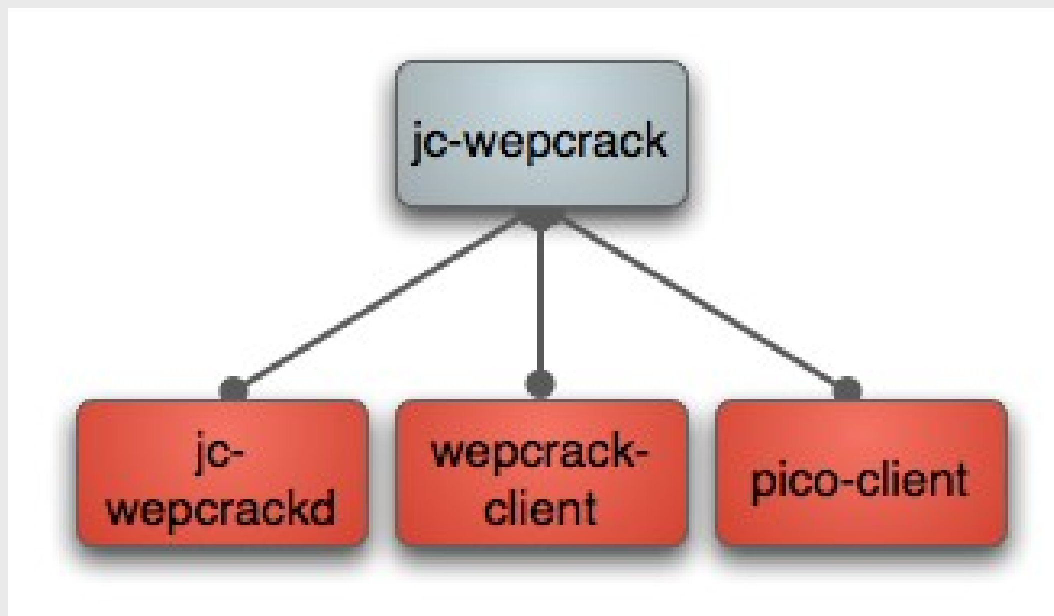
Default

```
                     jc-aircrack version 2.2
                     Net: 00 14 bf 3a 6c ef
                        Tried  0 x  keys
          Evaluated  6656 IVs.  Buffer   0% full. (0 / 166)
KB   depth       Fudge-Factor: 2.  Autonomous mode: Disabled.
 0    0/  1   [00]+---------------KEY FOUND---------------+ 21)[D4](  21)
 1    0/  1   [11]|                                       | 21)[CF](  20)
 2    0/  1   [22]| 00 11 22 33 44 55 66 77 88 99 AA BB CC | 20)[07](  16)
 3    0/  1   [33]|                                       | 20)[EA](  20)
 4    0/  1   [44]*---------------------------------------* 22)[10](  21)
 5    0/  1   [55](  80)[56](  37)[B9](  30)[53](  26)[90](  23)[FE](  20)
 6    0/  1   [66](  85)[12](  35)[5E](  24)[13](  22)[54](  20)[BC](  19)
 7    0/  1   [77]( 117)[AA](  27)[AF](  25)[5D](  25)[9E](  24)[01](  22)
 8    0/  1   [88]( 101)[89](  33)[47](  31)[A1](  26)[D0](  25)[53](  24)
 9    0/  1   [99]( 152)[59](  25)[C7](  22)[24](  21)[DB](  21)[B8](  21)
10    0/  6   [AA](  47)[E9](  31)[EF](  26)[0F](  25)[73](  25)[A0](  24)


[----------------------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------------------------------]
```

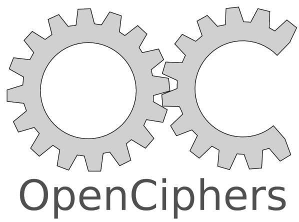

2007 © The OpenCiphers Project



jc-wepcrack



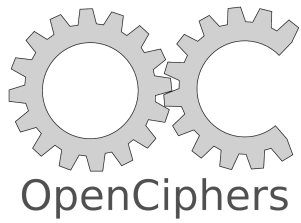
2007 © The OpenCiphers Project



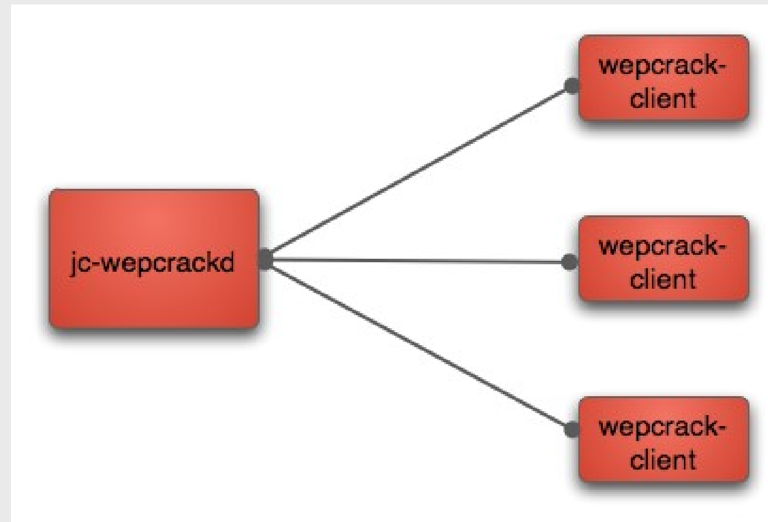
jc-wepcrack

```
Default
New  Bookmarks  Configure  Customize  Close
-------------------jc-wepcrack 1.1.0 by Johnny Cache-------------------
| Network: 00-30-bd-c0-38-9a     KeySize: 40     Status Running        |
|----------------------------------------------------------------------|
| Total Run Time: 0d 0h 0m 15s   Total Compute Time: 0d 0h 0m 0s       |
| Chunksize: 30   Chunks currently out: 0      Current Stragglers: 0   |
| Percent Complete:  0.0000   Straggler Threshold:  0d 2h 0m 0s        |
|----------------------------------------------------------------------|
| Next iKey: 00:00:00:00:00:                                           |
|----------------------------------------------------------------------|
| Total KeyChunks:       04:00:                                        |
| KeyChunks checked out: 00:00:                                        |
| KeyChunks checked in:  00:00:                                        |
|----------------------------------------------------------------------|
1: Default  2: Default
```


2007 © The OpenCiphers Project



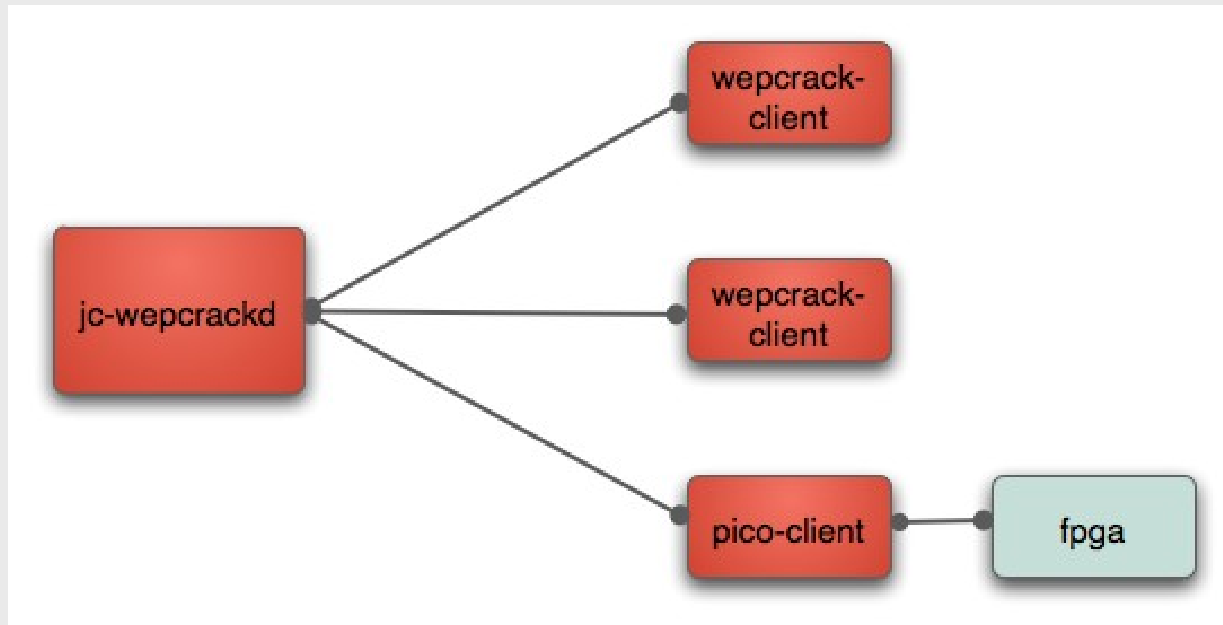
jc-wepcrack, no pico



2007 © The OpenCiphers Project

OpenCiphers

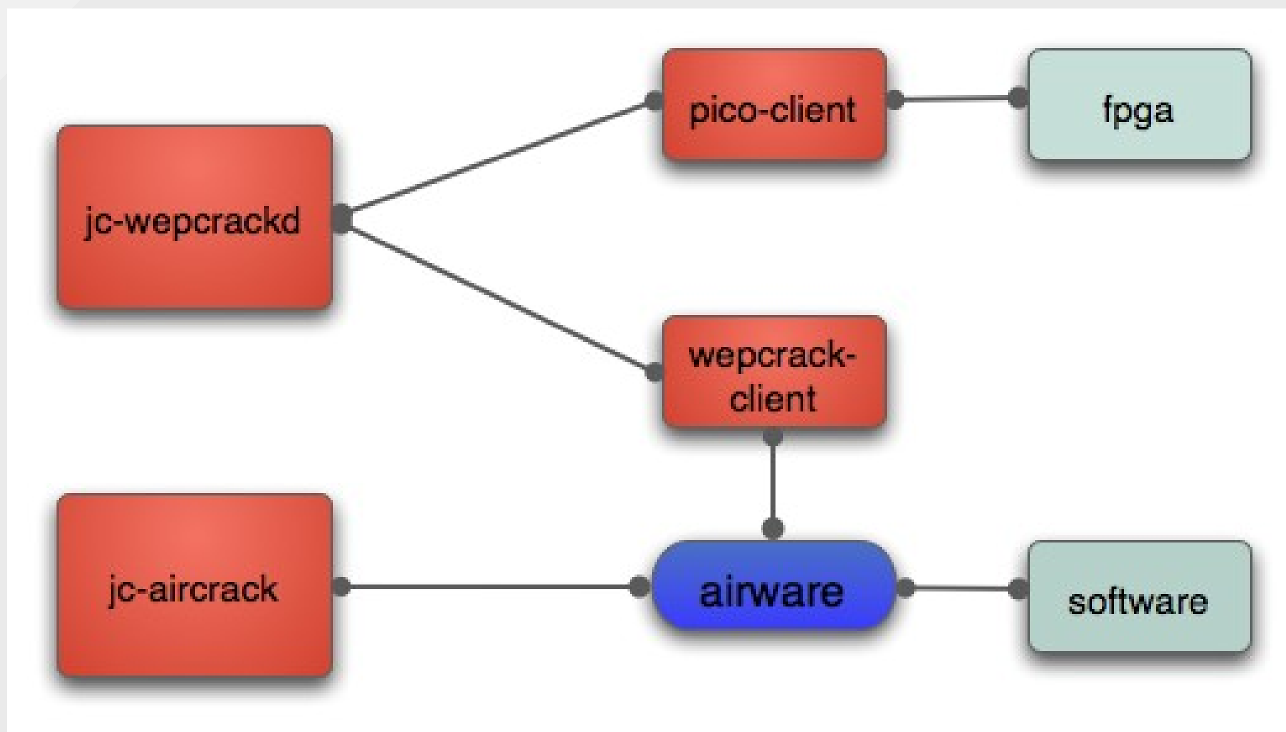
Accelerating brute forcing



2007 © The OpenCiphers Project

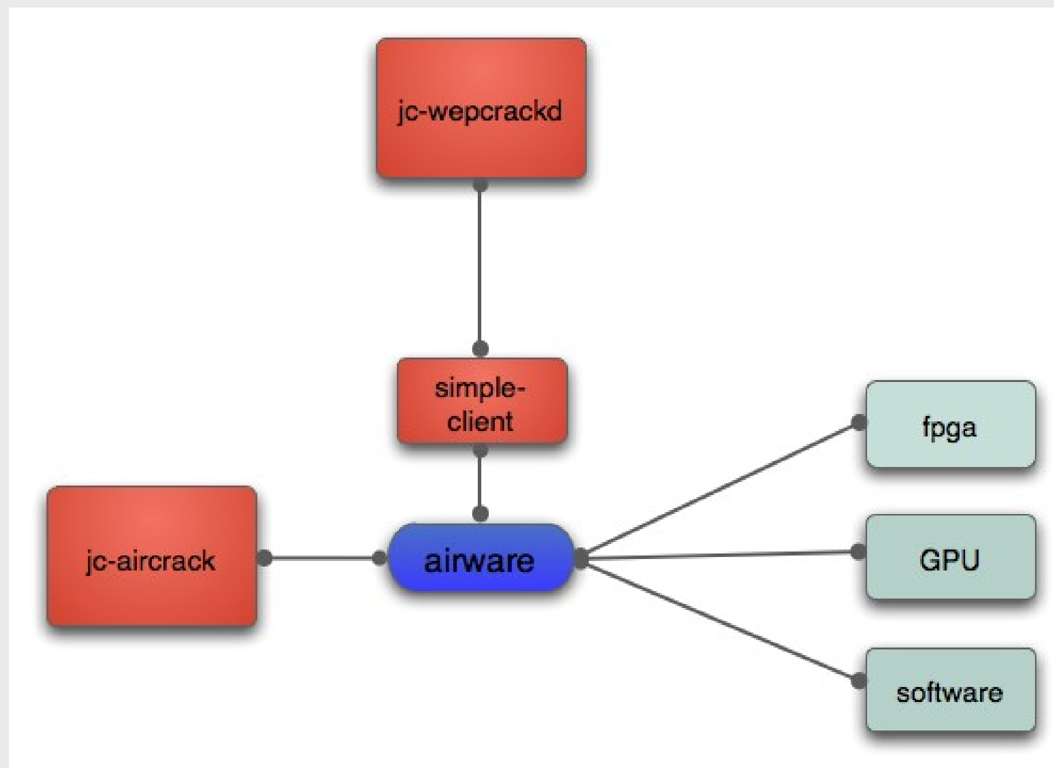
OpenCiphers

Current arch



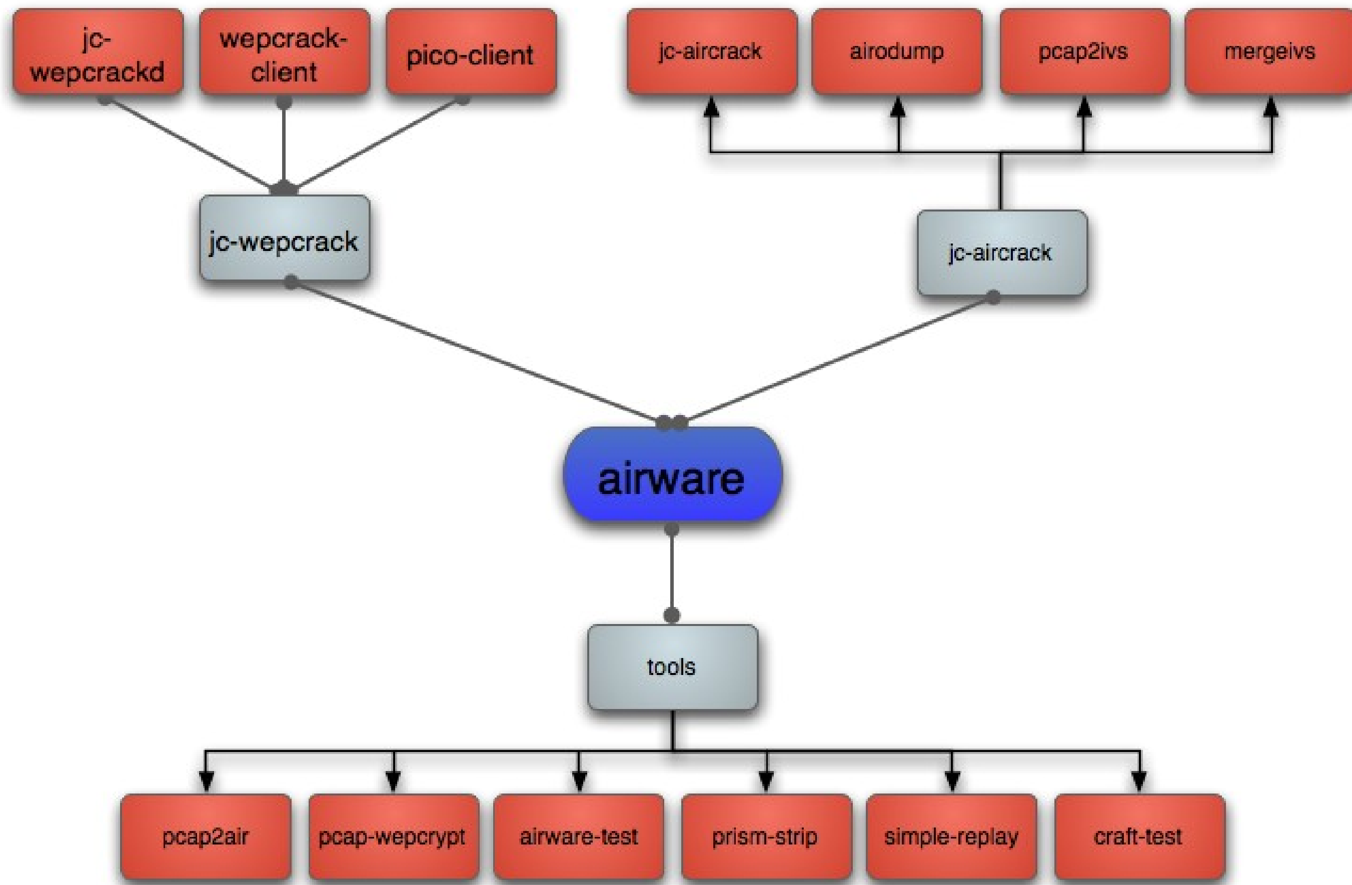
2007 © The OpenCiphers Project

Future arch

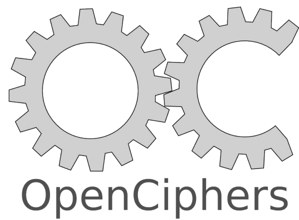


2007 © The OpenCiphers Project

Airbase



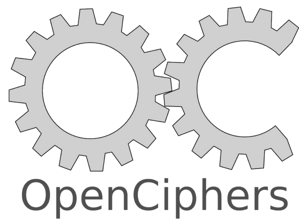
2007 © The OpenCiphers Project



pico-wepcrack

- WEP Cracking Basics
  - You sniff a packet
  - Determine the RC4 PRGA output by xor'ing the first few (ciphertext) bytes with the standard SNAP header (known plaintext)
  - Run every possible key through RC4 and compare PRGA output with output derived from the packet
  - The key that has a matching PRGA is the correct key

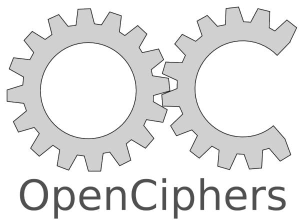
2007 © The OpenCiphers Project



pico-wepcrack

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

2007 © The OpenCiphers Project

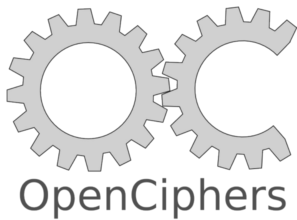


pico-wepcrack

- RC4:

```
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]];
}
```


2007 © The OpenCiphers Project

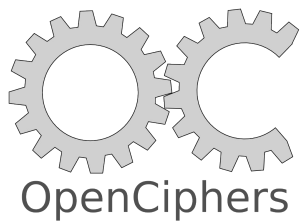


pico-wepcrack

- RC4:

```
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
}
```

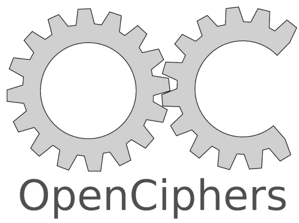

2007 © The OpenCiphers Project



pico-wepcrack

- Direct mapping from K -> PRGA
- But we need PRGA -> K
- Search all of K for matching PRGA

2007 © The OpenCiphers Project



pico-wepcrack

Computer

Start: 0
End: 16
PRGA: 0xf00d

RC4
Keygen

Key: 0

Key: 1

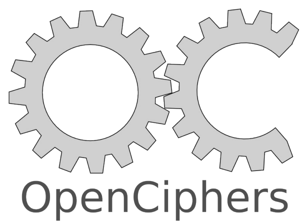
Key: 2

RC4

RC4

RC4

2007 © The OpenCiphers Project



pico-wepcrack

Computer

Start: 0
End: 16
PRGA: 0xf00d

RC4
Keygen

PRGA: 0xd00d

PRGA: 0xbeef

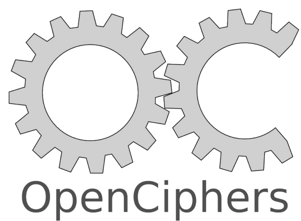
PRGA: 0xdead

RC4

RC4

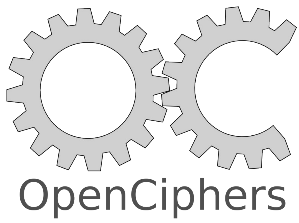
RC4

2007 © The OpenCiphers Project



pico-wepcrack

2007 © The OpenCiphers Project



pico-wepcrack

Computer

Start: 0
End: 16
PRGA: 0xf00d

RC4
Keygen

PRGA: 0xbead

PRGA: 0xcafe

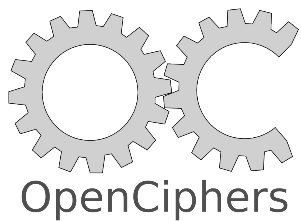
PRGA: 0xf00d

RC4

RC4

RC4

2007 © The OpenCiphers Project



pico-wepcrack

Computer

Start: 0
End: 16
PRGA: 0xf00d

RC4
Keygen

PRGA: 0xbead

PRGA: 0xcafe

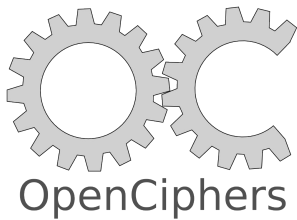
PRGA: 0xf00d

RC4

RC4

RC4

2007 © The OpenCiphers Project

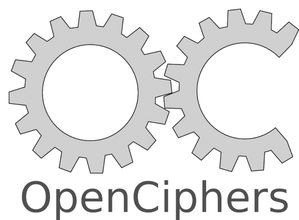


pico-wepcrack

- RC4:
  - Initialization can happen in parallel with KSA & PRGA

```
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]];
}
```


2007 © The OpenCiphers Project



Performance Comparison

PC

jc-wepcrack

1.25GHz G4	~150,000/sec
3.6GHz P4	~300,000/sec

~42 Days to break 40-bit

PS3

cbe-client (Thx HDM!)

1 SPU 3.2GHz	~241,000/sec
6 SPU 3.2GHz	~1,446,000/sec

~8.8 Days to break 40-bit

FPGA

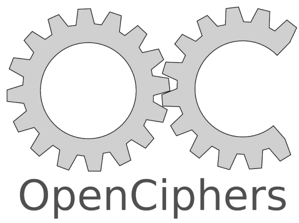
pico-wepcrack

LX25	~12,000,000/sec
15 Cluster	~180,000,000/sec

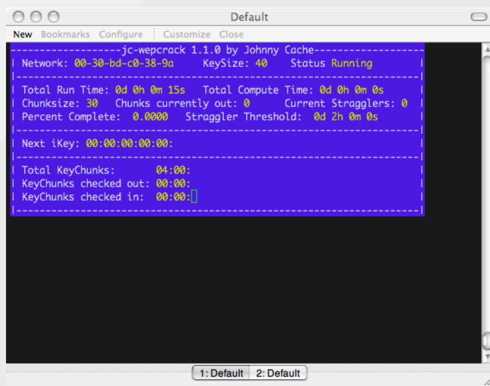
~25 Hours to break 40-bit

100 Minutes on a 15 Cluster

2007 © The OpenCiphers Project



pico-wepcrack

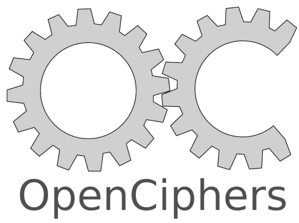


+

+

= ?

2007 © The OpenCiphers Project



jc-aircrack

```
Default
                    jc-aircrack version 2.2
                    Net: 00 14 bf 3a 6c ef
                       Tried  0 x  keys
          Evaluated  6656 IVs.  Buffer   0% full. (0 / 166)
KB   depth     Fudge-Factor: 2.  Autonomous mode: Disabled.
 0    0/  1   [00]+----------------KEY FOUND---------------+ 21)[D4]( 21)
 1    0/  1   [11]|                                        | 21)[CF]( 20)
 2    0/  1   [22]| 00 11 22 33 44 55 66 77 88 99 AA BB CC | 20)[07]( 16)
 3    0/  1   [33]|                                        | 20)[EA]( 20)
 4    0/  1   [44]*----------------------------------------* 22)[10]( 21)
 5    0/  1   [55](  80)[56](  37)[B9](  30)[53](  26)[90]( 23)[FE]( 20)
 6    0/  1   [66](  85)[12](  35)[5E](  24)[13](  22)[54]( 20)[BC]( 19)
 7    0/  1   [77]( 117)[AA](  27)[AF](  25)[5D](  25)[9E]( 24)[01]( 22)
 8    0/  1   [88]( 101)[89](  33)[47](  31)[A1](  26)[D0]( 25)[53]( 24)
 9    0/  1   [99]( 152)[59](  25)[C7](  22)[24](  21)[DB]( 21)[B8]( 21)
10    0/  6   [AA](  47)[E9](  31)[EF](  26)[0F](  25)[73]( 25)[A0]( 24)

[----------------------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-------------------------]
```

+

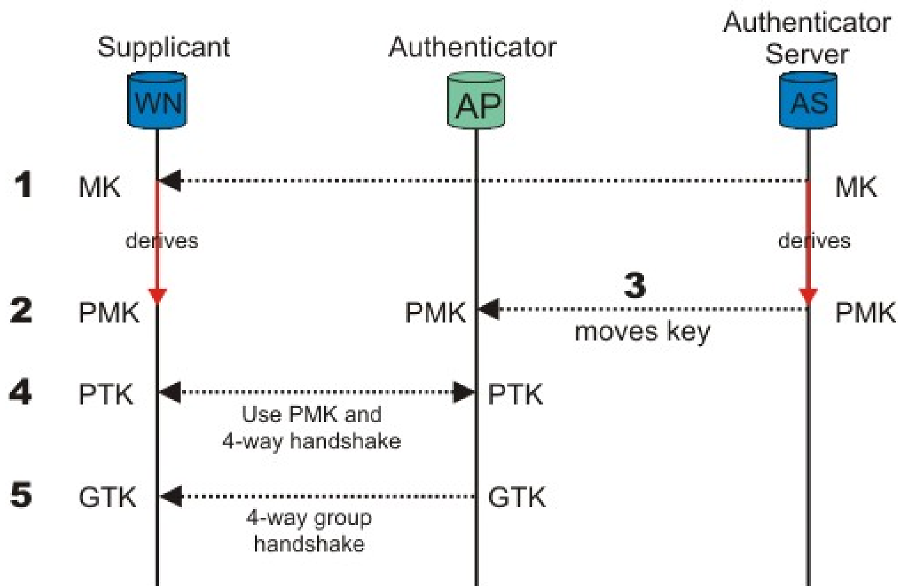
= ?

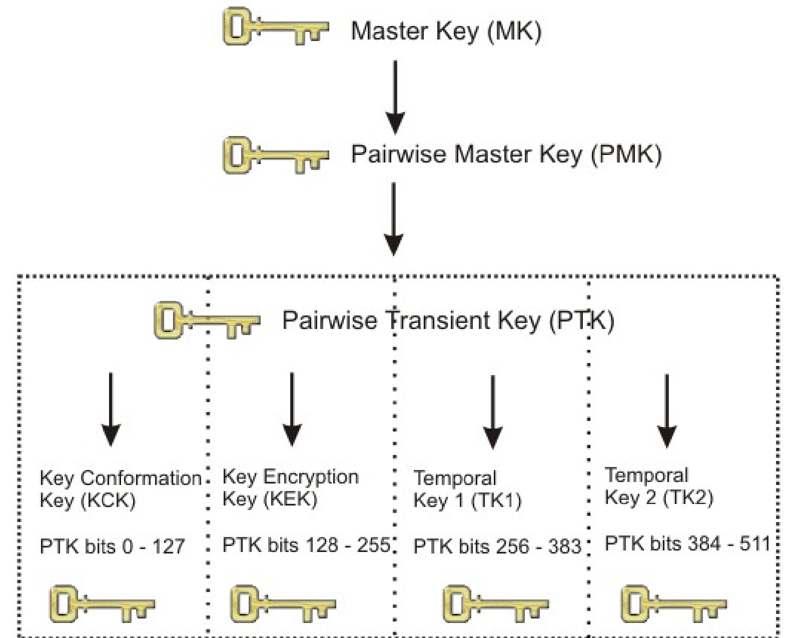
2007 © The OpenCiphers Project

Demo

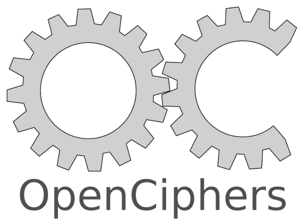
Introduction to WPA

- WiFi Protected Access



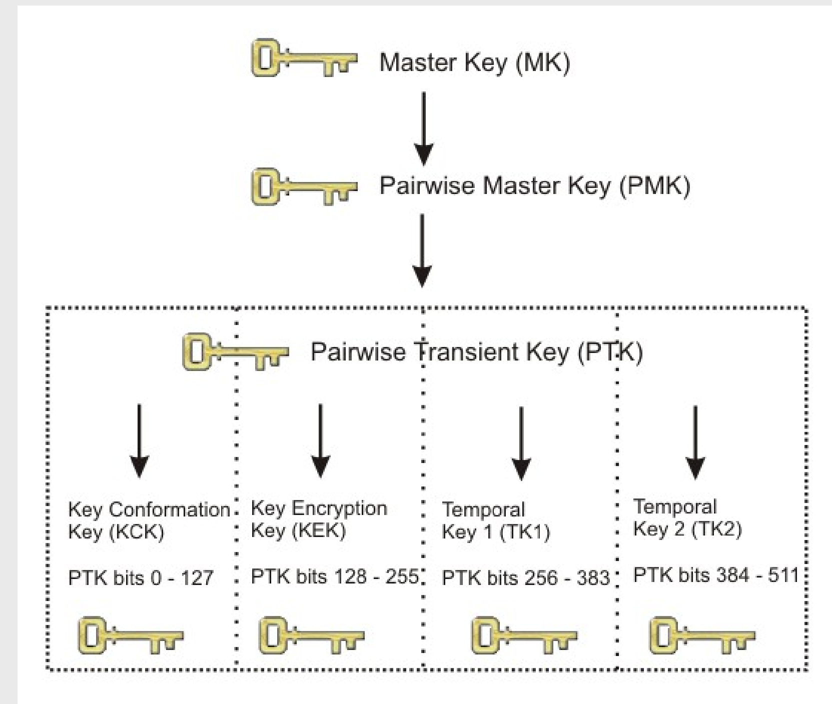


2007 © The OpenCiphers Project



Introduction to WPA

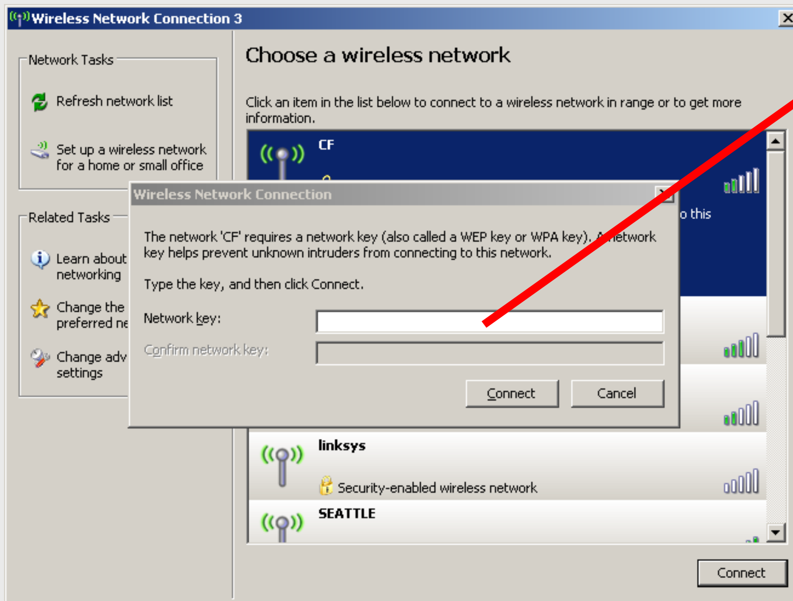
- PSK
  - MK is your passphrase
  - It's run through PBKDF2 to generate the PMK

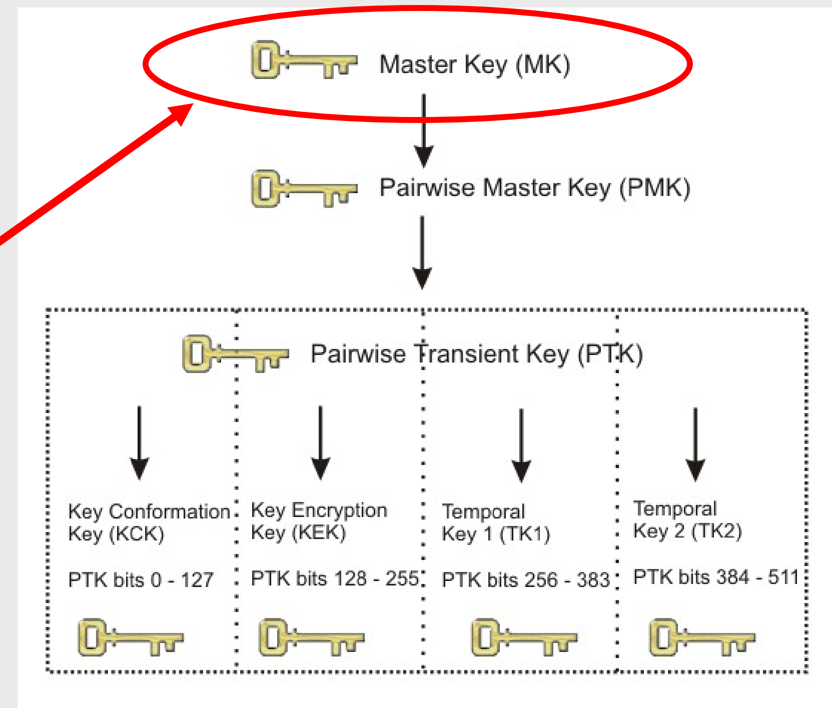


2007 © The OpenCiphers Project

Introduction to WPA

- PSK
  - MK is your passphrase
  - It's run through PBKDF2 to generate the PMK



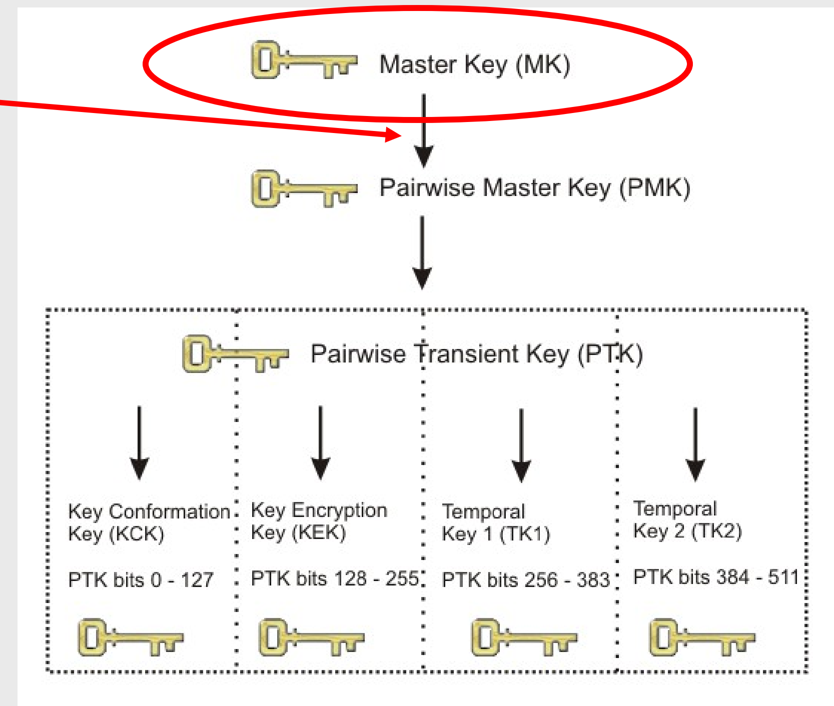


2007 © The OpenCiphers Project

OpenCiphers

Introduction to WPA

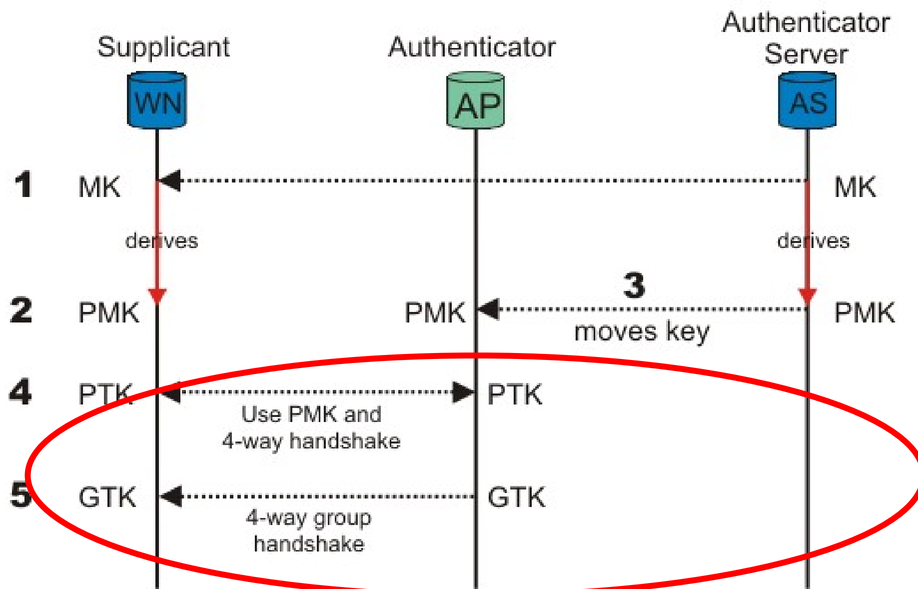
- PSK
  - MK is your passphrase
  - It's run through PBKDF2 to generate the PMK

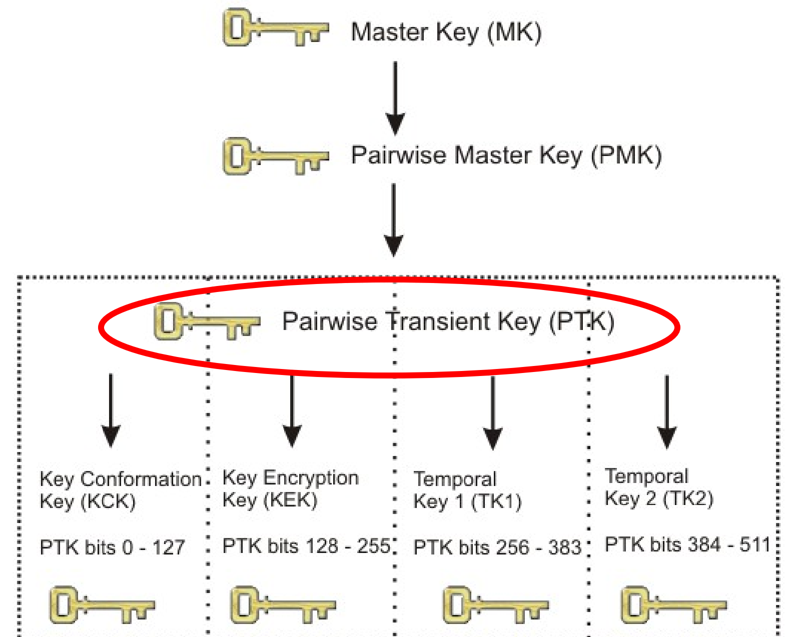


2007 © The OpenCiphers Project

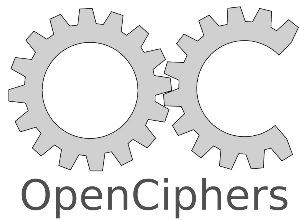
Introduction to WPA

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





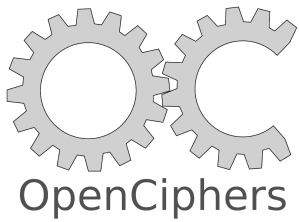
2007 © The OpenCiphers Project



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

2007 © The OpenCiphers Project



Performance Comparison

PC

Cowpatty

800MHz P3	~25/sec
3.6GHz P4	~60/sec
AMD Opteron	~70/sec
2.16GHz IntelDuo	~70/sec

Aircrack

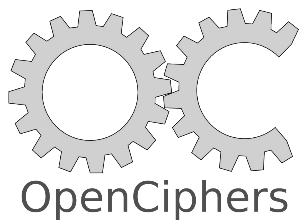
3.6GHz P4	~100/sec

FPGA

Cowpatty

LX25	~430/sec
15 Cluster	~6,500/sec
LX50	~650/sec

2007 © The OpenCiphers Project

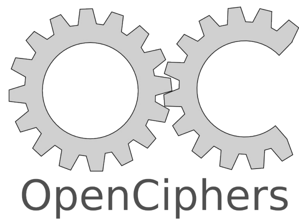


Results

- 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.”

2007 © The OpenCiphers Project



Results

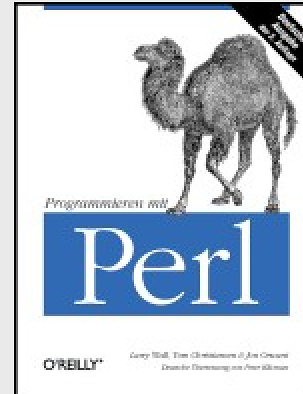
- Finally have the 40GB WPA tables on the tubes
- Thanks Shmoo! (3ricJ & Holt!)
- Check the Torrent trackers for seeds

2007 © The OpenCiphers Project

FPGA coWPAtty

+

+



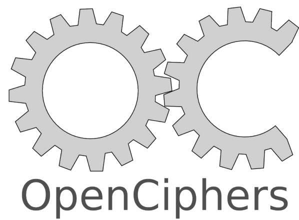
= ?

2007 © The OpenCiphers Project

Demo

Demo

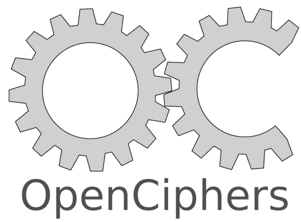
Oh, did I mention windows support?



VileFault

- “FileVault secures your home directory by encrypting its entire contents using the Advanced Encryption Standard with 128-bit keys. This high-performance algorithm automatically encrypts and decrypts in real time, so you don’t even know it’s happening.”

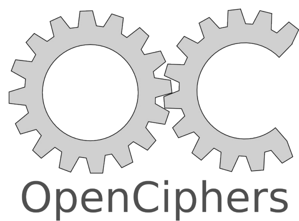
2007 © The OpenCiphers Project



VileFault

- We wanted to know what was happening

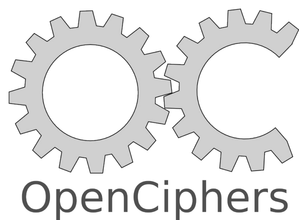
2007 © The OpenCiphers Project



VileFault

- Stores the home directory in a DMG file
- DMG is mounted when you login
- hdi framework handles everything
- Blocks get encrypted in 4kByte “chunks” AES-128, CBC mode
- Keys are encrypted (“wrapped”) in header of disk image
- Wrapping of keys done using 3DES-EDE
- Two different header formats (v1, v2)
- Version 2 header: support for asymmetrically (RSA) encrypted header

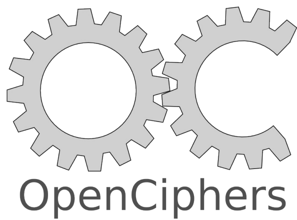
2007 © The OpenCiphers Project



VileFault

- Apple's FileVault
- Uses PBKDF2 for the password hashing
- Modified version of the WPA attack can be used to attack FileVault
- Just modified the WPA core to 1000 iterations instead of 4096
- Worked with Jacob Appelbaum & Ralf-Philip Weinmann to reverse engineer the FileVault format and encryption

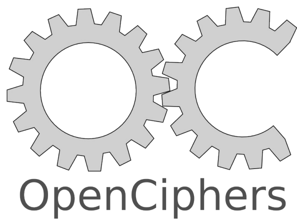
2007 © The OpenCiphers Project



VileFault

- Login password used to derive key for unwrapping
  - PBKDF2 (PKCS#5 v2.0), 1000 iterations
- Crypto parts implemented in CDSA/CSSM
  - DiskImages has own AES implementation, pulls in SHA-1 from OpenSSL dylib
- “Apple custom” key wrapping loosely according to RFC 2630 in Apple's CDSA provider (open source)

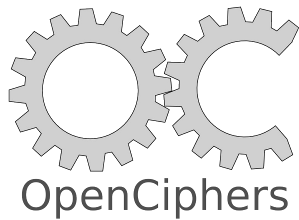
2007 © The OpenCiphers Project



VileFault

- vfdecrypt (Ralf Philip-Weinmann & Jacob Appelbaum)
  - Will use the same method with a correct password to decrypt the DMG file and output an unencrypted DMG file
  - Result can be mounted on any system without a password
- vfcrack (me!)
  - Unwrap the header
  - Use header to run PBKDF2 with possible passphrases
  - Use PBKDF2 hash to try and decrypt the AES key, if it doesn't work, try next passphrase
  - With the AES key decrypt the beginning of the DMG file and verify the first sector is correct (only needed with v2)

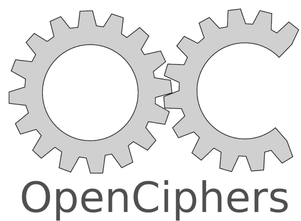
2007 © The OpenCiphers Project



VileFault

- Other attacks
  - Swap
    - The key can get paged to disk (whoops!)
    - Encrypted swap isn't enabled by default
  - Hibernation
    - You can extract the FileVault key from a hibernation file
    - Ring-0 code can find the key in memory
  - Weakest Link
    - The password used for the FileVault image is the same as your login password
    - Salted SHA-1 is much faster to crack than PBKDF2 (1 iteration vs 1000)
    - The RSA key is easier to crack than PBKDF2

2007 © The OpenCiphers Project



Performance Comparison

PC

vfcrack

800MHz P3	~100/sec
3.6GHz P4	~180/sec
AMD Opteron	~200/sec
2.16GHz IntelDuo	~200/sec

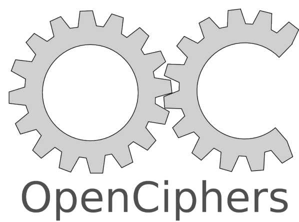
FPGA

vfcrack

LX25	~2,000/sec
LX50	~3,000/sec
15 Cluster	~30,000/sec

2007 © The OpenCiphers Project

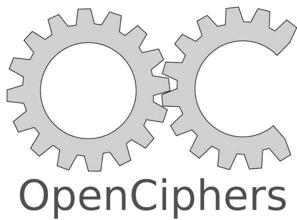
Demo



Bluetooth PIN Cracking

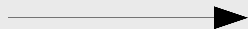
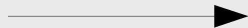
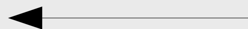
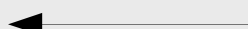
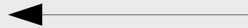
- Pairing bluetooth devices is similar to wifi authentication
- Why not crack the bluetooth PIN?
- Uses a modified version of SAFER+
- SAFER+ inherently runs much faster in hardware
- Attack originally explained and published by Yaniv Shaked and Avishai Wool
- Thierry Zoller demonstrated his own tool at hack.lu

2007 © The OpenCiphers Project

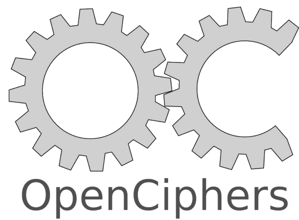


Bluetooth PIN Cracking

- How it works
  - Capture a bluetooth authentication
    (sorry, requires an expensive protocol analyzer)
  - This is what you'll see

Master		Slave	
in_rand			master sends a random nonce
m_comb_key			
		s_comb_key	sides create key based on the pin
m_au_rand			master sends random number
		s_res	slave hashes with E1 and replies
		s_au_rand	slave sends random number
m_sres			master hashes with E1 and replies

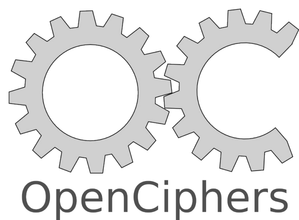
2007 © The OpenCiphers Project



Bluetooth PIN Cracking

- Just try a PIN and if the hashes match the capture, it is correct
- Extremely small keyspace since most devices just use numeric PINs (10^{16})
- My implementation is command line and should work on all systems with or without FPGA(s)

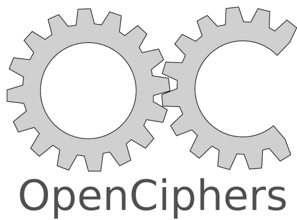
2007 © The OpenCiphers Project



Bluetooth PIN Cracking

- FPGA Implementation
  - Requires implementations of E21, E22, and E1 which all rely on SAFER+
  - Uses 16-stage pipeline version of SAFER+ which feeds back into itsself after each stage
  - To explain, here's some psuedocode

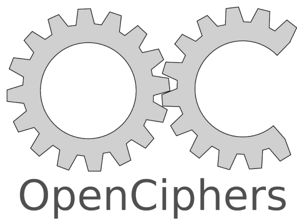
2007 © The OpenCiphers Project



Bluetooth PIN Cracking

```
for (pin = 0; ; pin++) {
    Kinit = E22(pin, s_bd_addr, in_rand);          //determine initialization key

    m_comb_key ^= Kinit;                           //decrypt comb_keys
    s_comb_key ^= Kinit;

    m_lk = E21(m_comb_key, m_bd_addr);             //determine link key
    s_lk = E21(s_comb_key, s_bd_addr);
    lk = m_lk ^ s_lk;

    m_sres_t = E1(lk, s_au_rand, m_bd_addr);       //verify authentication
    s_sres_t = E1(lk, m_au_rand, s_bd_addr);

    if (m_sres_t == m_sres && s_sres_t == s_sres)
        found!
}
```

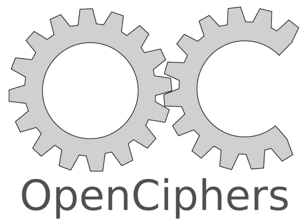

2007 © The OpenCiphers Project



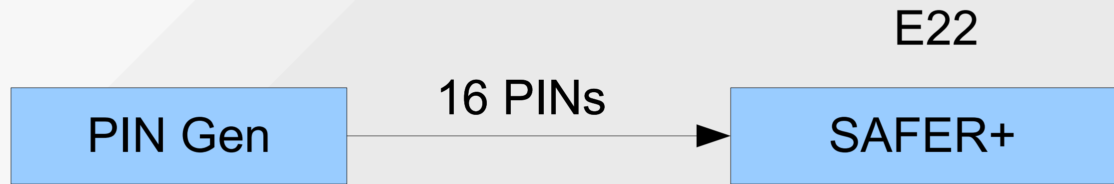
Bluetooth PIN Cracking

```
for (pin = 0; ; pin++) {
    Kinit = E22 (pin, s_bd_addr, in_rand);          //determine initialization key

    m_comb_key ^= Kinit;                            //decrypt comb_keys
    s_comb_key ^= Kinit;

    m_lk = E21 (m_comb_key, m_bd_addr);             //determine link key
    s_lk = E21 (s_comb_key, s_bd_addr);
    lk = m_lk ^ s_lk;

    m_sres_t = E1 (lk, s_au_rand, m_bd_addr);       //verify authentication
    s_sres_t = E1 (lk, m_au_rand, s_bd_addr);

    if (m_sres_t == m_sres && s_sres_t == s_sres)
        found !
}
```

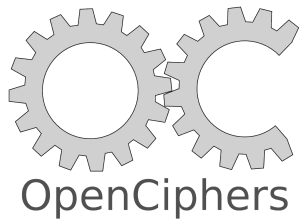

2007 © The OpenCiphers Project



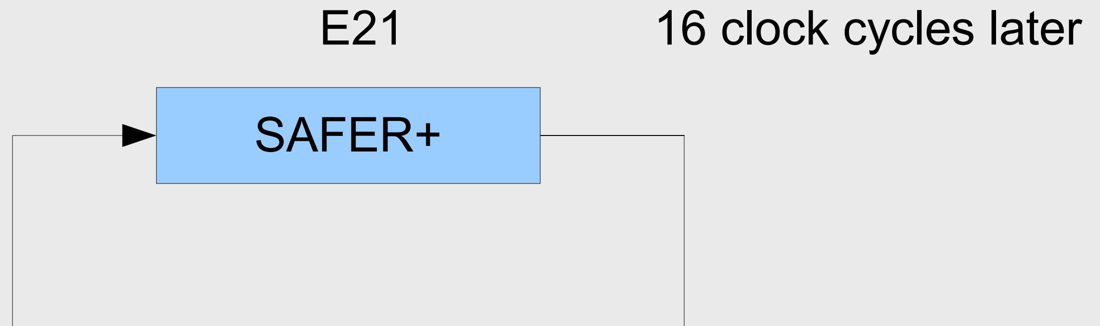
Bluetooth PIN Cracking



2007 © The OpenCiphers Project

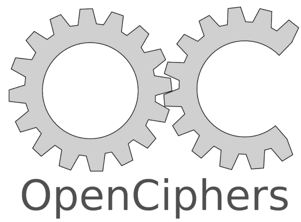


Bluetooth PIN Cracking

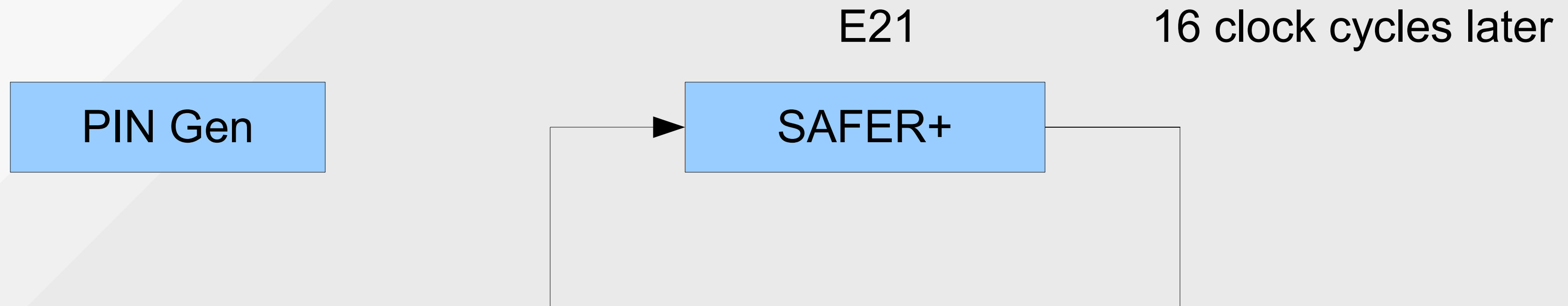


Output loops back and SAFER+ now does E21 for the Master

2007 © The OpenCiphers Project

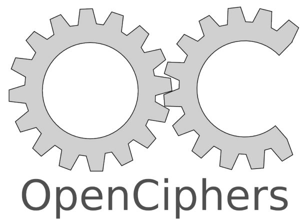


Bluetooth PIN Cracking



Then does the second E21 for the Slave
and combines the keys to create the link key

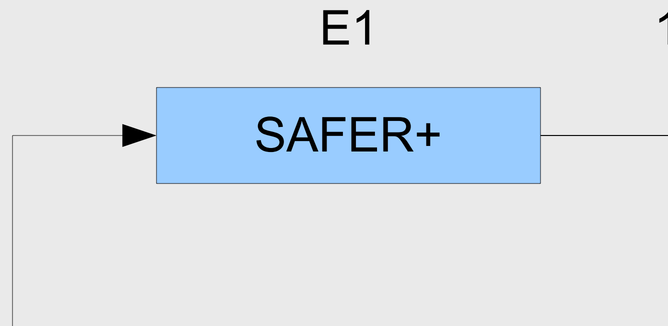
2007 © The OpenCiphers Project



Bluetooth PIN Cracking

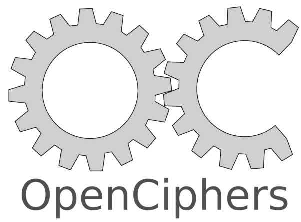
E1 16 clock cycles later

PIN Gen



Then the first part of E1 for the Slave

2007 © The OpenCiphers Project

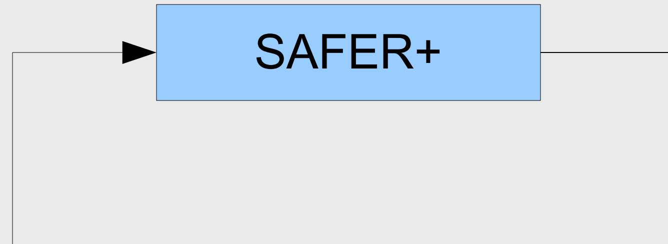


Bluetooth PIN Cracking

E1

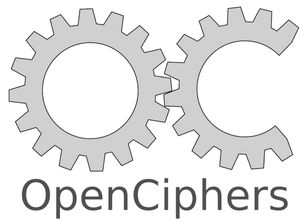
16 clock cycles later

PIN Gen



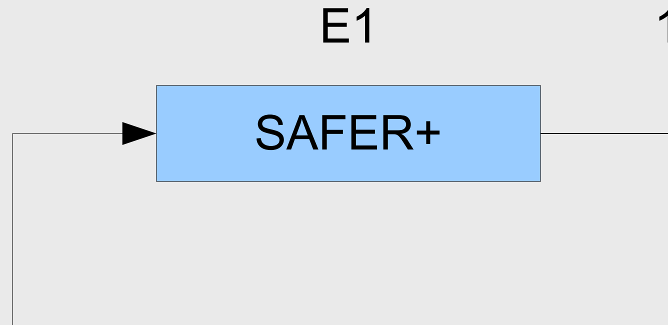
Then the second part of E1 for the Slave

2007 © The OpenCiphers Project



Bluetooth PIN Cracking

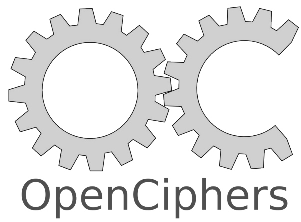
PIN Gen



16 clock cycles later

Then the first part of E1 for the Master

2007 © The OpenCiphers Project

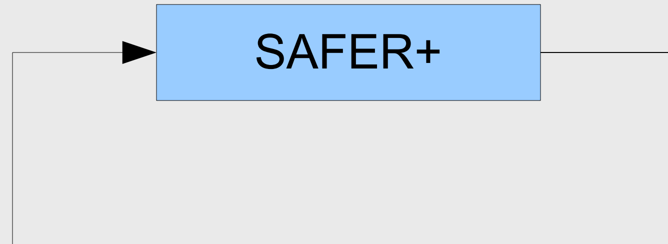


Bluetooth PIN Cracking

E1

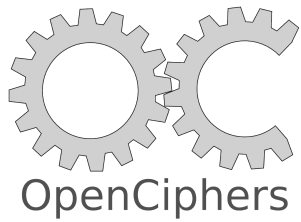
16 clock cycles later

PIN Gen

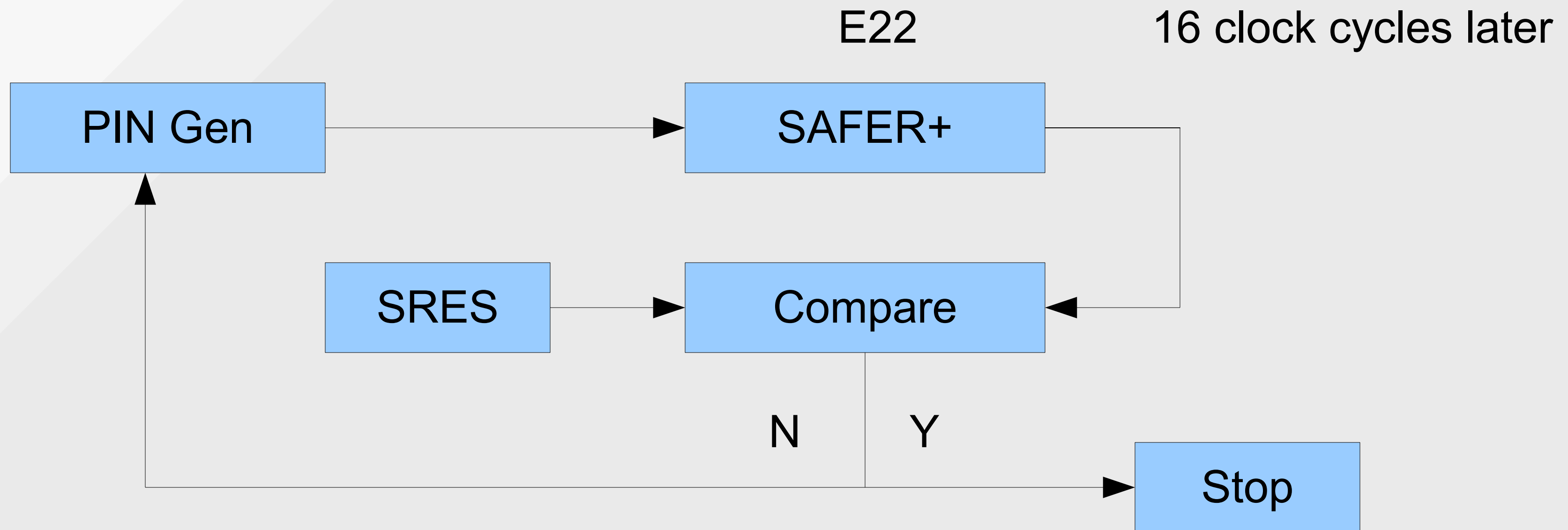


Then the second part of E1 for the Master

2007 © The OpenCiphers Project

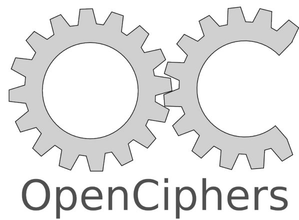


Bluetooth PIN Cracking



Then checks all of the sres values to see if any match while the process starts over

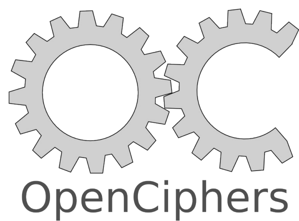
2007 © The OpenCiphers Project



Bluetooth PIN Cracking

- If the cracker stops the computer reads back the last generated PIN from the pin generator to determine what the valid PIN was
- The last generated PIN – 16 should be the cracked PIN

2007 © The OpenCiphers Project



Performance Comparison

PC

btpincrack

3.6GHz P4 ~40,000/sec

BTCrack

3.6GHz P4 ~100,000/sec

0.24 secs to crack 4 digit

42 min to crack 8 digit

FPGA

btpincrack

LX25 ~10,000,000/sec

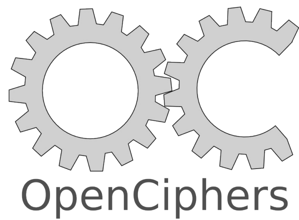
15 Cluster ~150,000,000/sec

LX50 ~15,000,000/sec

0.001 secs to crack 4 digit

10 secs to crack 8 digit

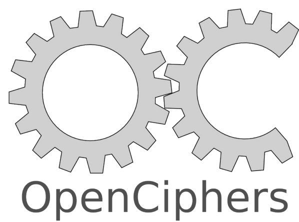
2007 © The OpenCiphers Project



Works in Progress

- Salted SHA-1 Hashes
  - Can do ~12M/sec
  - Requires more bandwidth to the card
  - Currently implementing it on the E-16
- Plugins to JtR
  - With the higher speed interface we can accelerate cracking other hashes as well
  - Can drop directly into JtR or just use JtR to create wordlists
  - Typically our only limitation would be the bandwidth

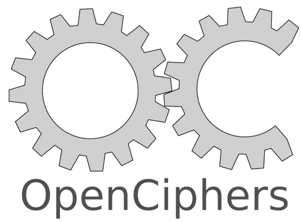
2007 © The OpenCiphers Project



Conclusion

- Get an FPGA and start cracking!
- Make use if your hardware to break crypto
- Add cool ascii matrix fx when you can :-)
- Choose bad passwords (please!)
- Don't feed the moose

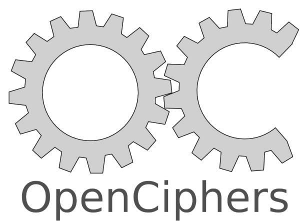
2007 © The OpenCiphers Project



Hardware Used

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

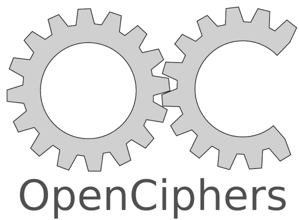
2007 © The OpenCiphers Project



Hardware Used

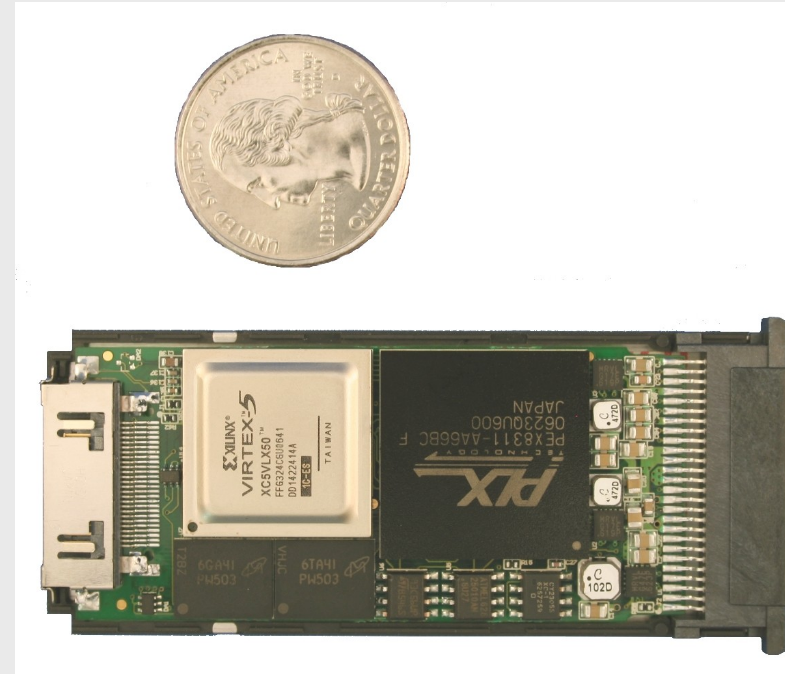
- Pico E-12 Super Cluster
  - 15 - E-12's
  - 2 - 2.8GHz Pentium 4's
  - 2 - 120GB HDD
  - 2 - DVD-RW
  - 550 Watt Power Supply

2007 © The OpenCiphers Project

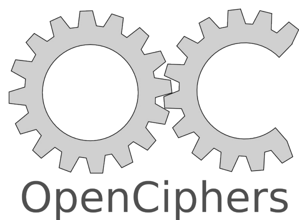


Future Hardware

- Pico E-16
  - ExpressCard 34
    - Works in MacBook Pros
    - 2.5Gbps full-duplex
  - Virtex-5 LX50 (~2x faster)
  - 32MB SRAM
  - External ExpressCard Chip
  - Made for Crypto Cracking
  - More affordable
  - Availability -Soon



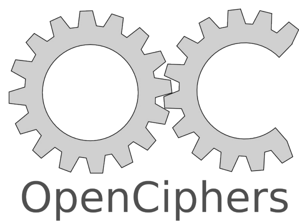
2007 © The OpenCiphers Project



Thanks

- Johny Cache (airbase/jc-wepcrack/jc-aircrack)
- Josh Wright (cowpatty)
- RenderMan (pmk hashtable monkey)
- Beetle (ghettopmk!)
- Jacob Appelbaum & Ralf-Philip Weinmann
- Audience (feel free to throw moosepattys now!!)

2007 © The OpenCiphers Project



Questions?

- David Hulton
  - david@toorcon.org
  - http://www.openciphers.org
  - http://www.picocomputing.com
  - http://www.toorcon.org
  - http://www.802.11mercenary.net
  - http://www.churchofwifi.org

2007 © The OpenCiphers Project