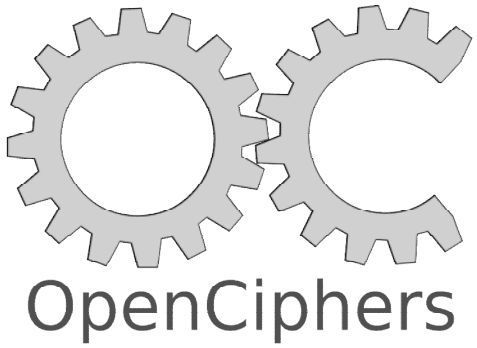
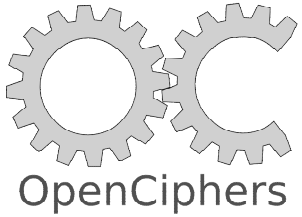


Cracking WiFi... Faster!

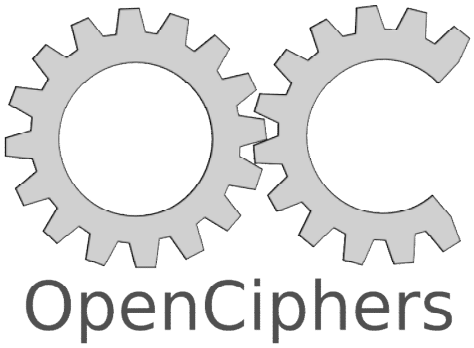
David Hulton <dhulton@openciphers.org>
Johnny Cache <johnycsh@gmail.com>
Beetle <beetle@shmoo.com>



Cracking WiFi… Faster!

- coWPAtty
  - Church of WiFi Introduction
  - WPA Overview
  - Precomputing tables
  - Performance
- Airbase
  - jc-aircrack
  - jc-wepcrack
  - pico-wepcrack
  - Performance
- Conclusion

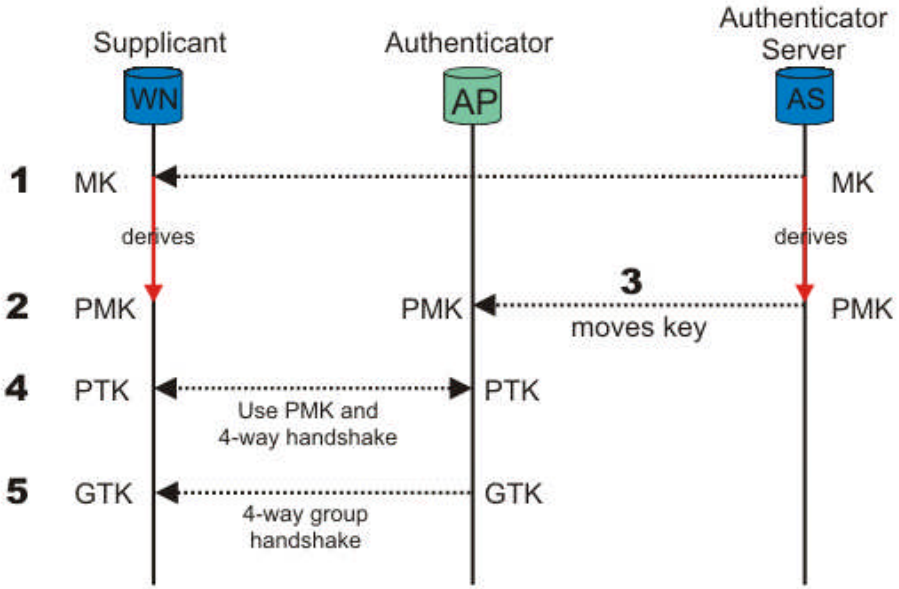
2006 © The OpenCiphers Project

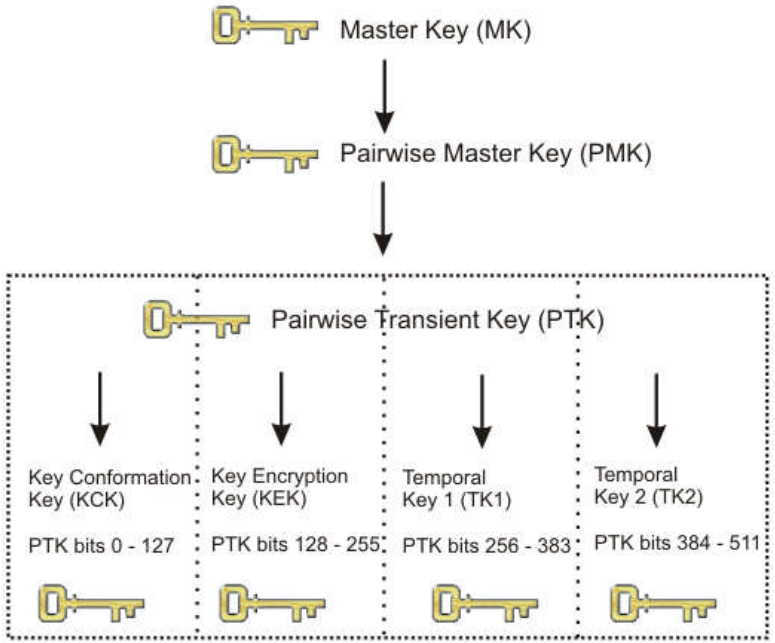


Church of Wifi Presentation

Introduction to WPA

- WiFi Protected Access

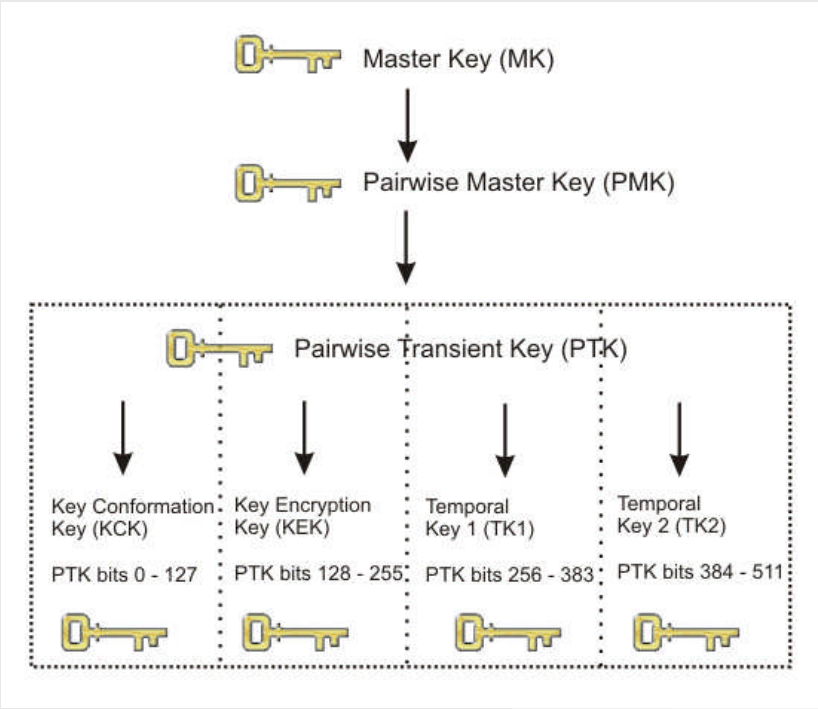




2006 © The OpenCiphers Project

Introduction to WPA

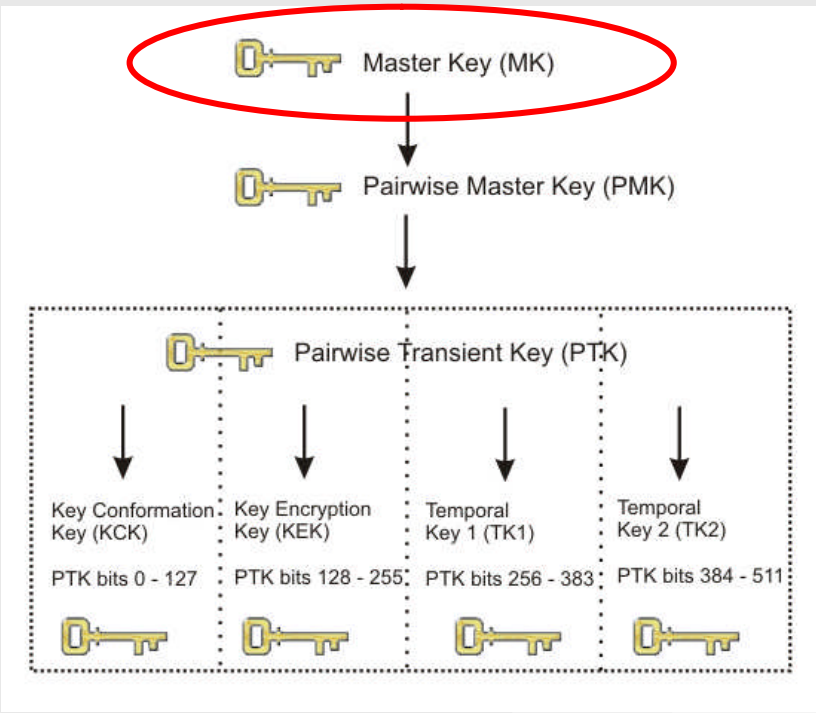
- PSK
  - MK is your passphrase
  - It's run through PBKDF2 to generate the PMK



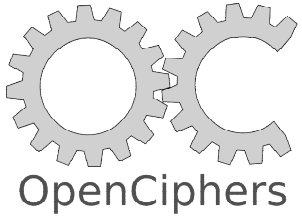
2006 © The OpenCiphers Project

Introduction to WPA

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

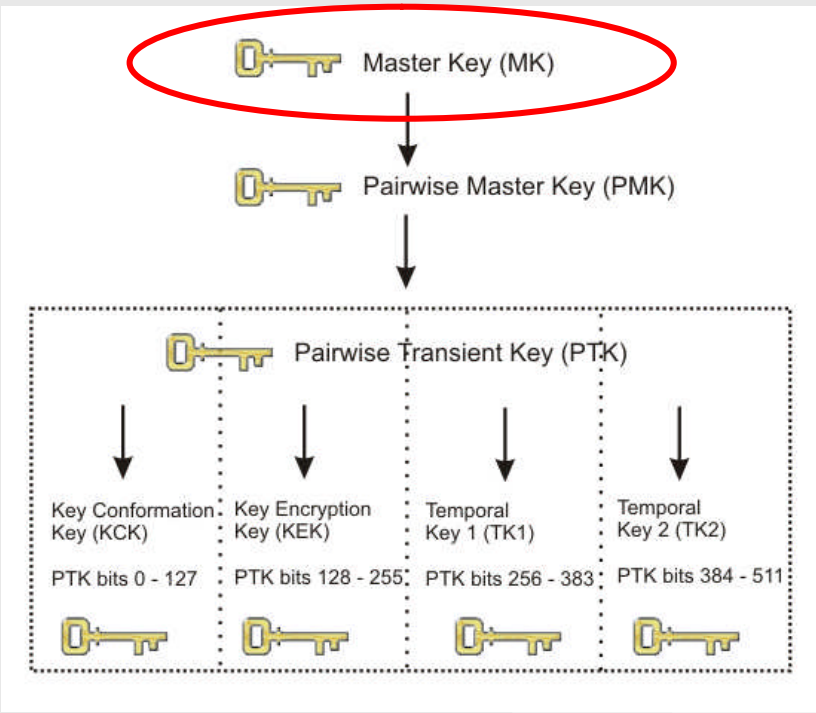


2006 © The OpenCiphers Project

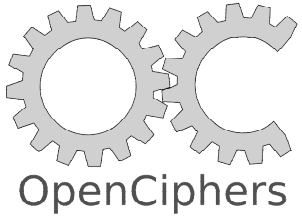


Introduction to WPA

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



2006 © The OpenCiphers Project

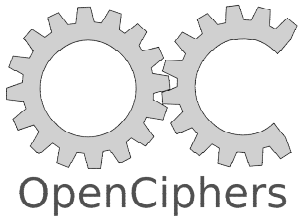


Introduction to WPA

- PBKDF2

```
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);
```


2006 © The OpenCiphers Project



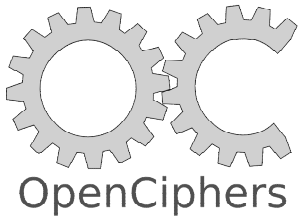
Introduction to WPA

- sha1_hmac

```
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);
```


2006 © The OpenCiphers Project



Introduction to WPA

- sha1_hmac

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

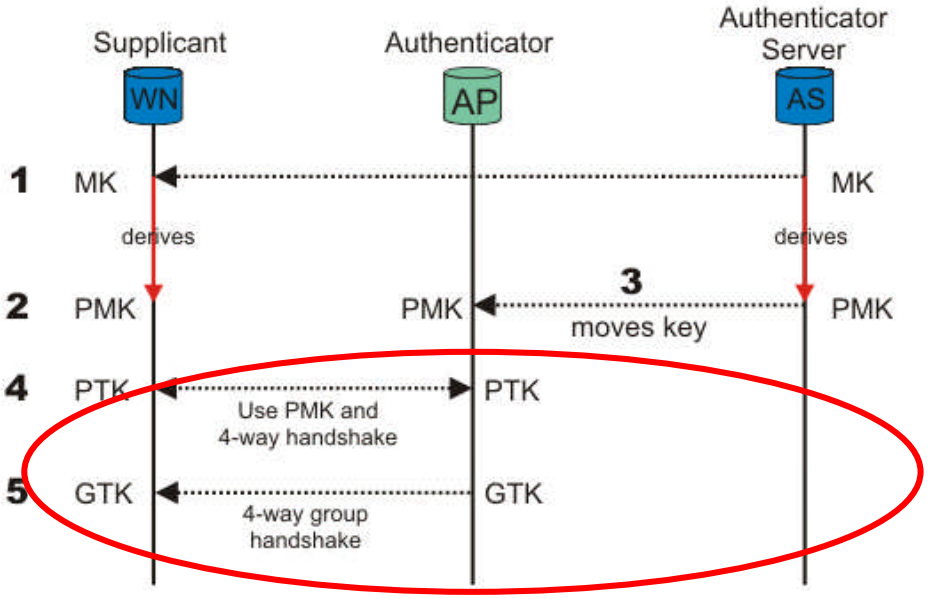
```
sha1init(ctx);
ctx = sha1update(ctx, MK ^ 0x36);
ctx = sha1update(ctx, t);
innersha1_ctx = sha1final(ctx);

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

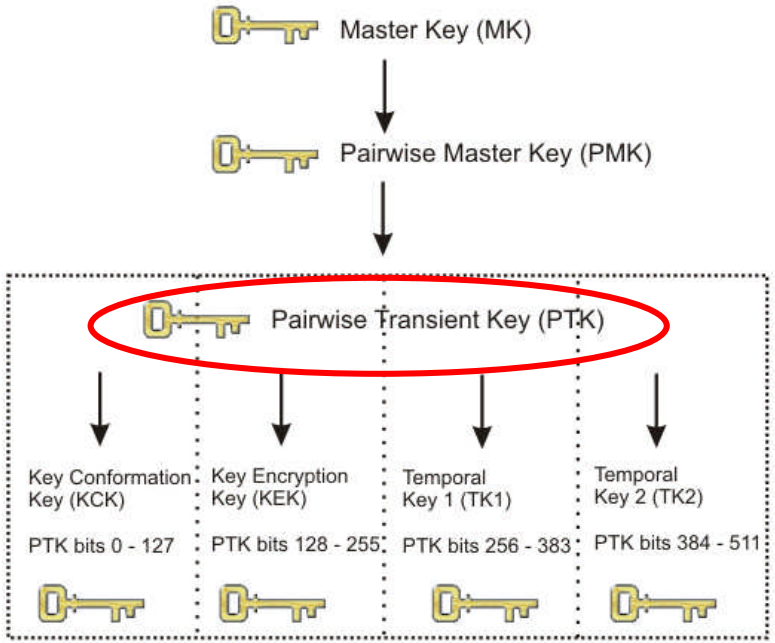
You can cache some of the state to reduce the number of required SHA1's

2006 © The OpenCiphers Project

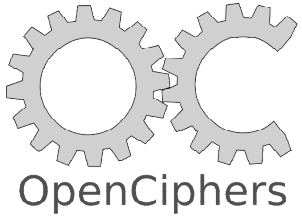
Introduction to WPA

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





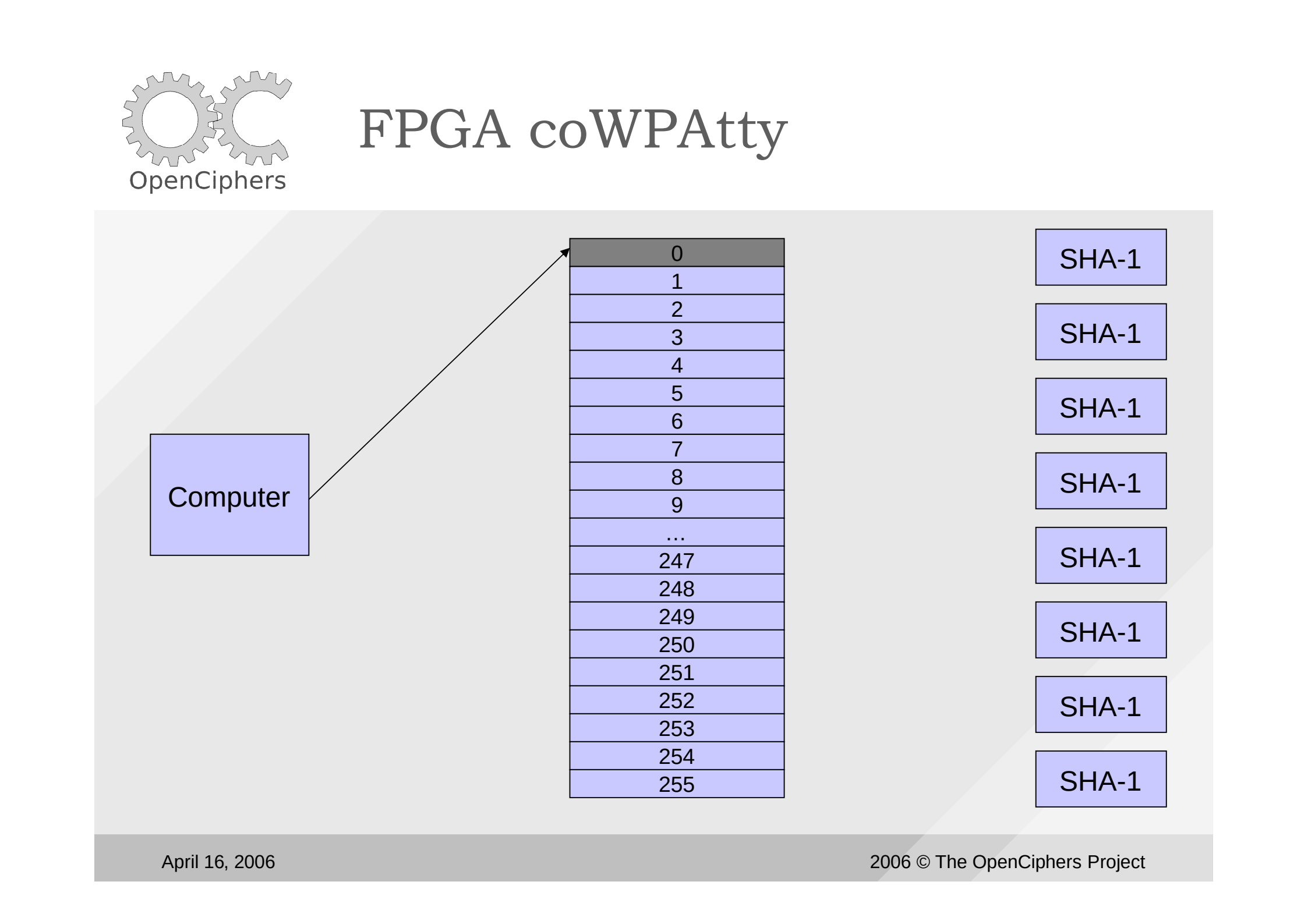
2006 © The OpenCiphers Project

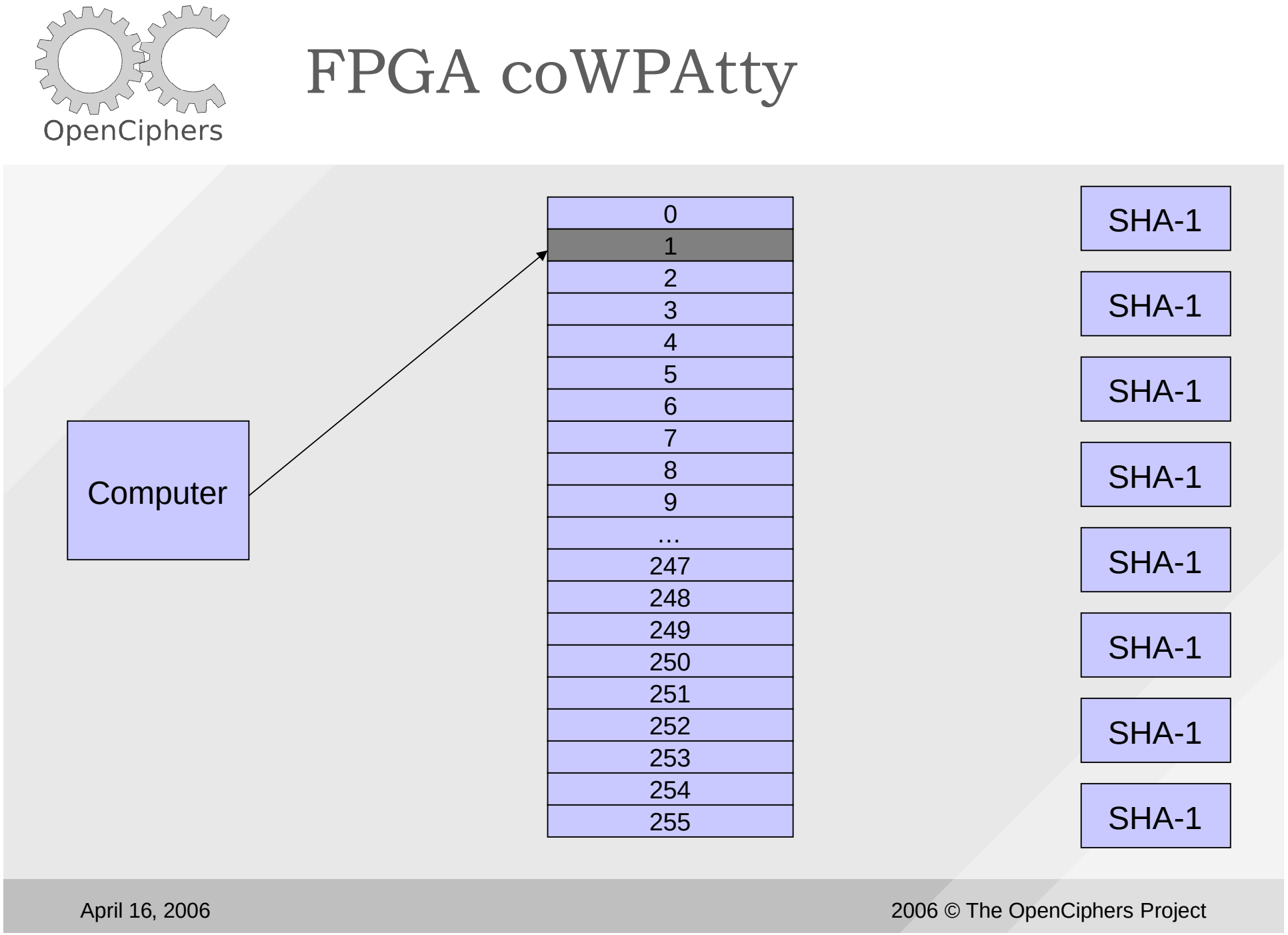


FPGA coWPAtty

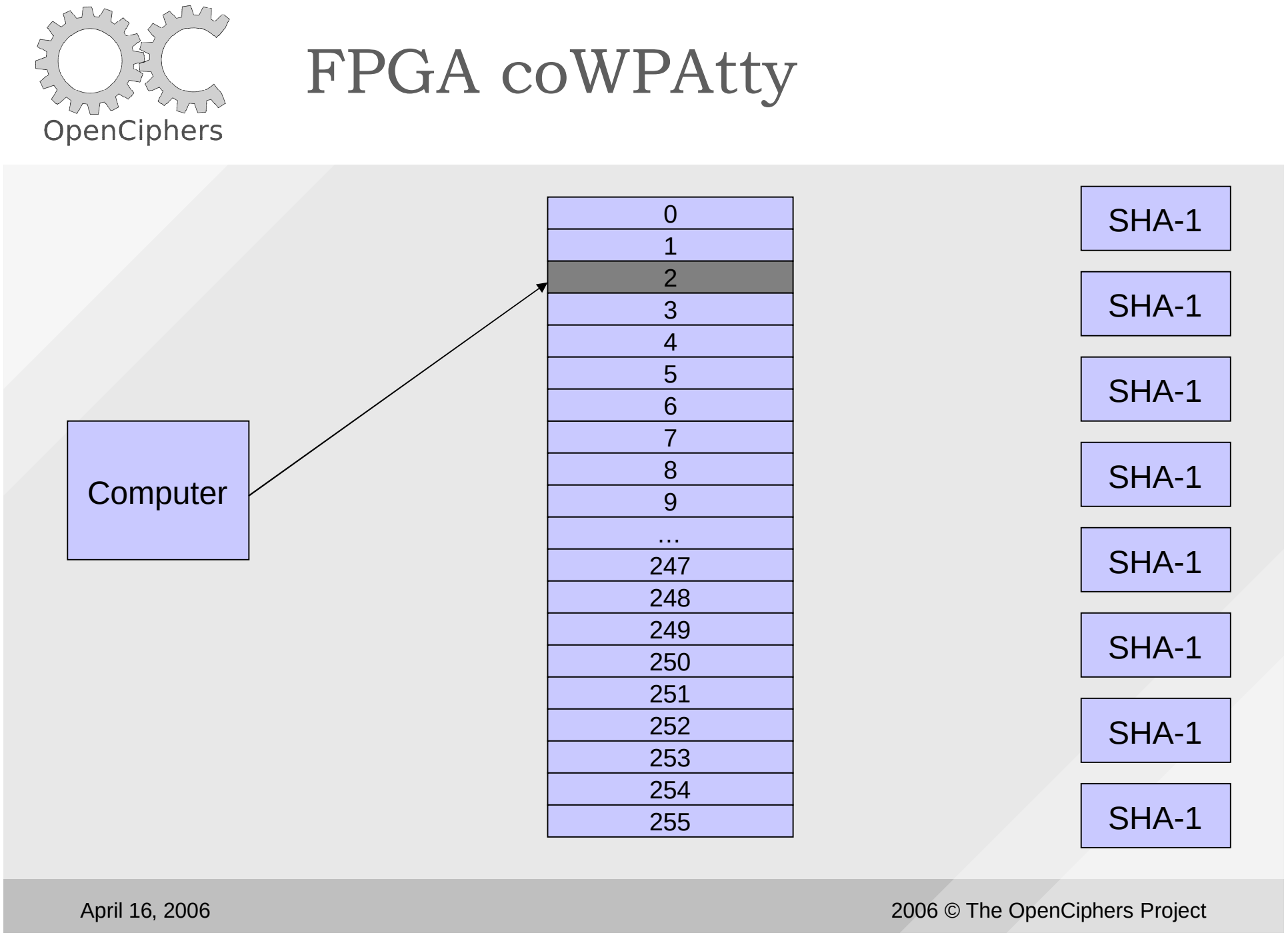
- Uses 8 SHA-1 Cores
- Uses BlockRAM to buffer the words fed to the cores
- As long as the machine is able to supply words fast enough, the SHA-1 cores will be utilized fully

2006 © The OpenCiphers Project

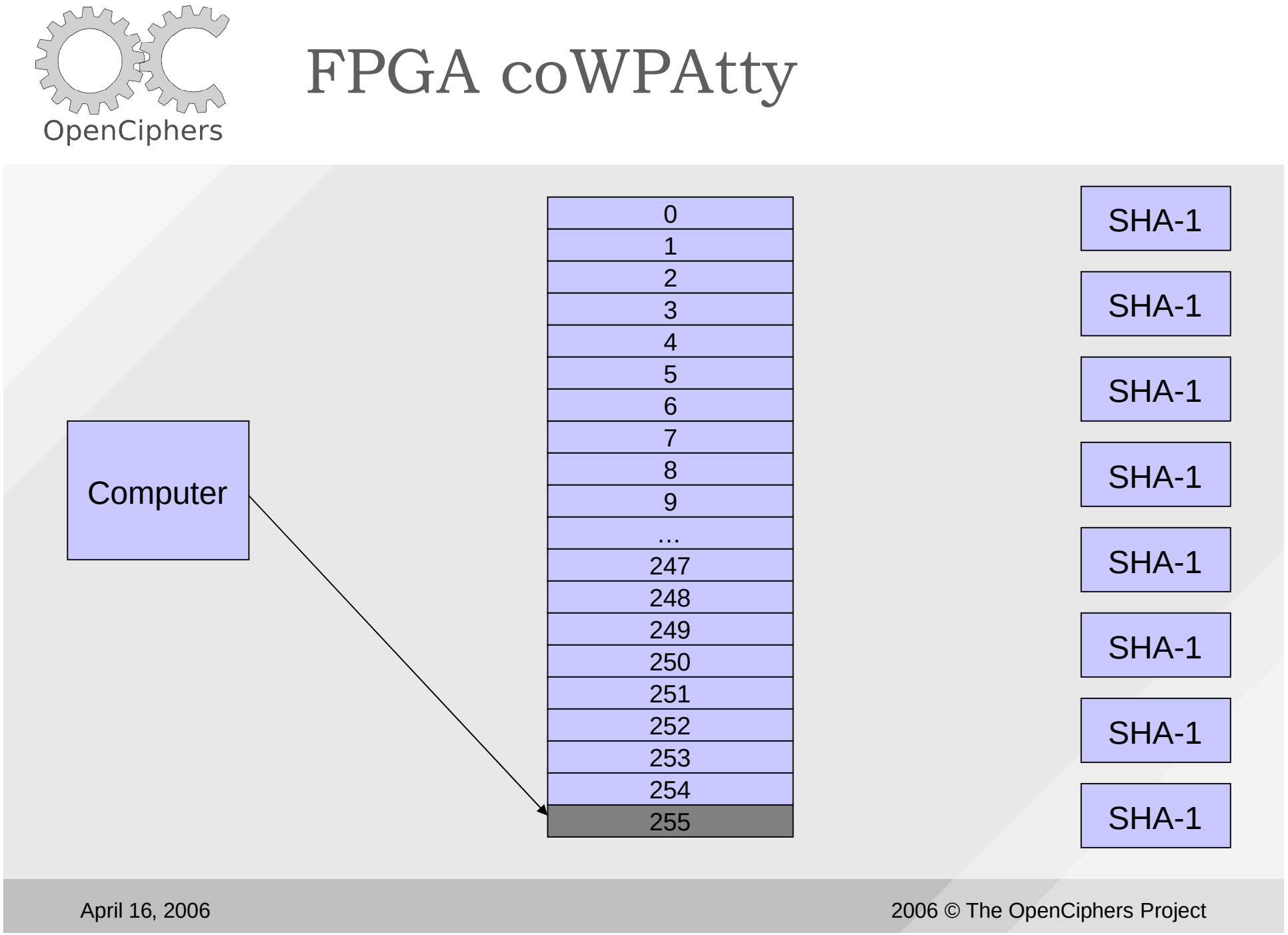

OpenCiphers
FPGA coWPAtty
Computer
0
1
2
3
4
5
6
7
8
9
...
247
248
249
250
251
252
253
254
255
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
April 16, 2006
2006 © The OpenCiphers Project



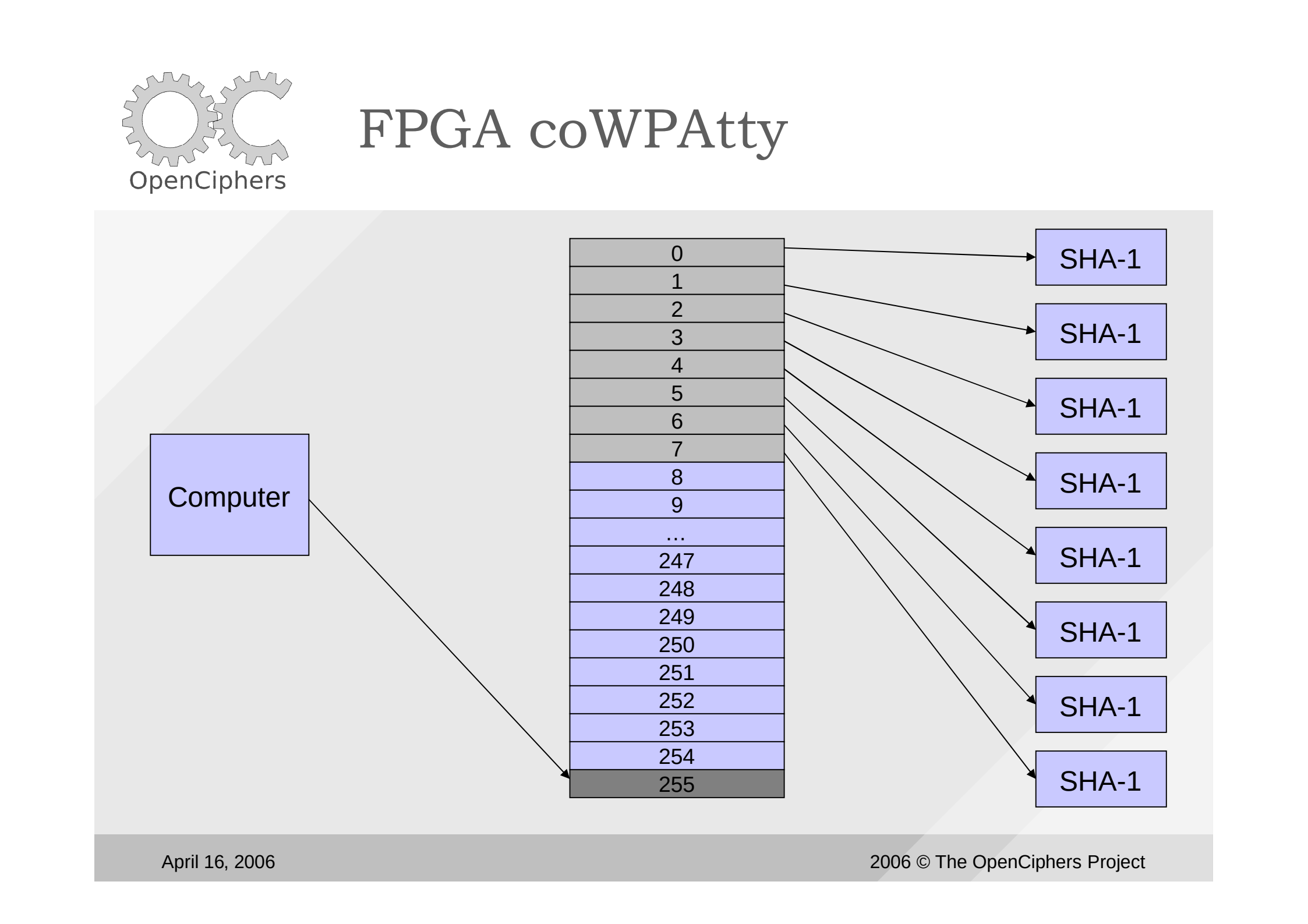
2006 © The OpenCiphers Project

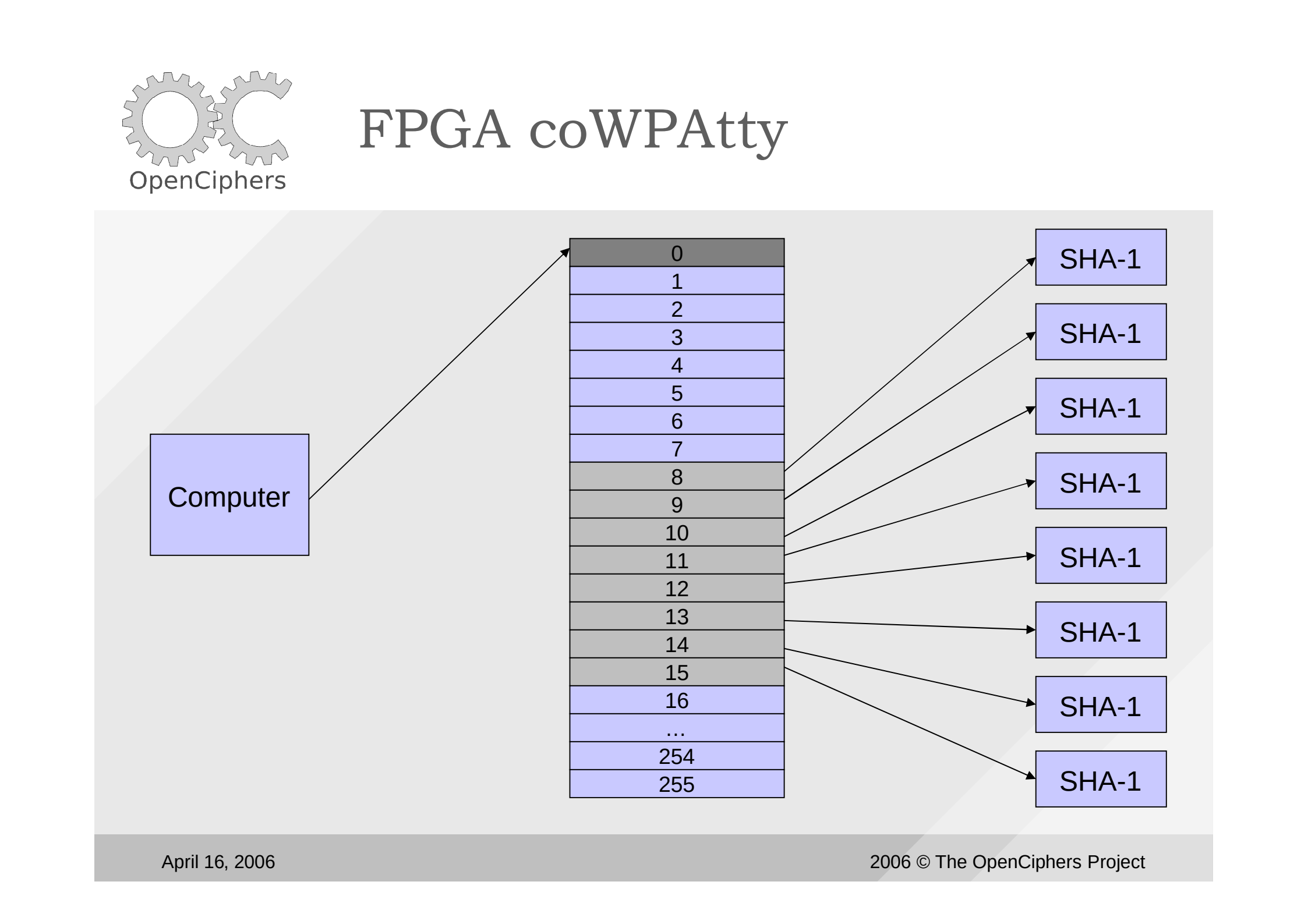


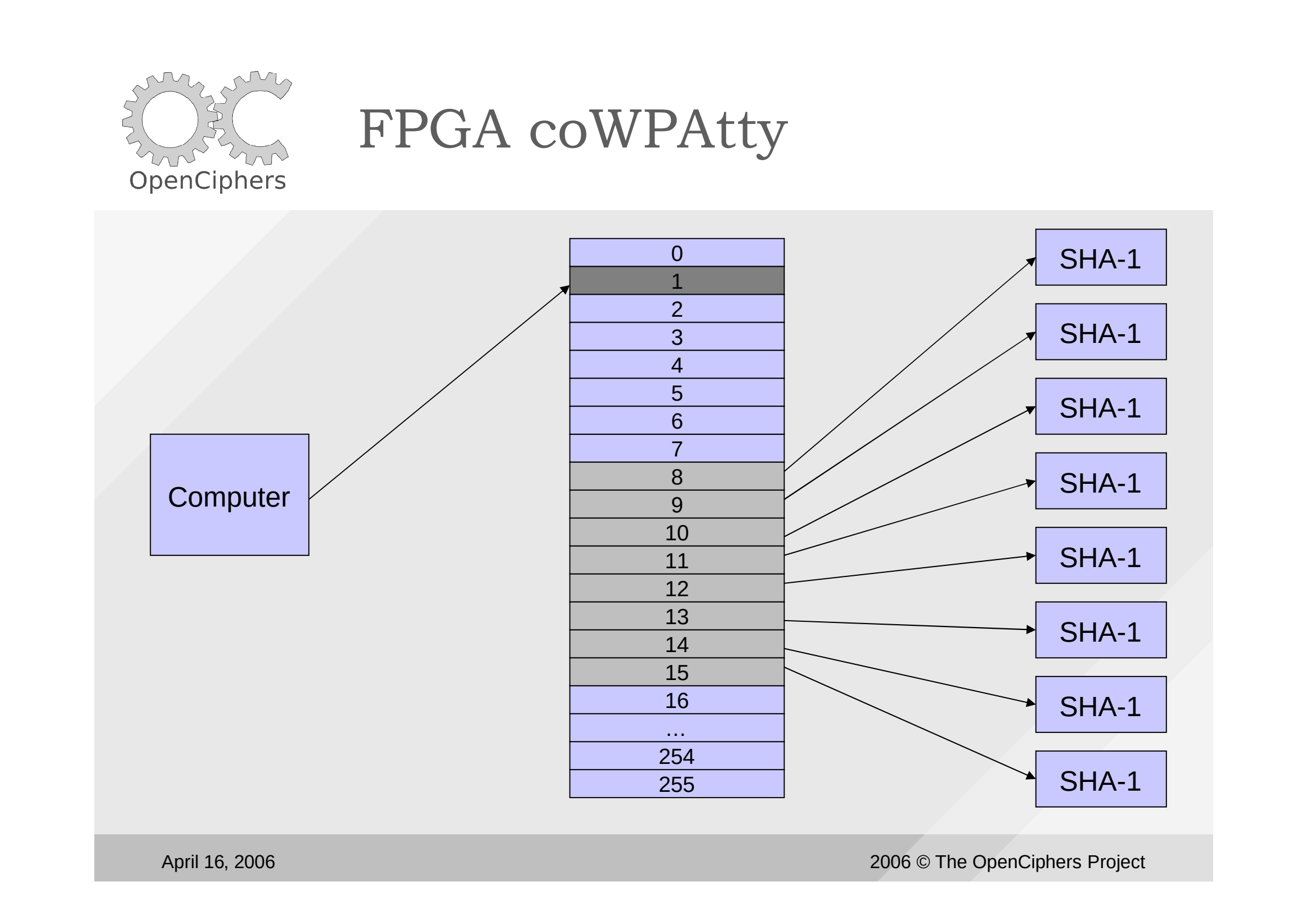
2006 © The OpenCiphers Project

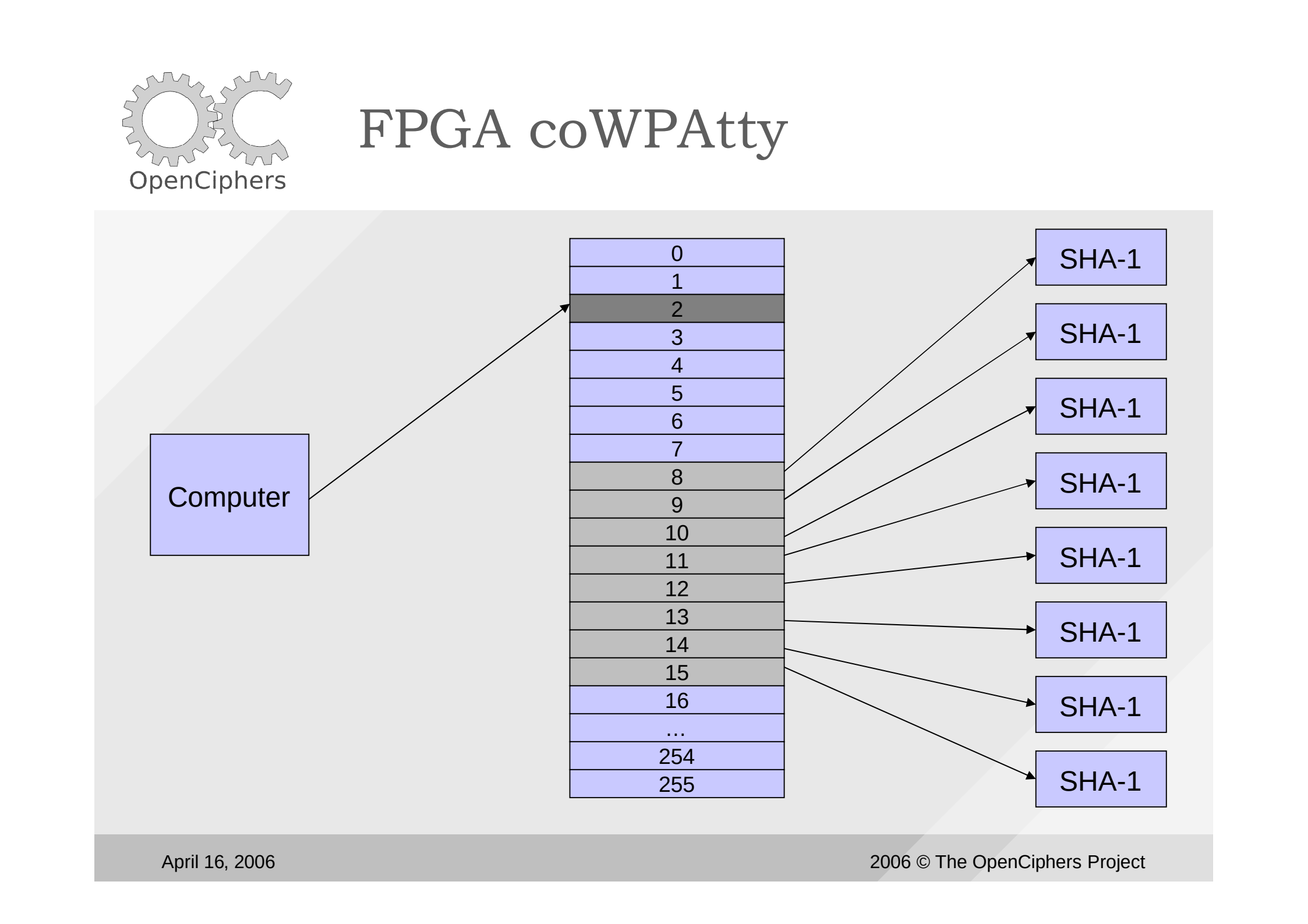


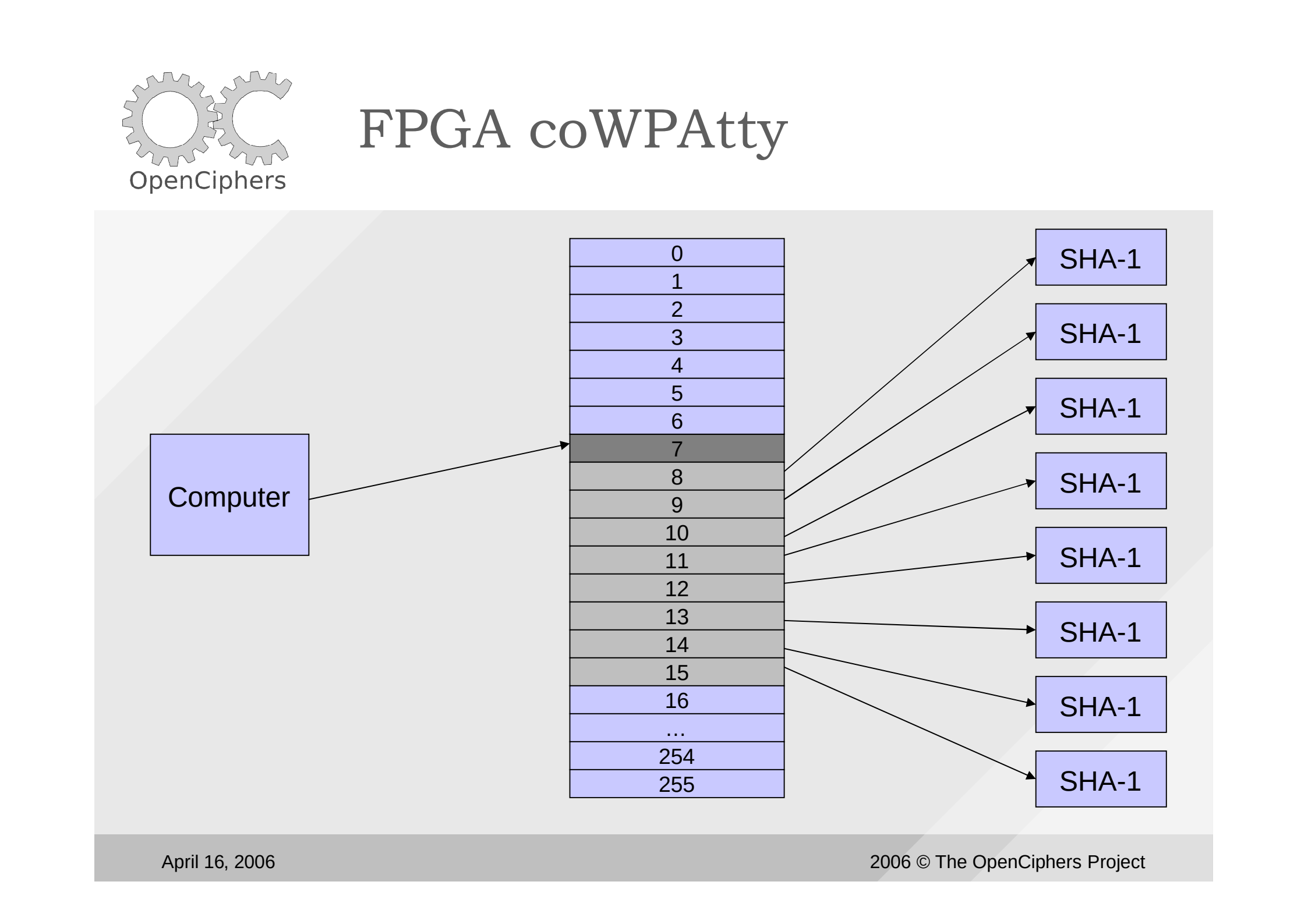
2006 © The OpenCiphers Project

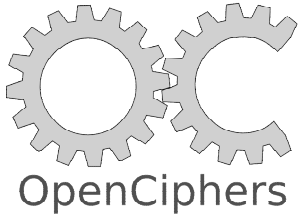

OpenCiphers
FPGA coWPAtty
Computer
0
1
2
3
4
5
6
7
8
9
...
247
248
249
250
251
252
253
254
255
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
SHA-1
April 16, 2006
2006 © The OpenCiphers Project


OpenCiphers
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
April 16, 2006
2006 © The OpenCiphers Project


OpenCiphers
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
April 16, 2006
2006 © The OpenCiphers Project


OpenCiphers
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
April 16, 2006
2006 © The OpenCiphers Project


OpenCiphers
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
April 16, 2006
2006 © The OpenCiphers Project



Performance Comparison

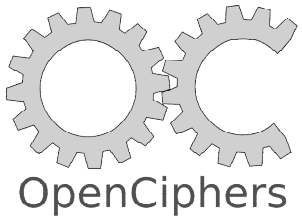
PC

- 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
  - FX60 ~1,000/sec

2006 © The OpenCiphers Project



Results

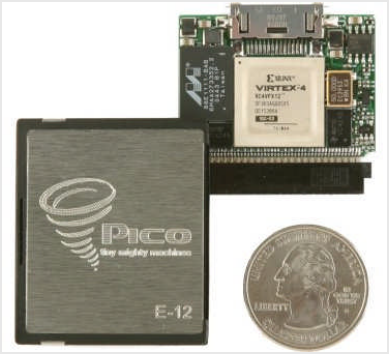
- Decided to compute hash tables for a 1,000,000 passphrase wordlist for the top 1,000 SSIDs
- Took RenderMan 1 month to compute on his cluster
- Found out that his wordlist had return characters at the end of every line
- (after computing for a month)
- He sent me an email asking for help
- A 15 card cluster did it in 2 days ;-)

2006 © The OpenCiphers Project

OpenCiphers

FPGA coWPAtty

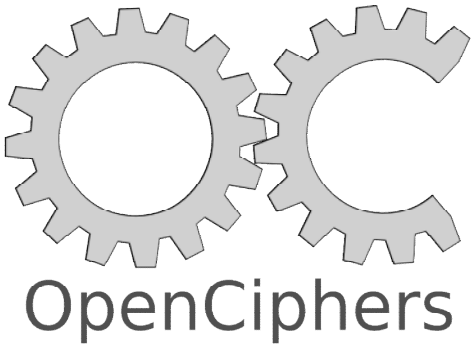
+



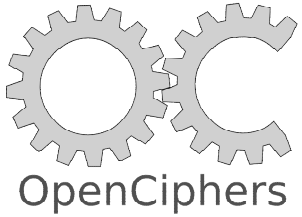
+

= ?

2006 © The OpenCiphers Project



Demo

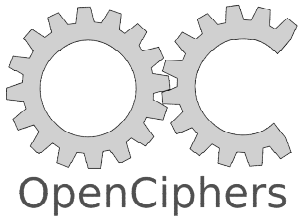


Mac OS-X coWPAtty???

- /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
```

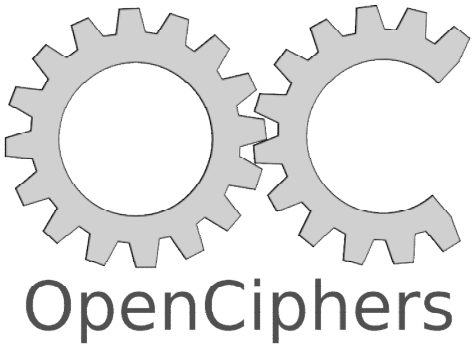

2006 © The OpenCiphers Project



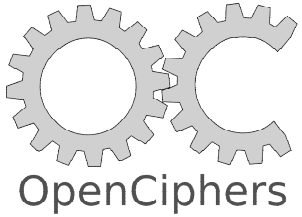
ghetto_pmk.pl

```
#!/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
```


2006 © The OpenCiphers Project



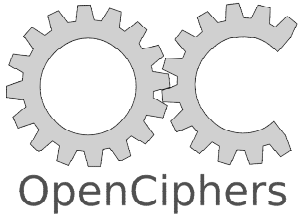
Airbase



pico-wepcrack

- FPGA Core
  - Uses 16 custom RC4 cores
  - Uses BlockRAM for S-Boxes
  - Will try every key between a start and end

2006 © The OpenCiphers Project

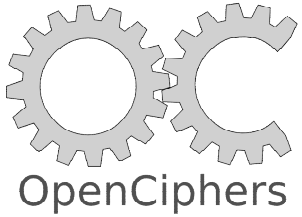


pico-wepcrack

- 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]];
}
```


2006 © The OpenCiphers Project

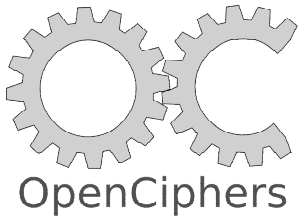


pico-wepcrack

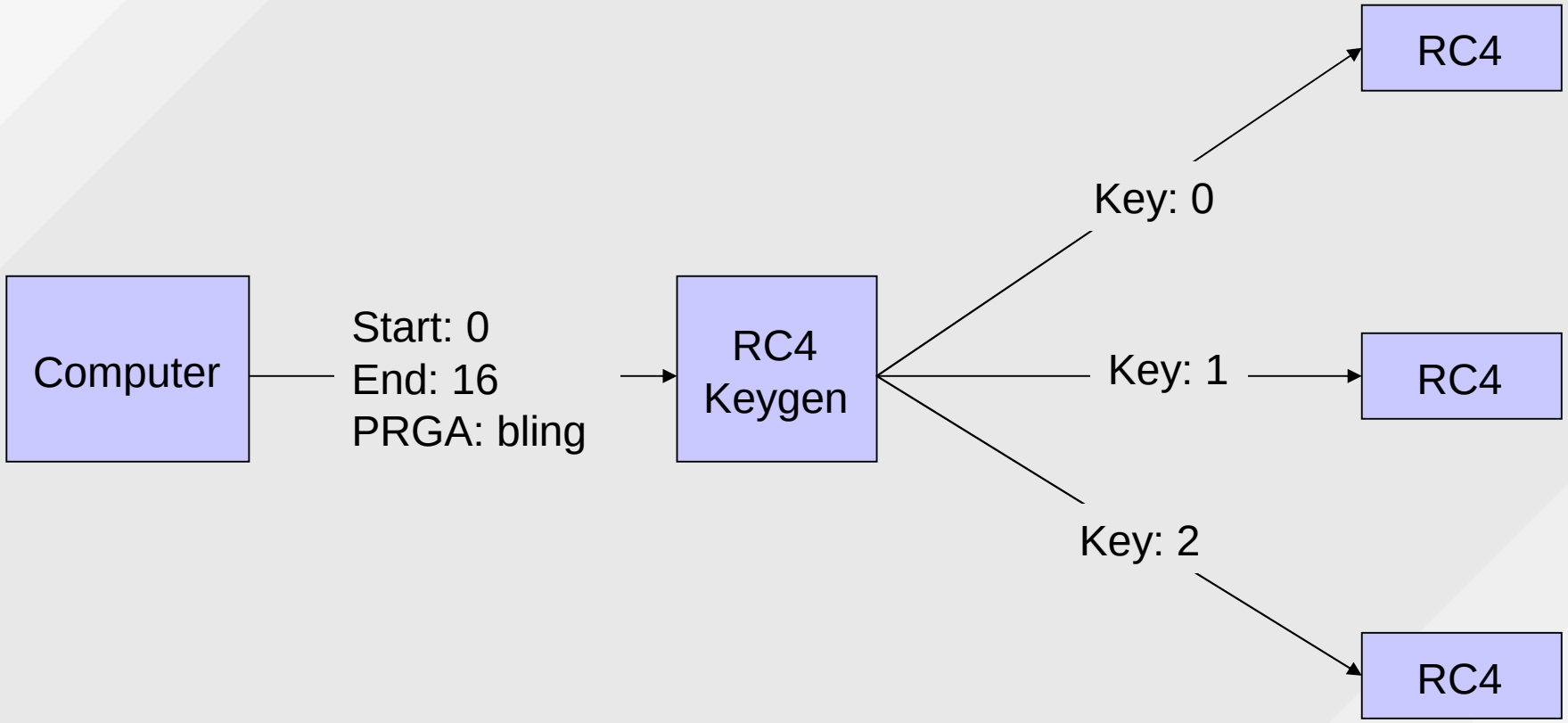
- RC4:

```
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
}
```

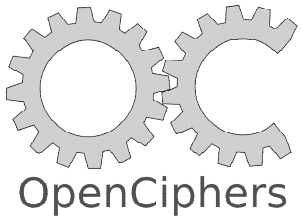

2006 © The OpenCiphers Project



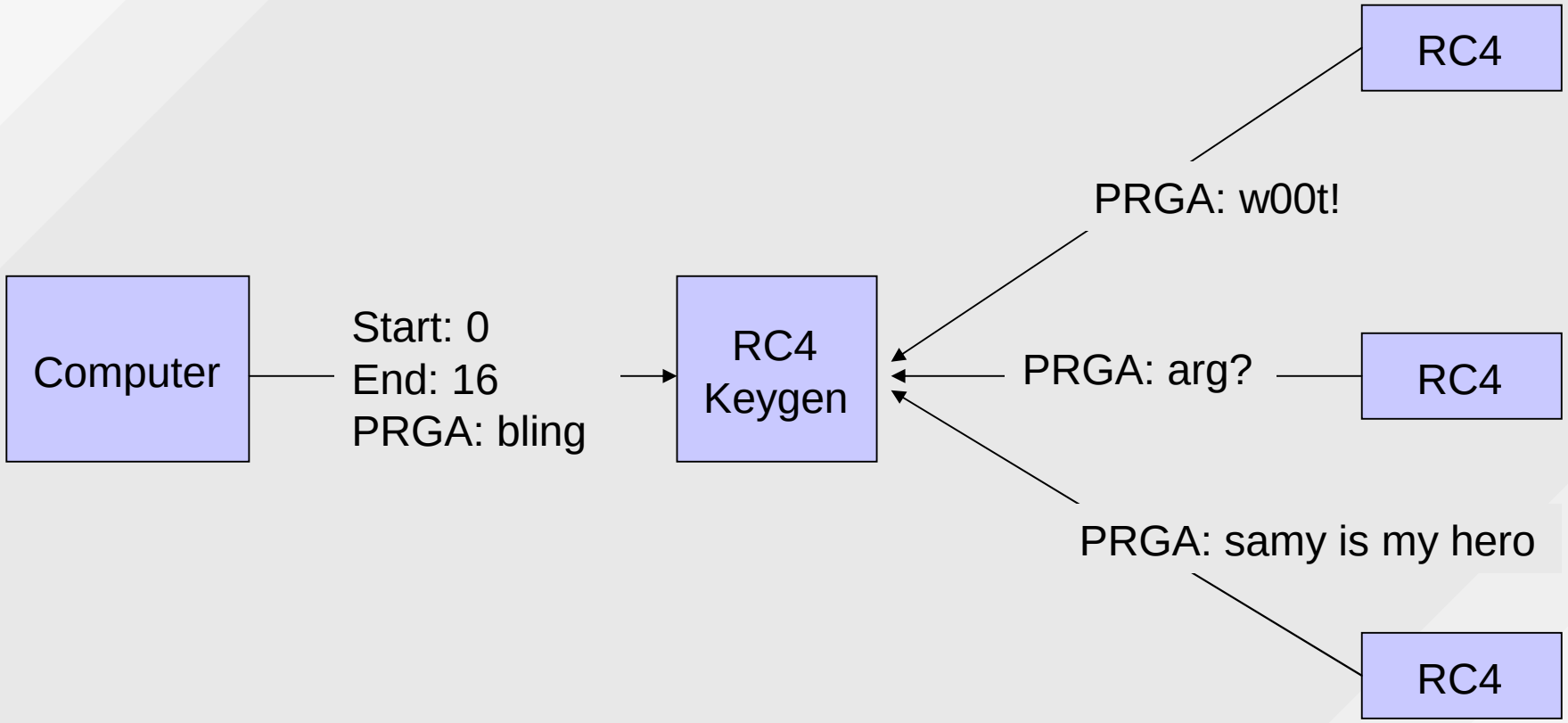
pico-wepcrack



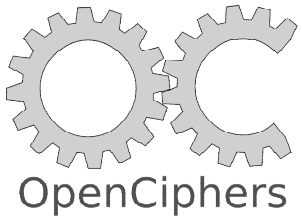
2006 © The OpenCiphers Project



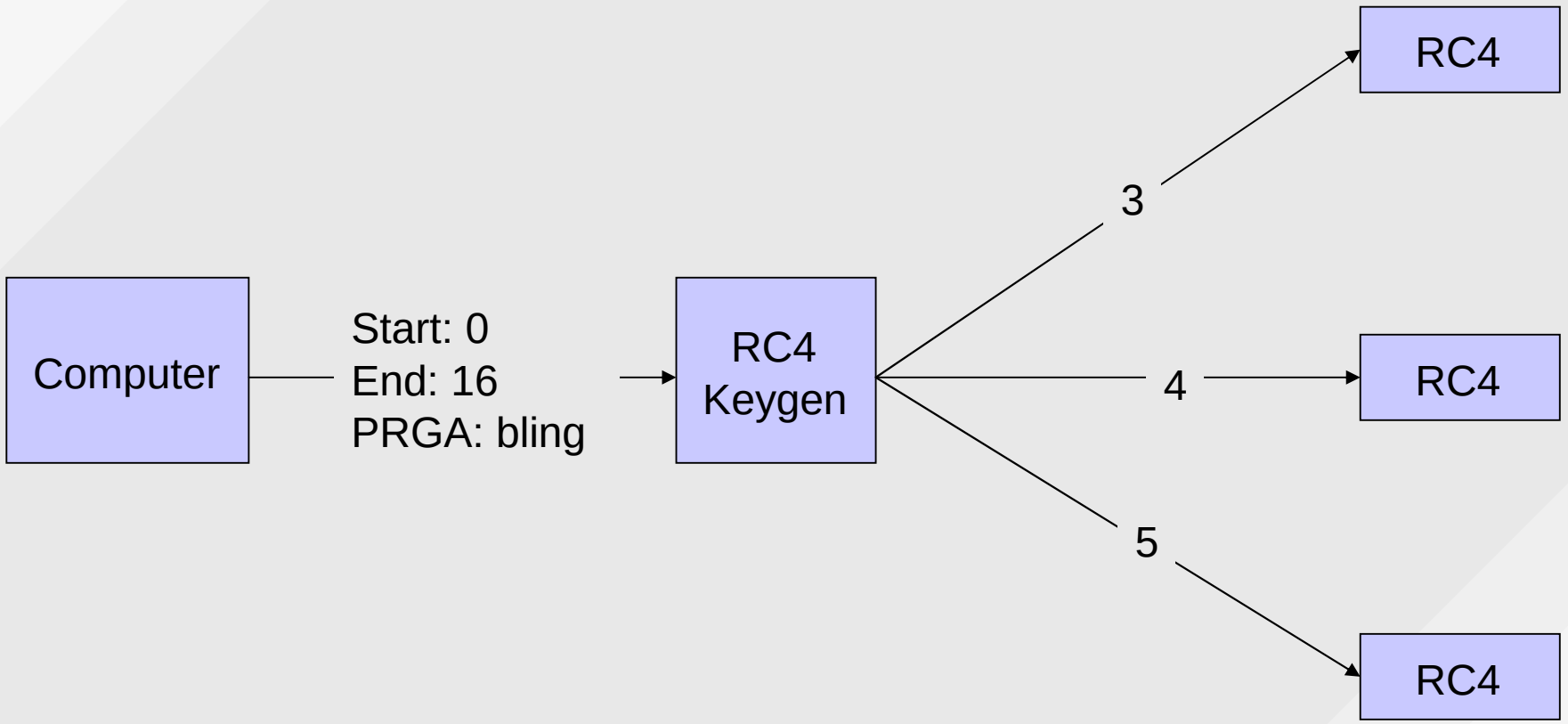
pico-wepcrack



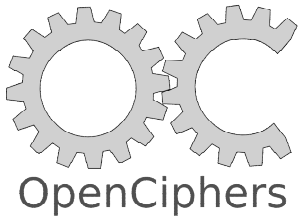
2006 © The OpenCiphers Project



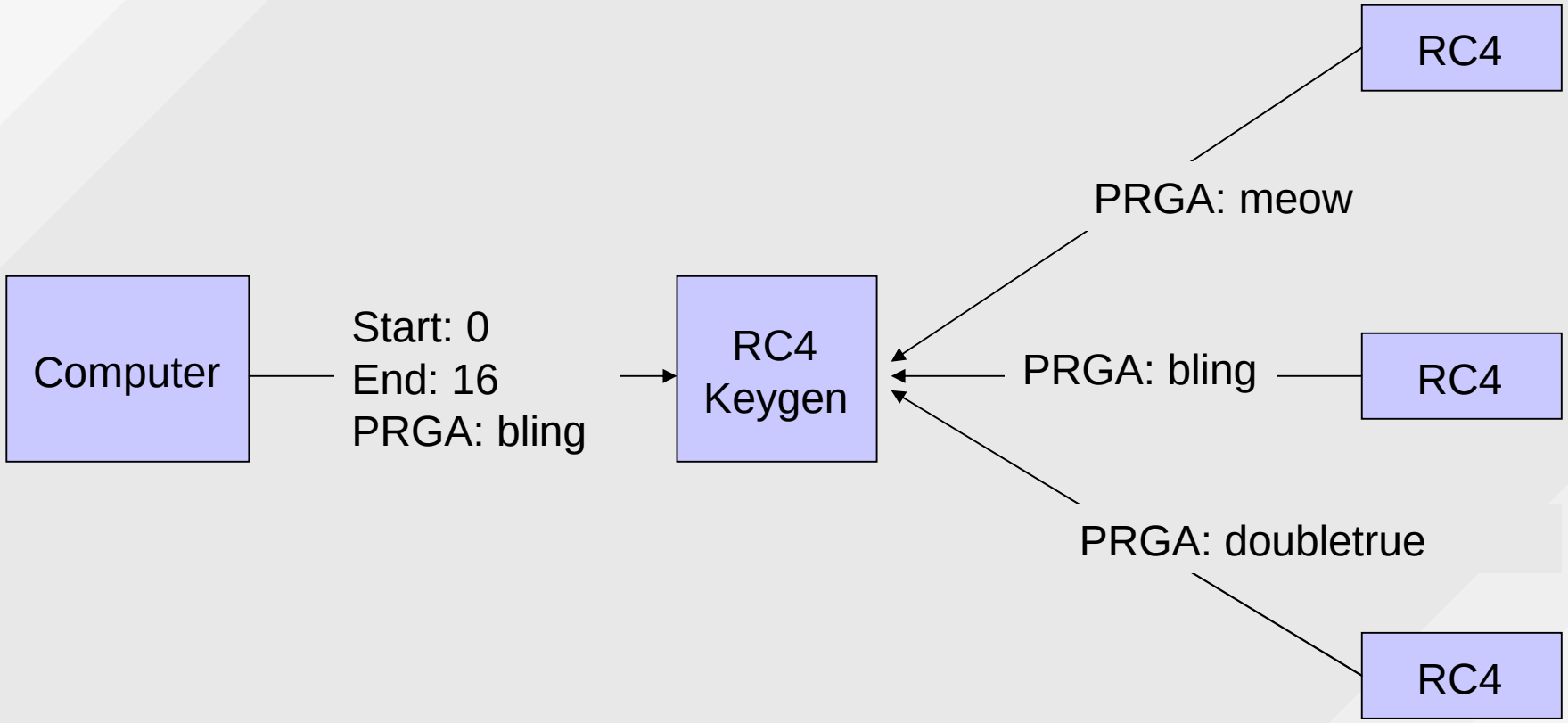
pico-wepcrack



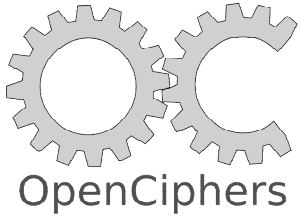
2006 © The OpenCiphers Project



pico-wepcrack



2006 © The OpenCiphers Project

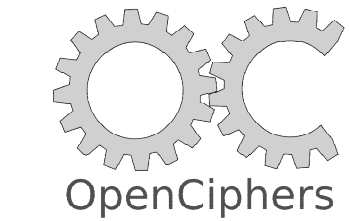


pico-wepcrack

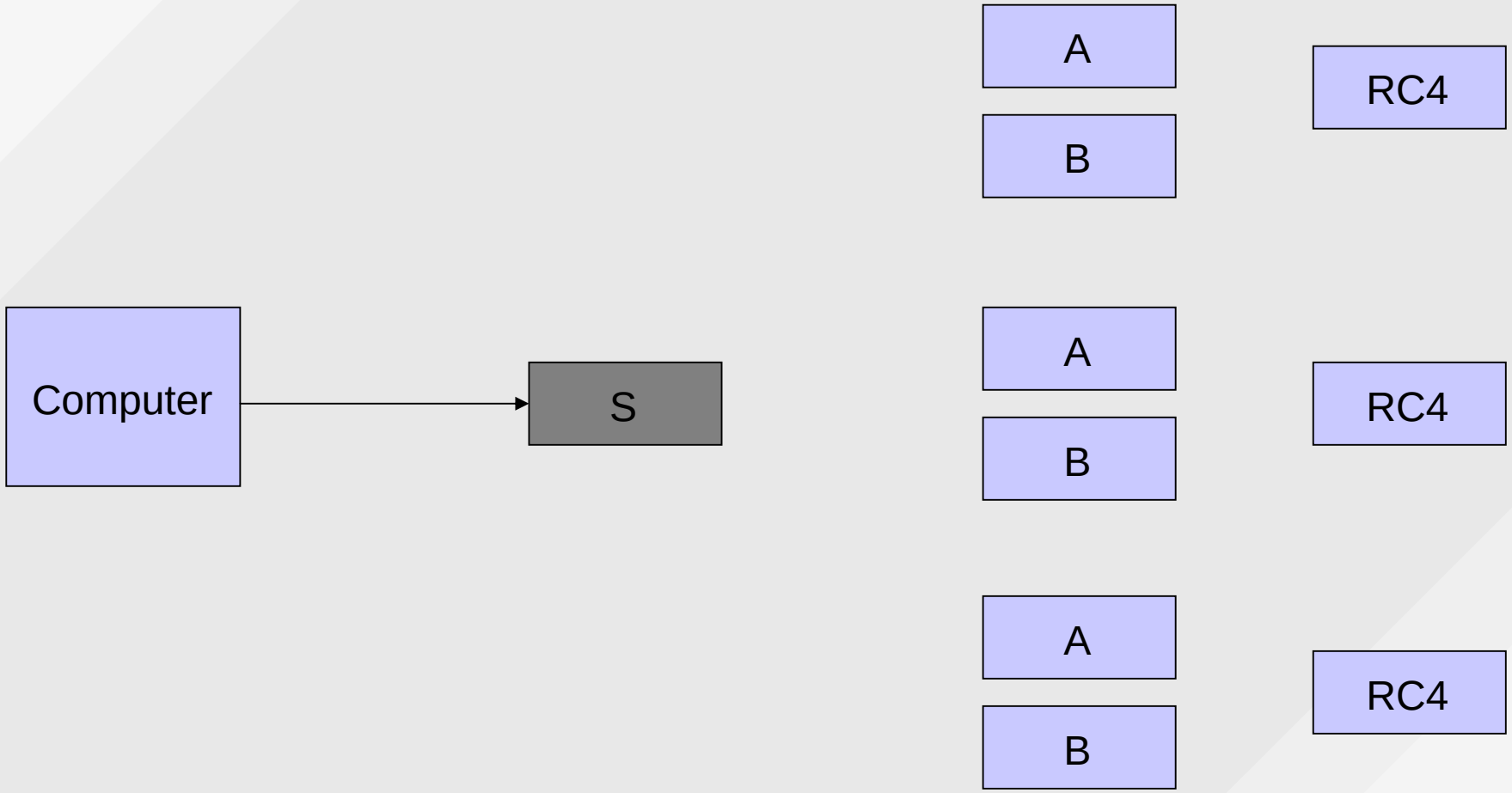
- RC4:

```
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]];
}
```

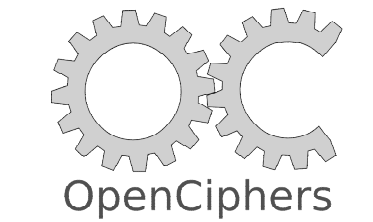

2006 © The OpenCiphers Project



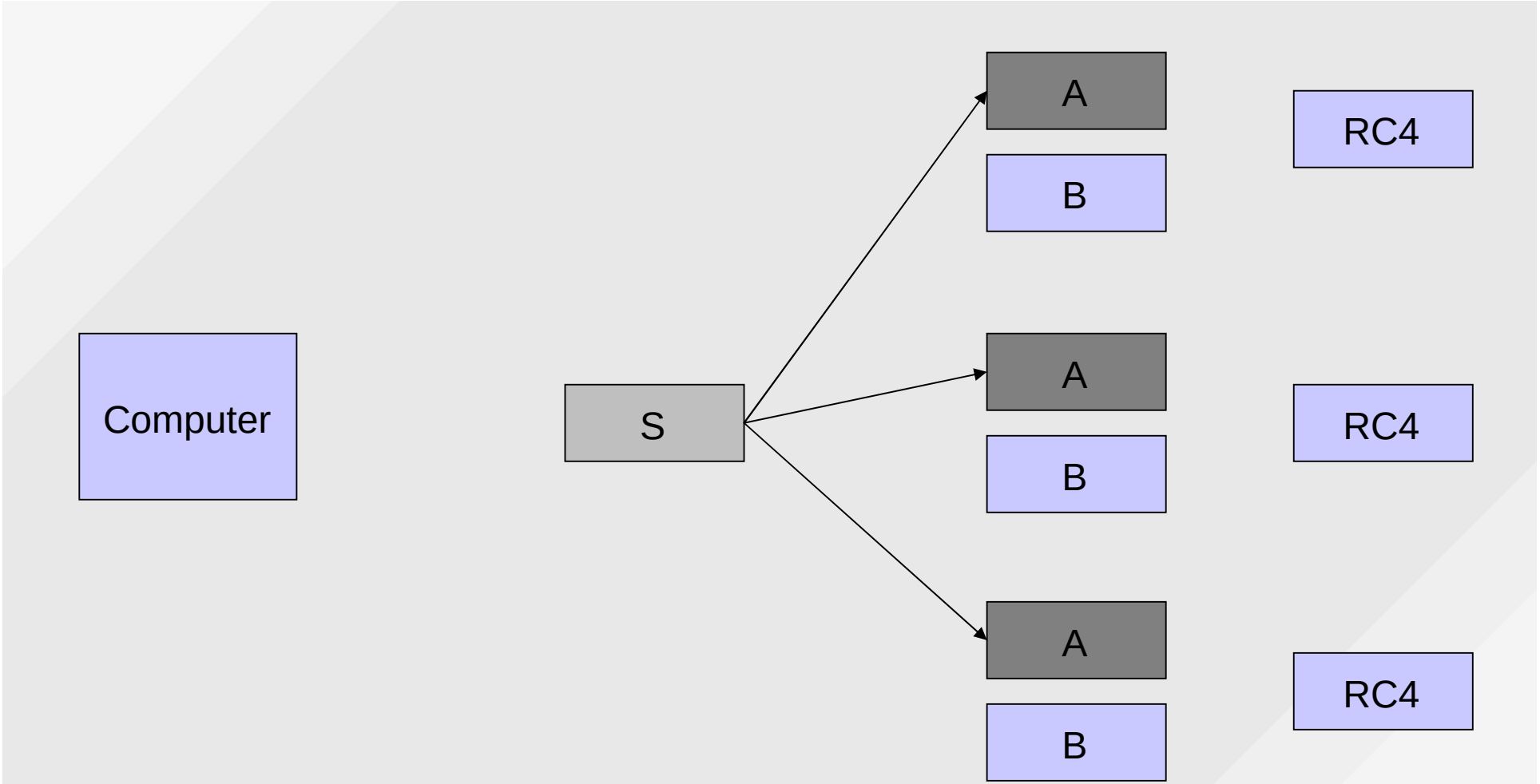
pico-wepcrack



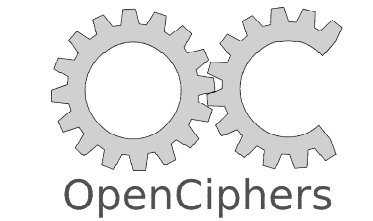
2006 © The OpenCiphers Project



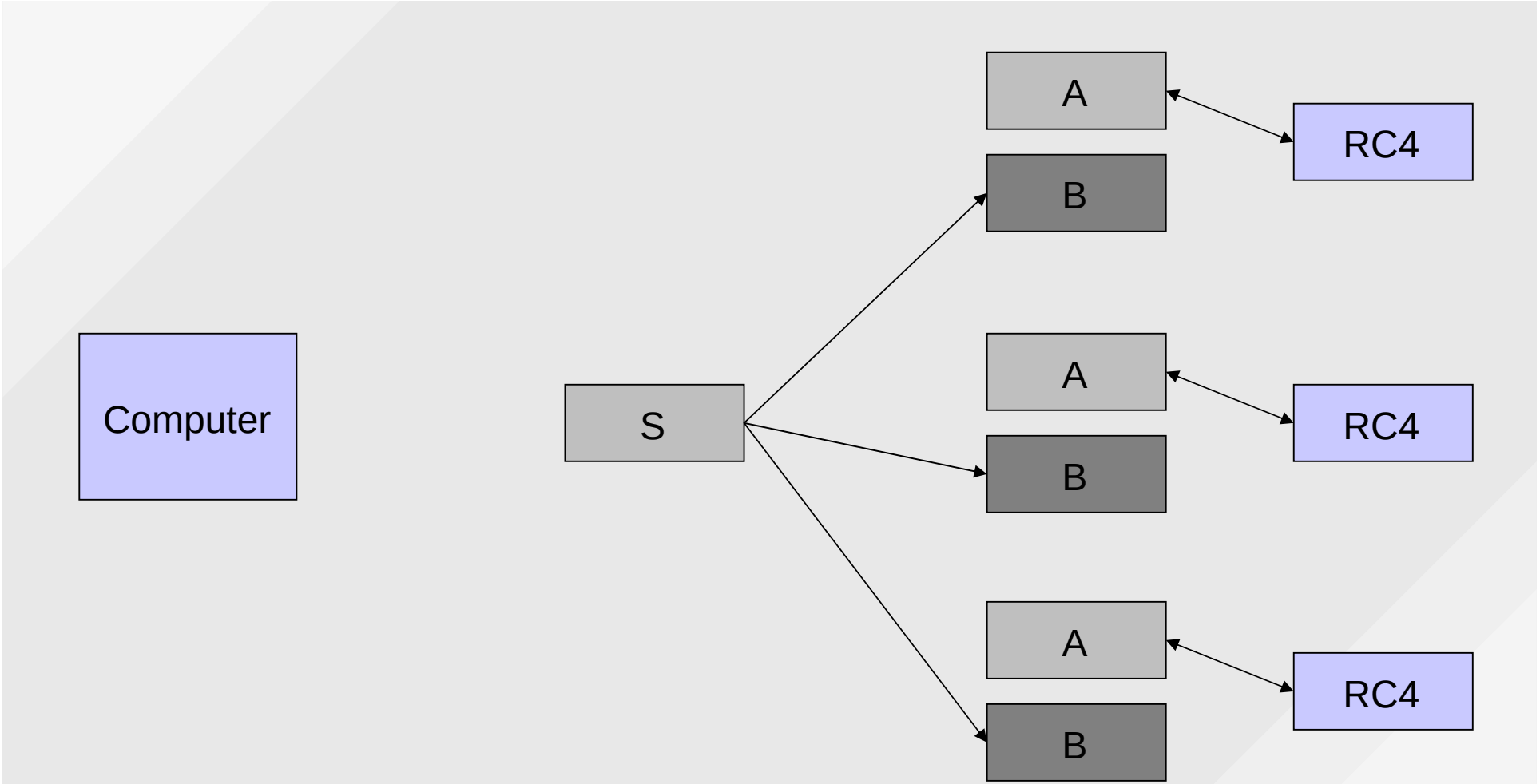
pico-wepcrack



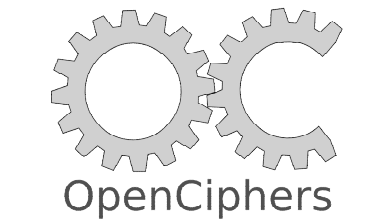
2006 © The OpenCiphers Project



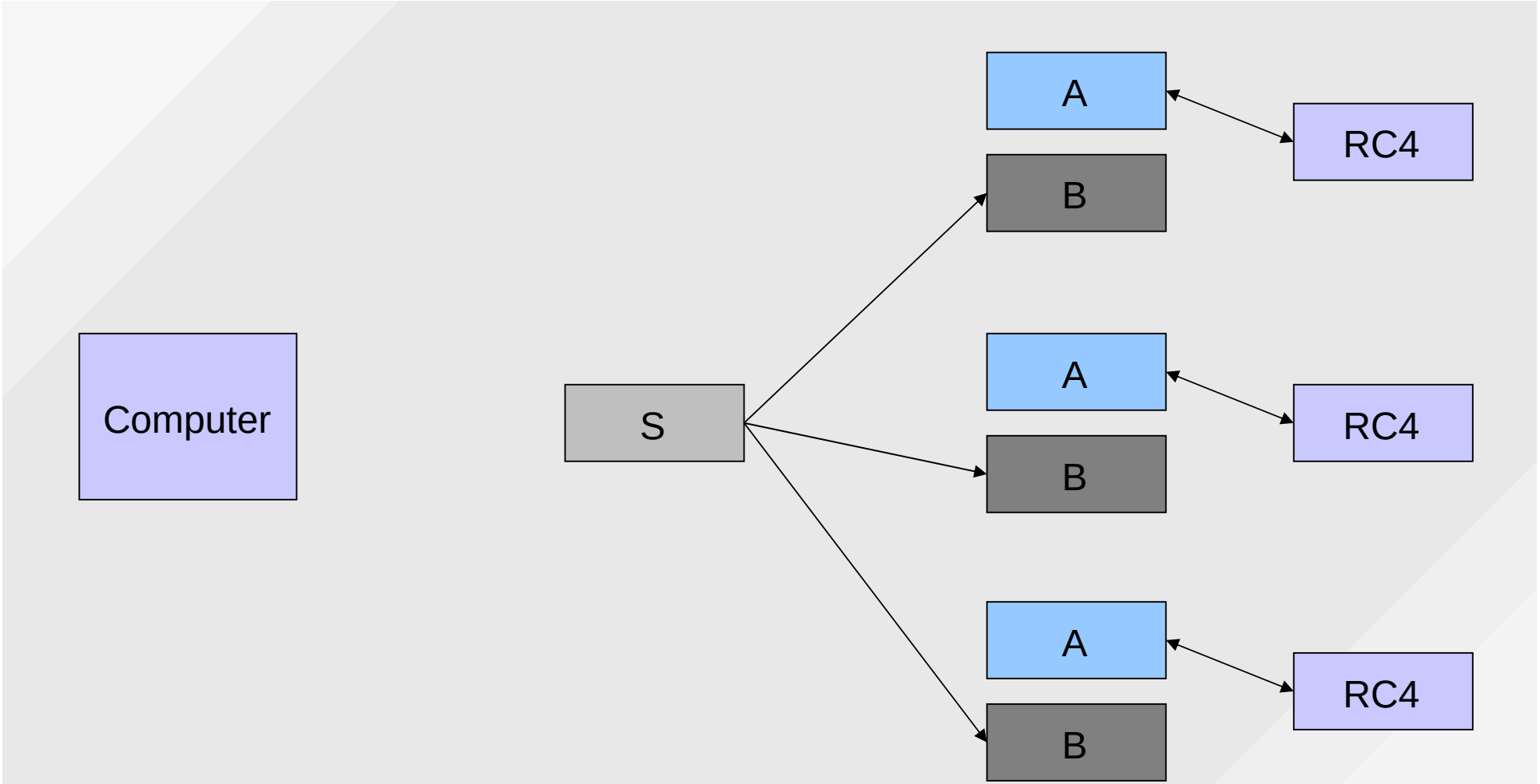
pico-wepcrack



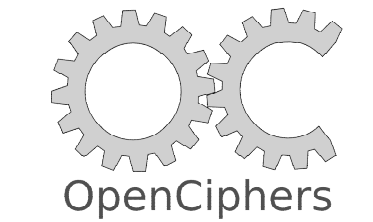
2006 © The OpenCiphers Project



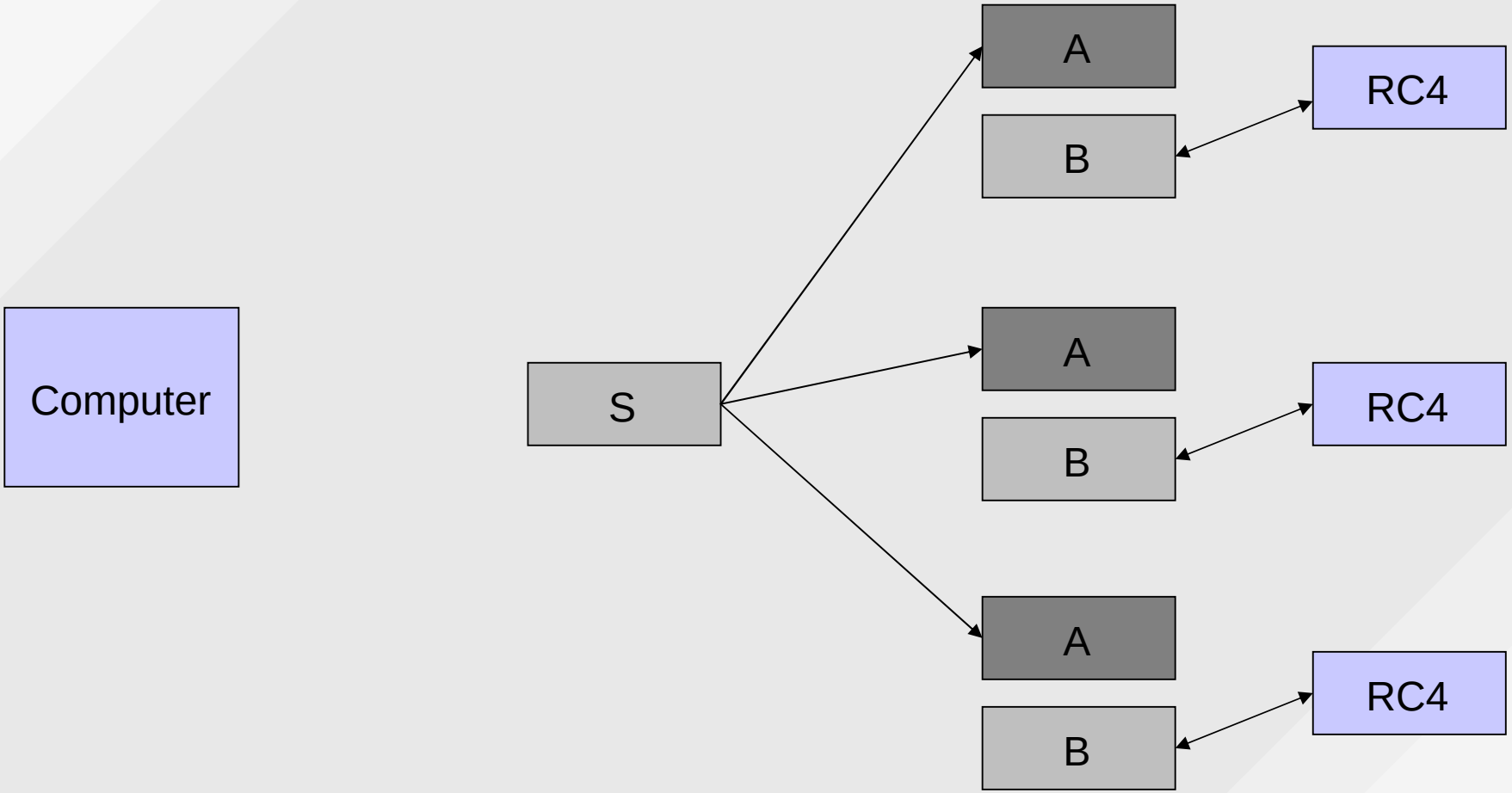
pico-wepcrack



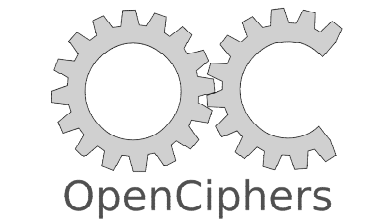
2006 © The OpenCiphers Project



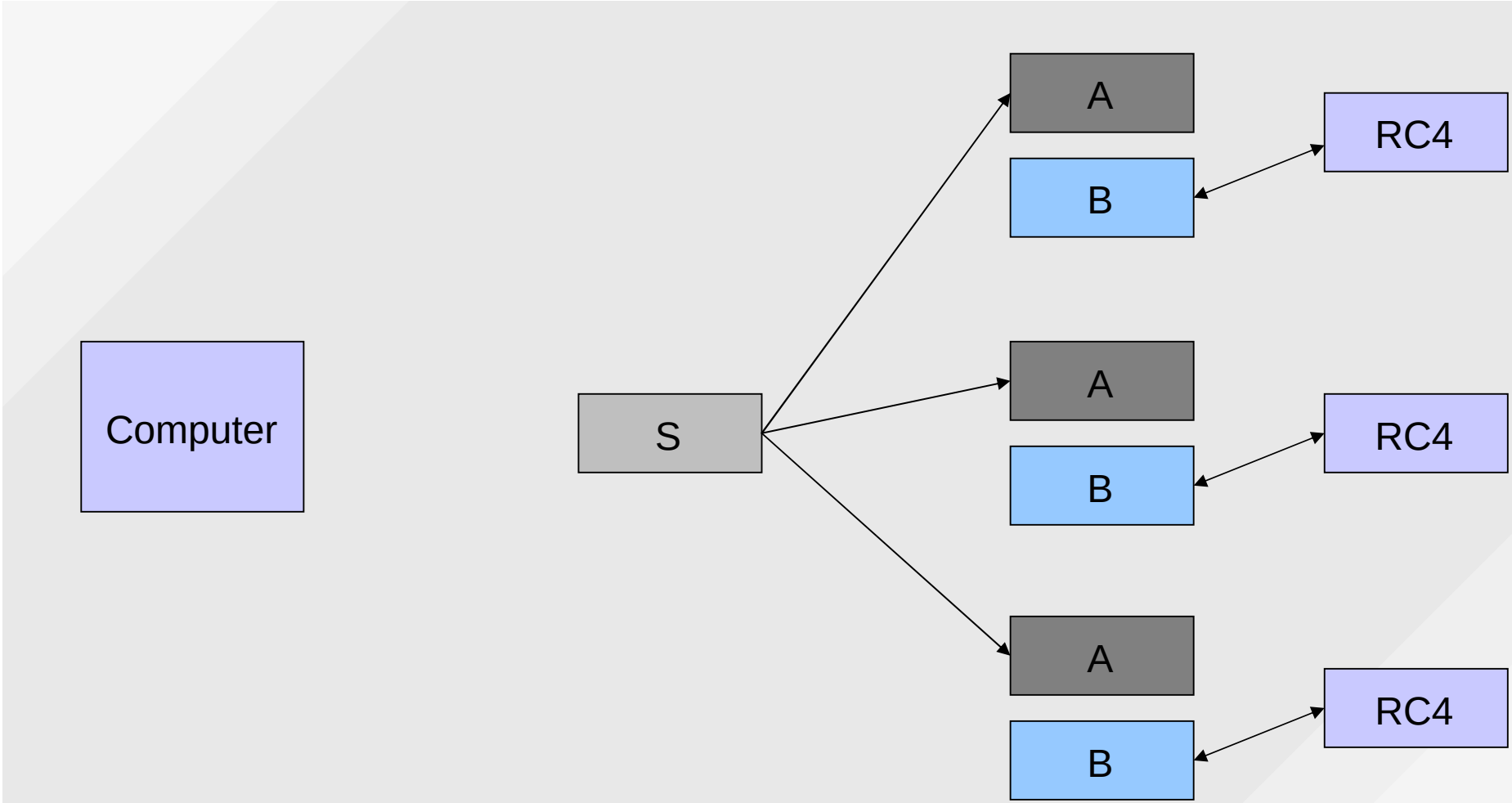
pico-wepcrack



2006 © The OpenCiphers Project

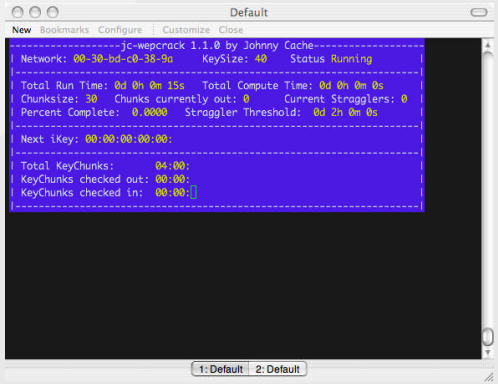


pico-wepcrack

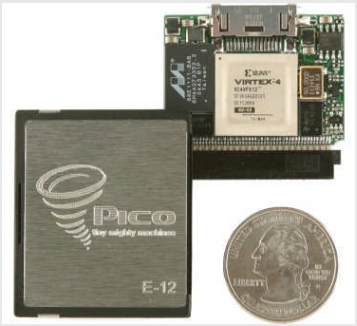


2006 © The OpenCiphers Project

pico-wepcrack



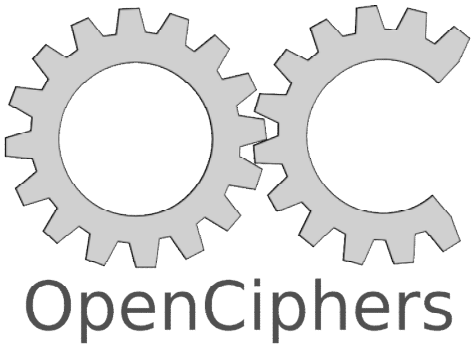
+



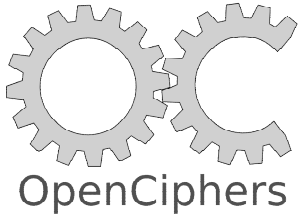
+

= ?

2006 © The OpenCiphers Project



Demo



Performance Comparison

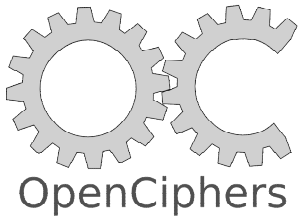
PC

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

FPGA

- pico-wepcrack
  - LX25 ~2,000,000/sec
  - 15 Cluster ~30,000,000/sec
  - FX60 ~5,000,000/sec

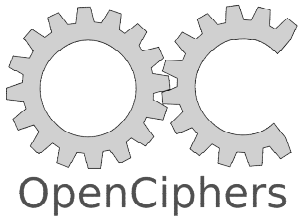
2006 © The OpenCiphers Project



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!)

2006 © The OpenCiphers Project



Questions?

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

- Johnny Cache
  - johnycsh@gmail.com
  - http://www.802.11mercenary.net
- David Hulton
  - dhulton@openciphers.org
  - http://www.openciphers.org
  - http://www.picocomputing.com
- Beetle
  - http://www.shmoocon.org

2006 © The OpenCiphers Project