Faster PwninG Assured: New Adventures with FPGAs

ClubHACK 2007

David Hulton <david@toorcon.org>

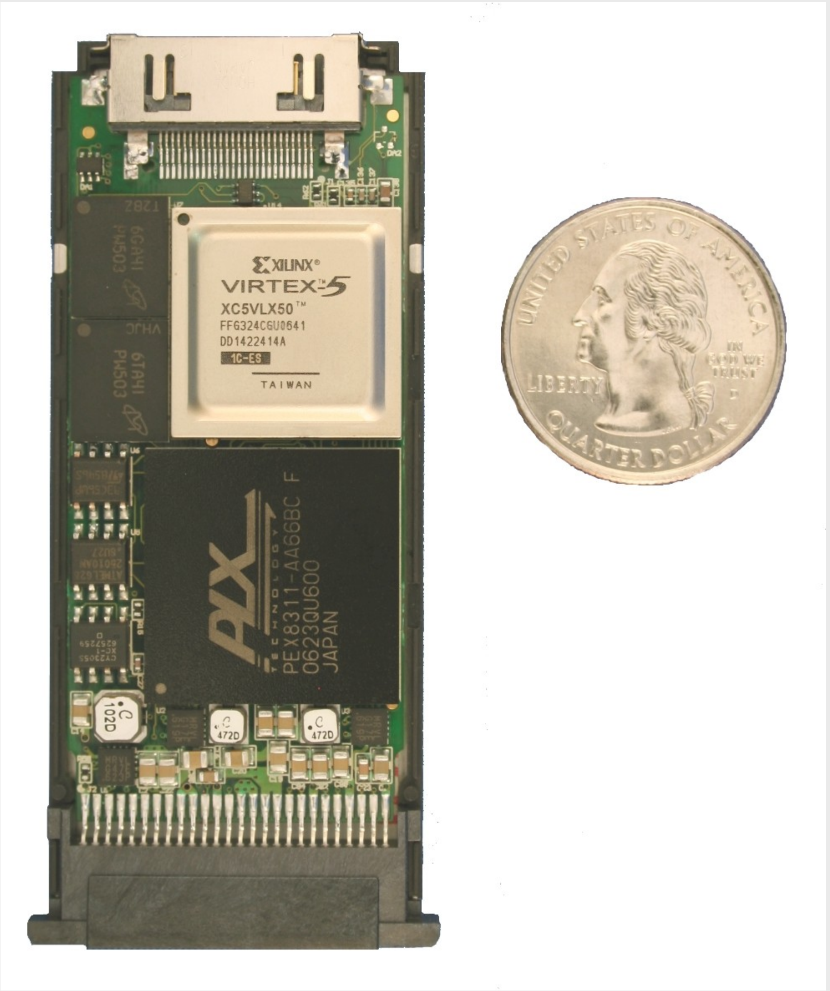
Chairman, ToorCon
Director Security Applications, Pico Computing, Inc.
Researcher, The OpenCiphers Project

Midnight Research Labs
The Church of WiFi
The Hacker's Choice

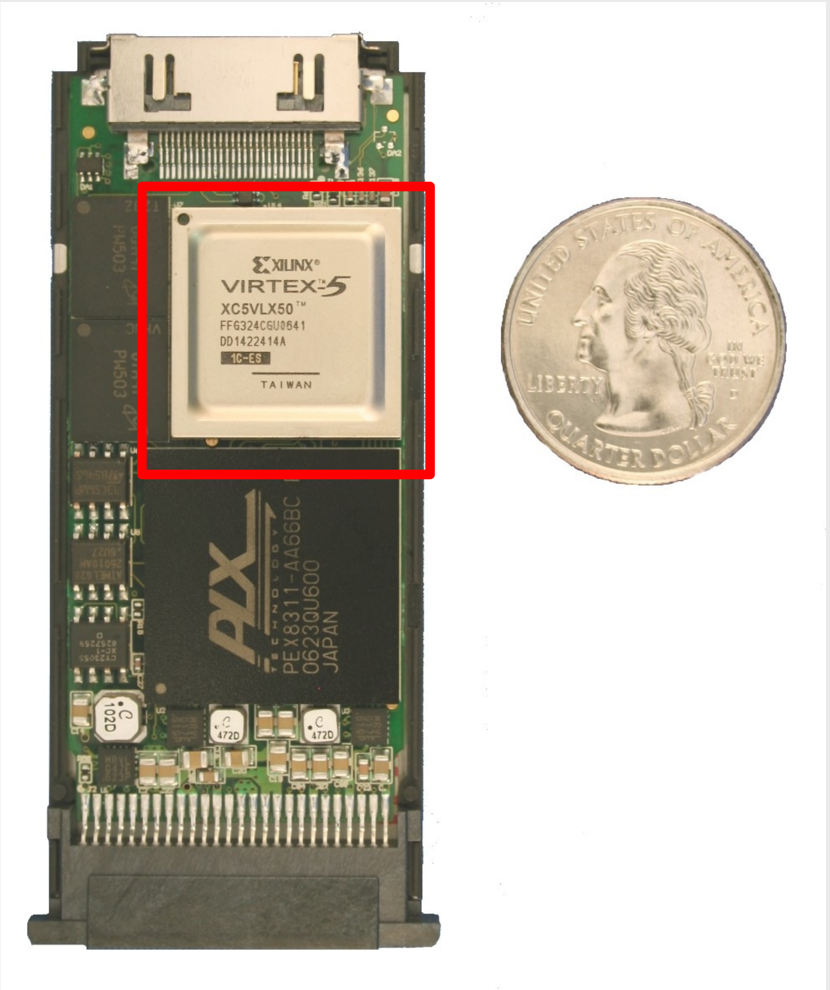
Overview

- FPGAs – Quick Intro
- New to 2007! (Since Last Defcon)
  - CoWPAtty – WPA Cracking
  - VileFault – Mac OS-X FileVault
- New Cracking Tools! (Since ShmooCon)
  - BTCrack – Bluetooth Authentication
  - WinZipCrack – WinZip AES Encryption
  - The A5 Cracking Project – GSM Encryption
- Conclusions

FPGAs


XILINX
VIRTEX-5
XC5VLX50
FFG324C6U0641
DD1422414A
1C-ES
TAIWAN
PLX TECHNOLOGY
PEX8311-AA66BC F
0623QU600
JAPAN

FPGAs

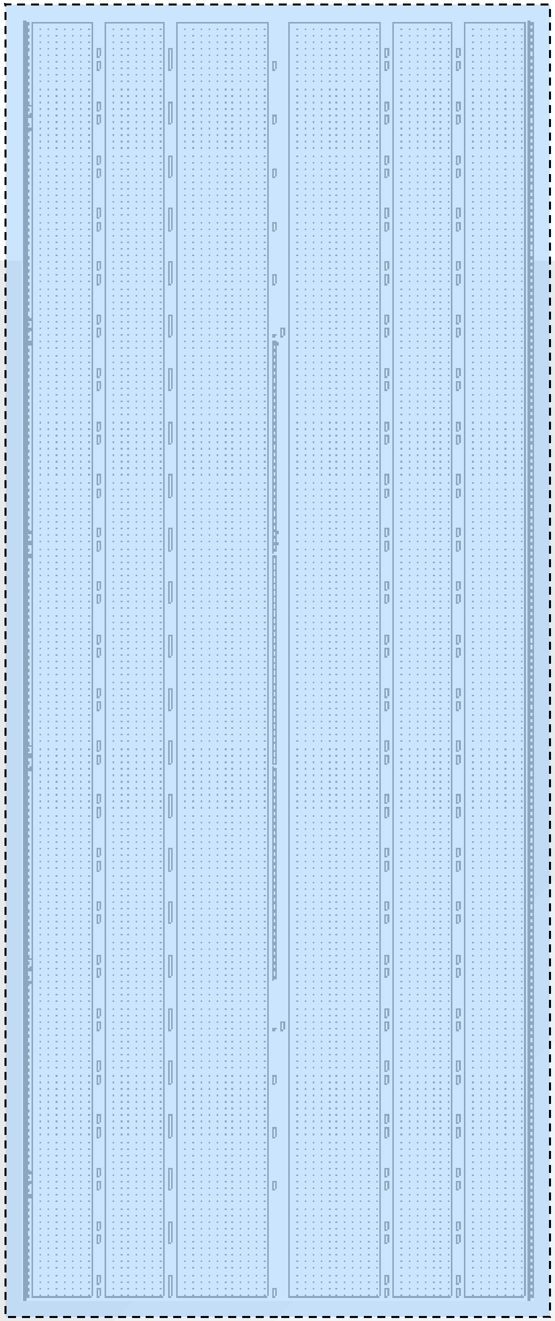

XILINX
VIRTEX-5
XC5VLX50
FFG324C6U0641
DD1422414A
1C-ES
TAIWAN
PLX TECHNOLOGY
PEX8311-AA66BC
0623QU600
JAPAN
UNITED STATES OF AMERICA
LIBERTY
IN GOD WE TRUST
QUARTER DOLLAR

FPGAs

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

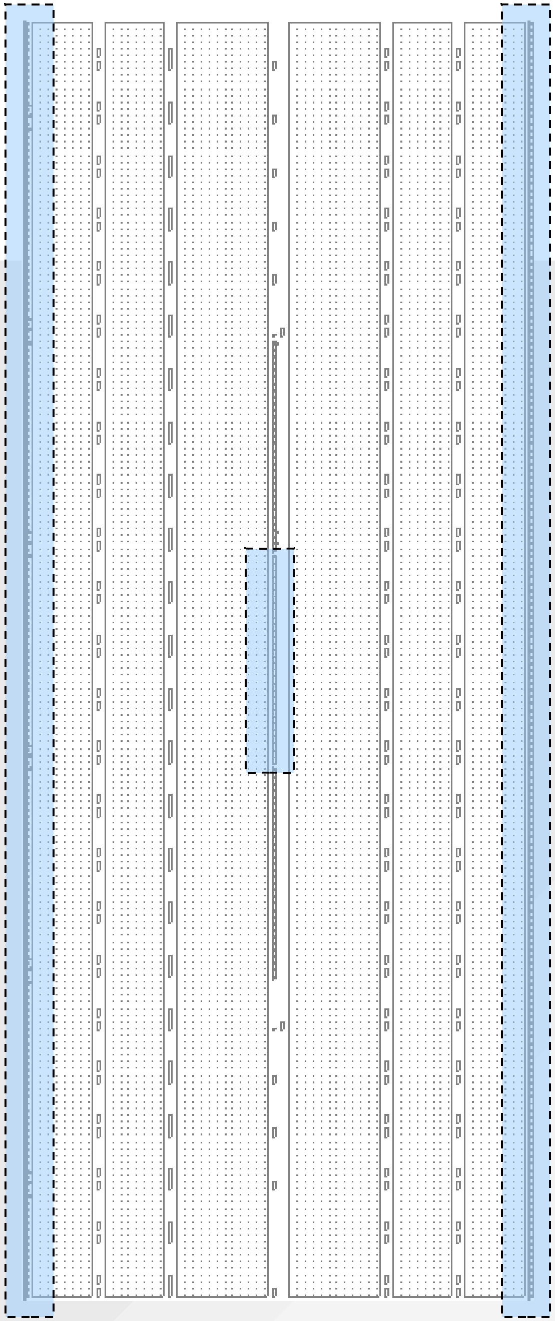
FPGAs

- Virtex-4 LX25



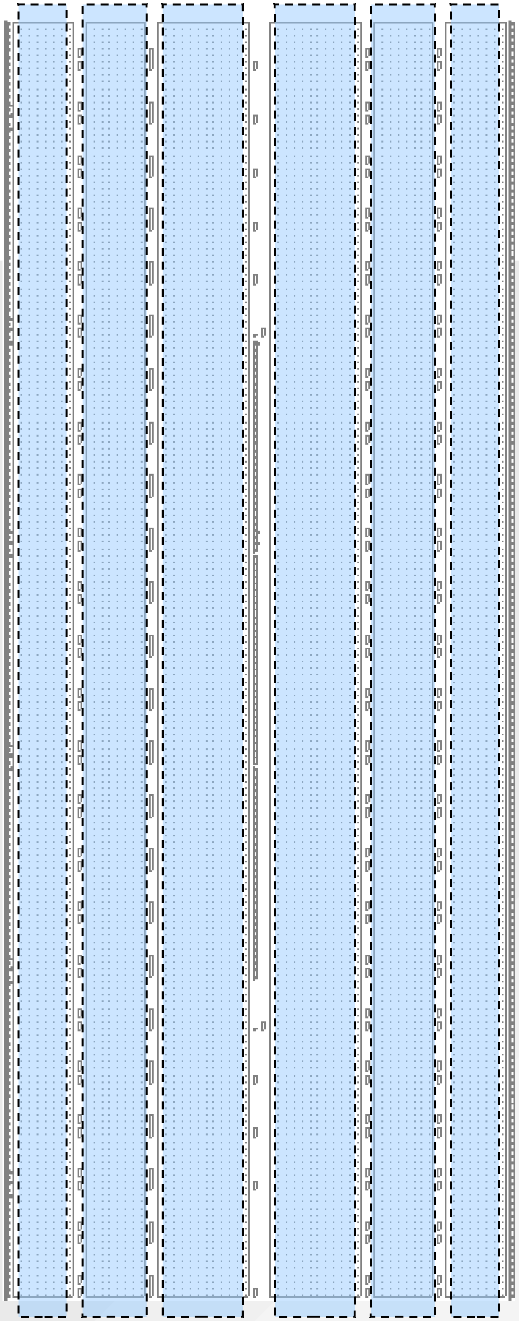
FPGAs

- Virtex-4 LX25
  - IOBs (448)



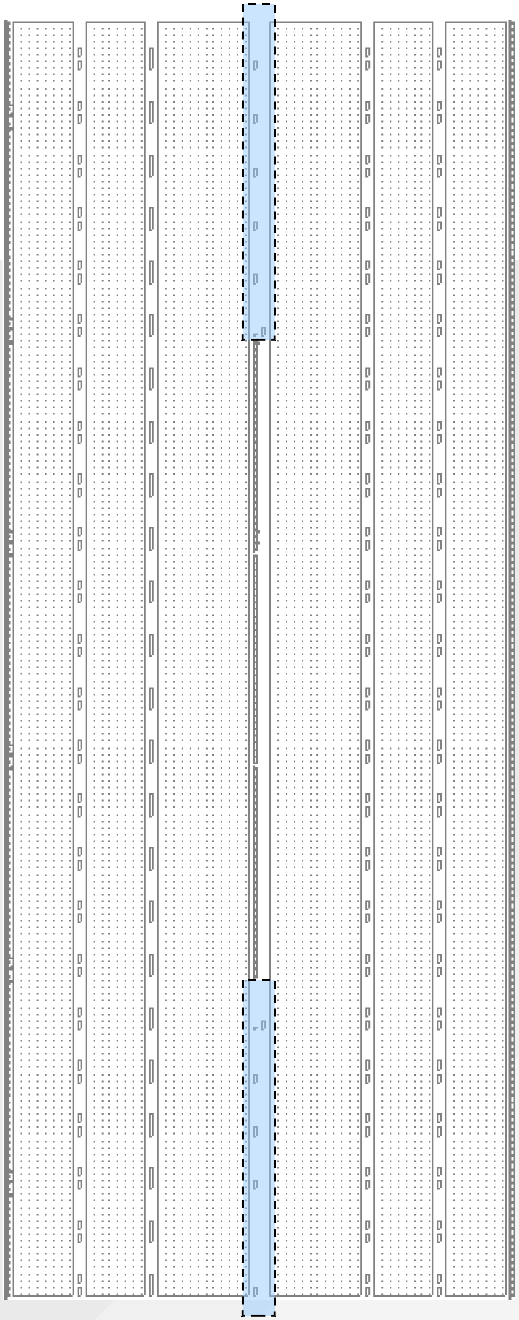
FPGAs

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



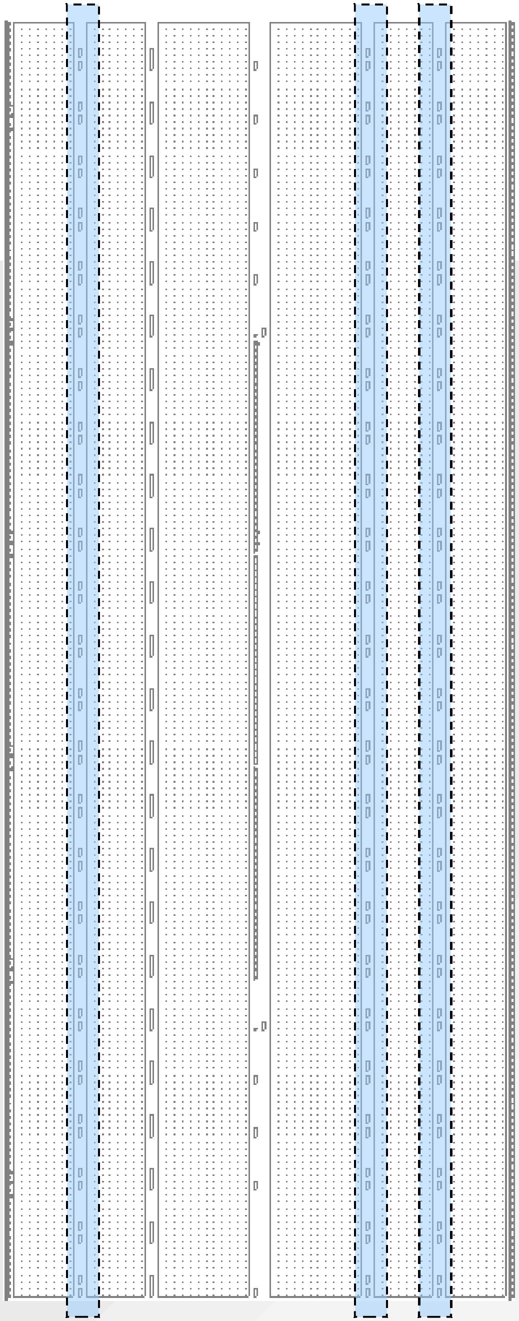
FPGAs

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



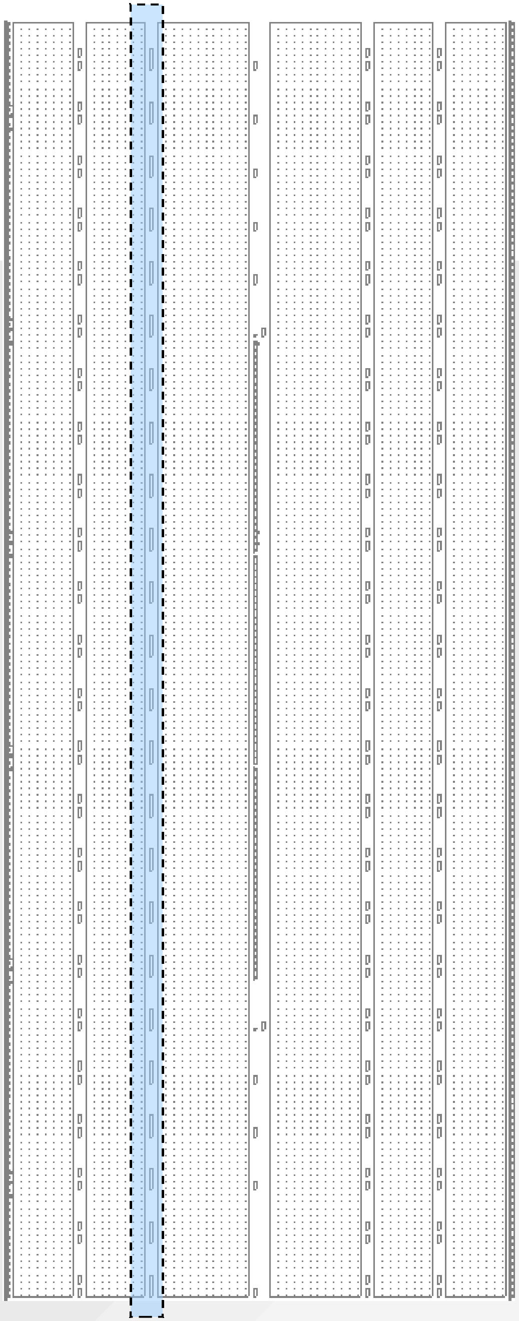
FPGAs

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



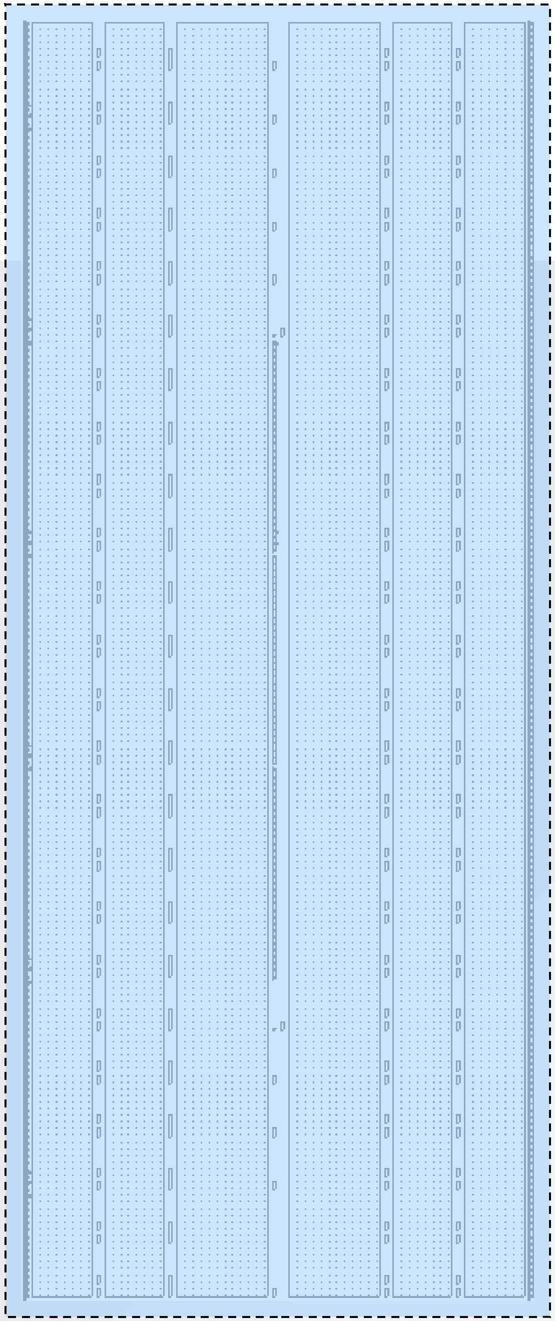
FPGAs

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



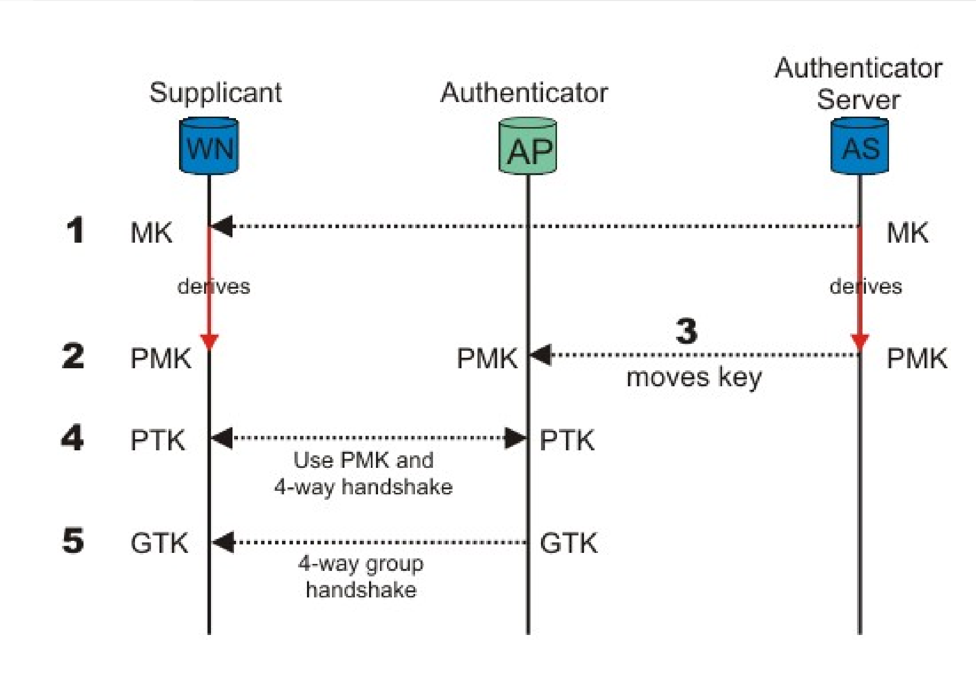
FPGAs

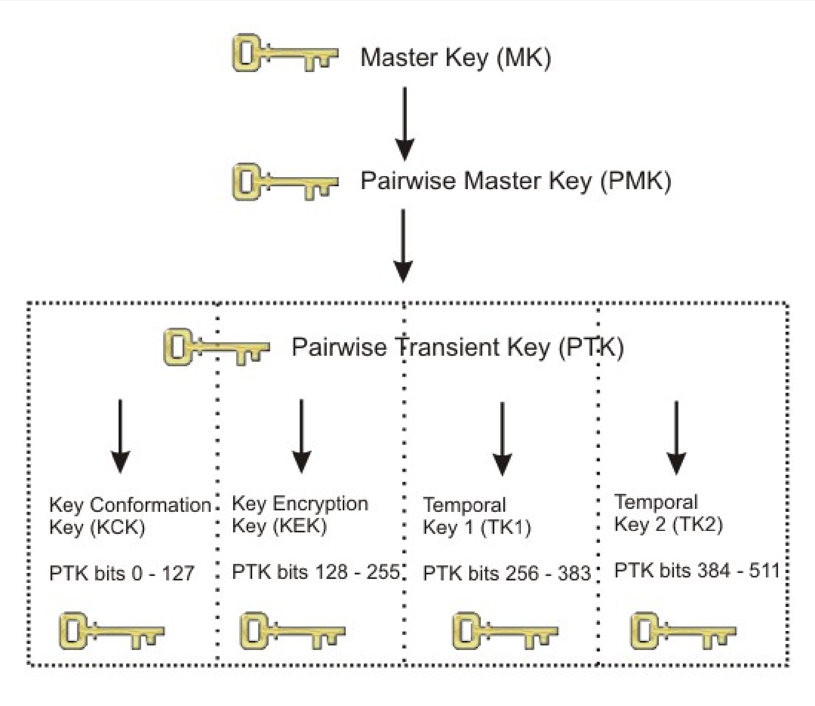
- Virtex-4 LX25
  - IOBs
  - Slices
  - DCMs
  - BlockRAMs
  - DSP48s
  - Programmable Routing Matrix (~18 layers)



Introduction to WPA

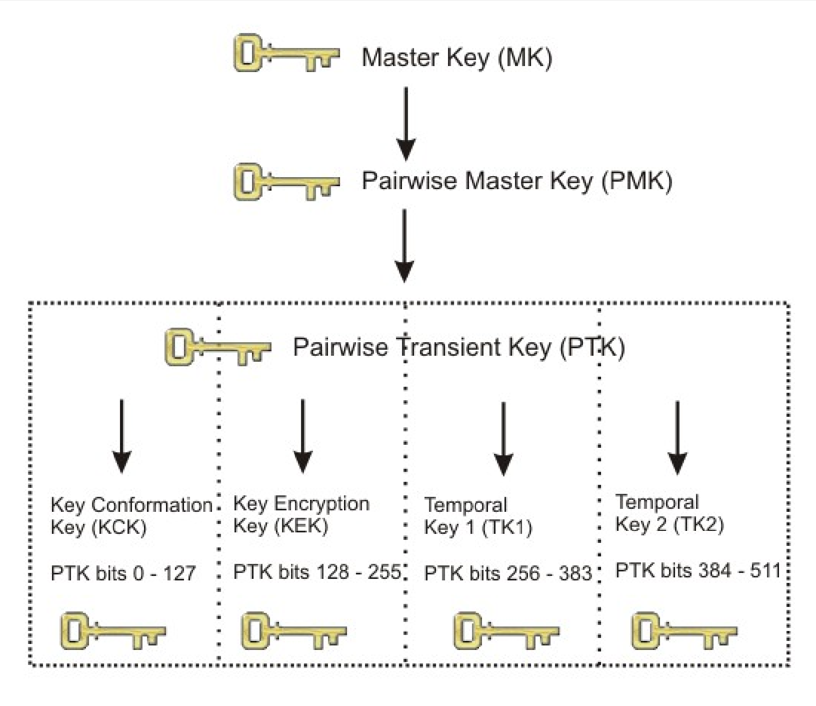
- WiFi Protected Access





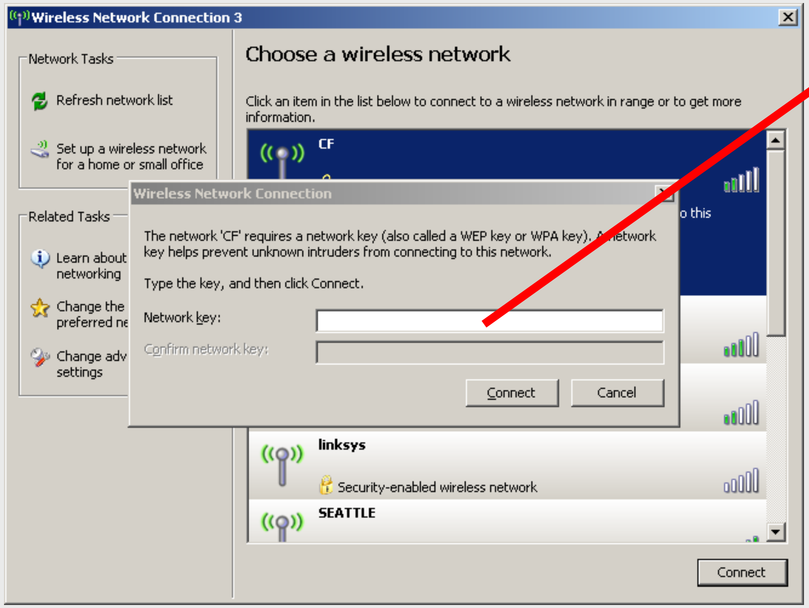
Introduction to WPA

- 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



Wireless Network Connection 3

Network Tasks

Refresh network list

Set up a wireless network for a home or small office

Related Tasks

Learn about networking

Change the preferred ne

Change adv settings

Choose a wireless network

Click an item in the list below to connect to a wireless network in range or to get more information.

CF

o this

Wireless Network Connection

The network 'CF' requires a network key (also called a WEP key or WPA key). A network key helps prevent unknown intruders from connecting to this network.

Type the key, and then click Connect.

Network key:

Confirm network key:

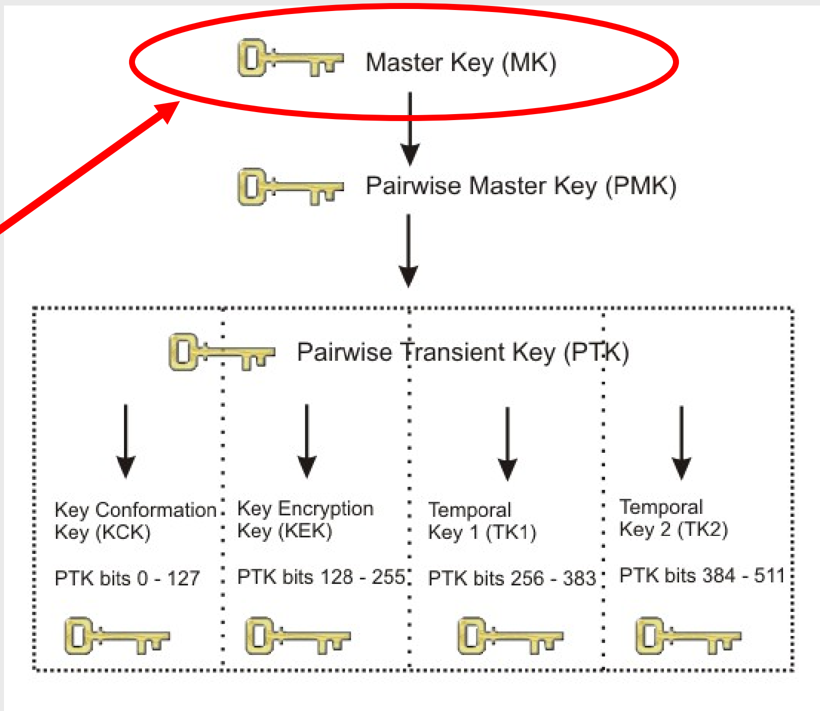
Connect | Cancel

linksys

Security-enabled wireless network

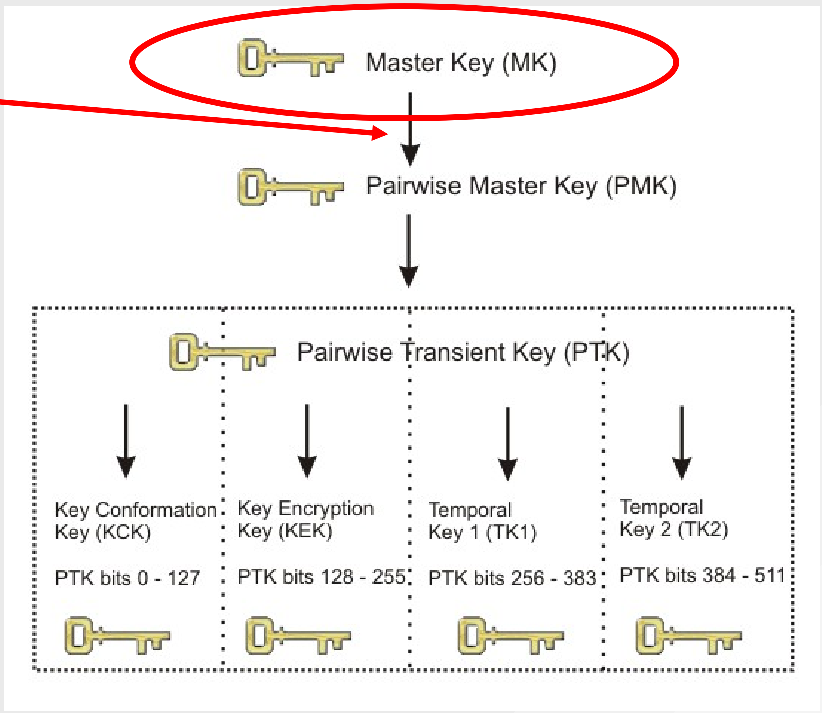
SEATTLE

Connect



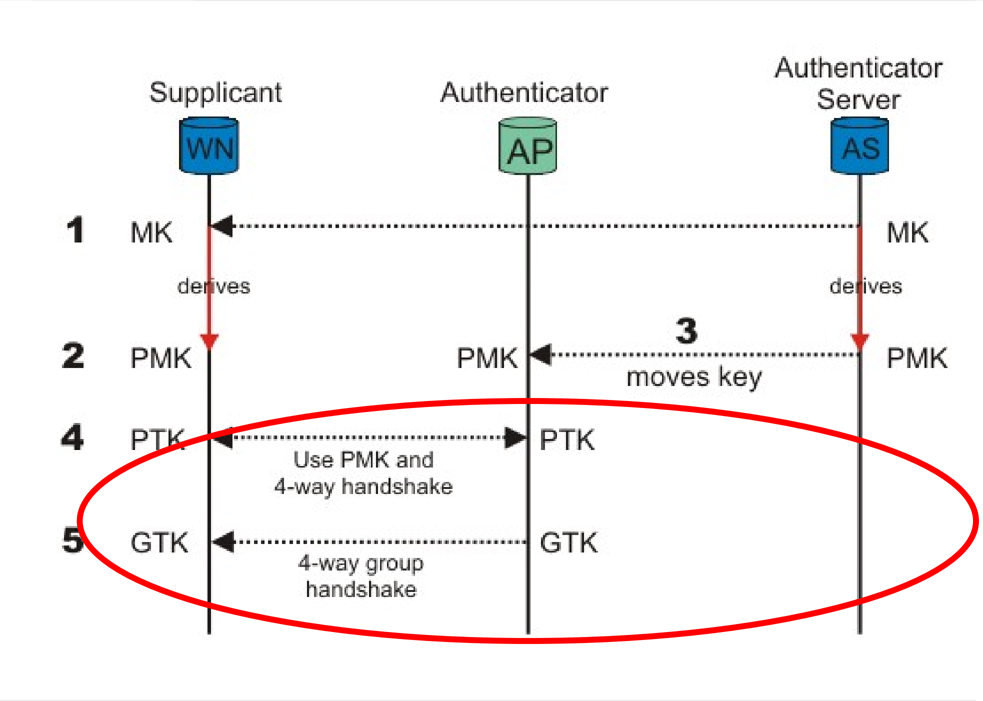
Introduction to WPA

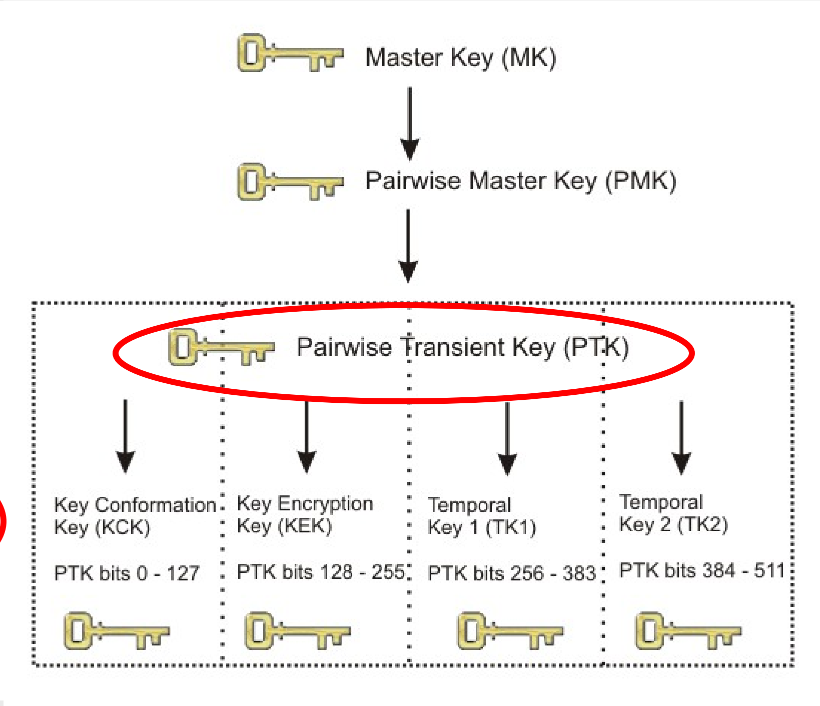
- PSK
  - MK is your passphrase
  - It's run through PBKDF2 to generate the PMK



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

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

OpenCiphers

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

- Finally have the 40GB WPA tables on the tubes
- Thanks Shmoo! (3ricJ & Holt!)
- Check the Torrent trackers for seeds
- CoWPAtty FPGA support has recently been added to wicrawl

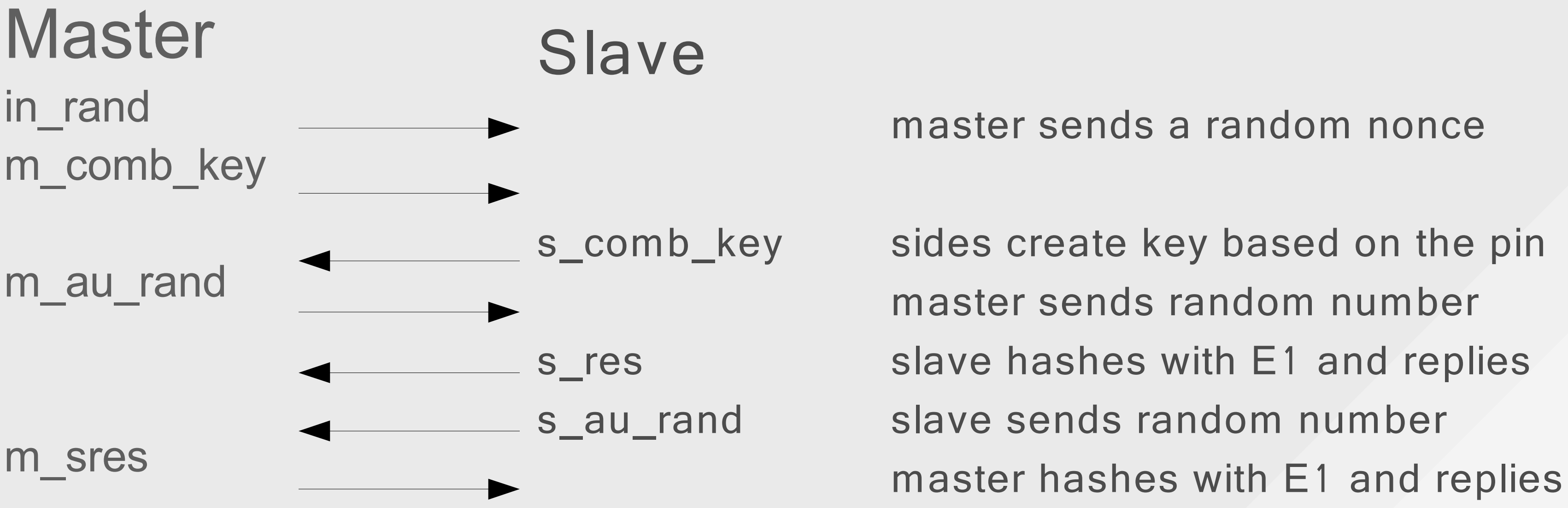
Demo

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 originally demonstrated his implementation at hack.lu

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

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)

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

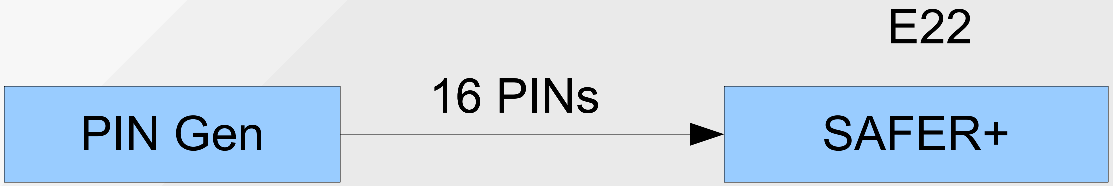
Bluetooth PIN Cracking

```
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!
}
```

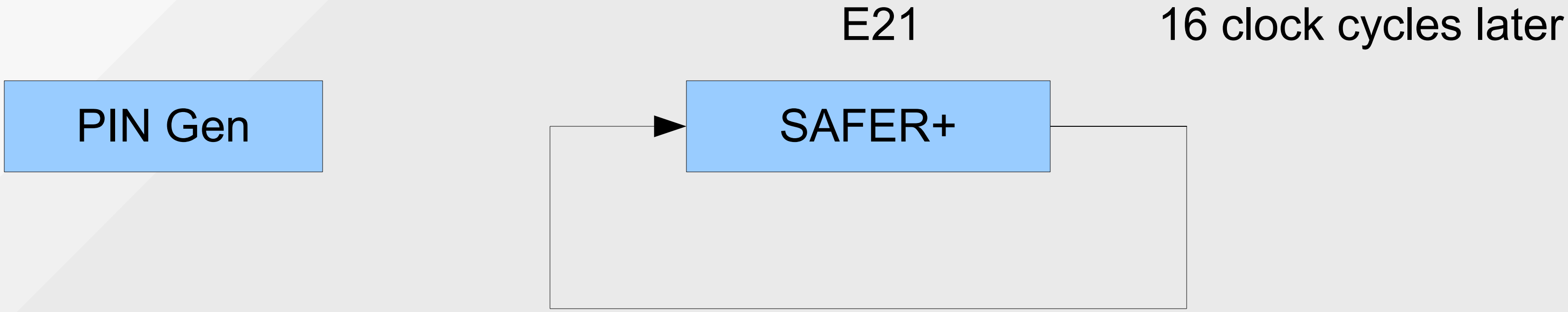
Bluetooth PIN Cracking

```
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!
}
```

Bluetooth PIN Cracking

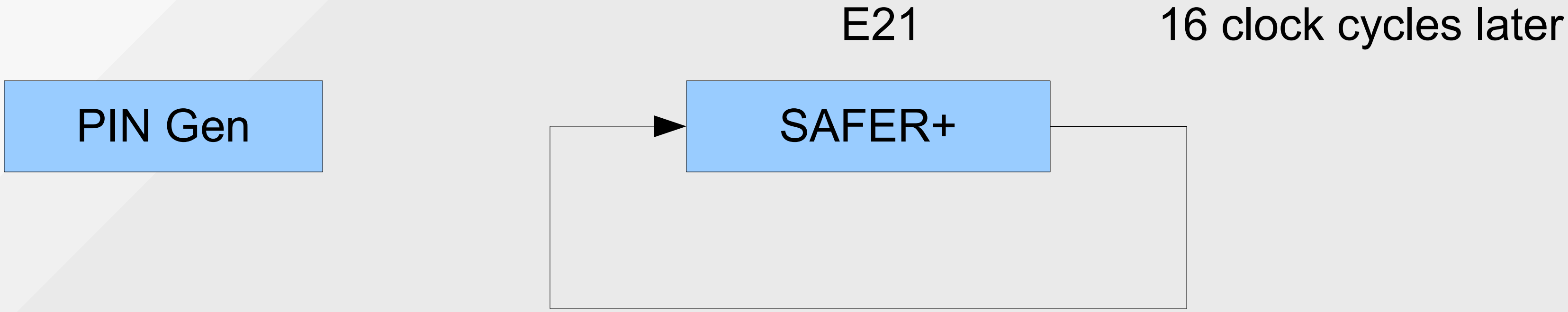


Bluetooth PIN Cracking



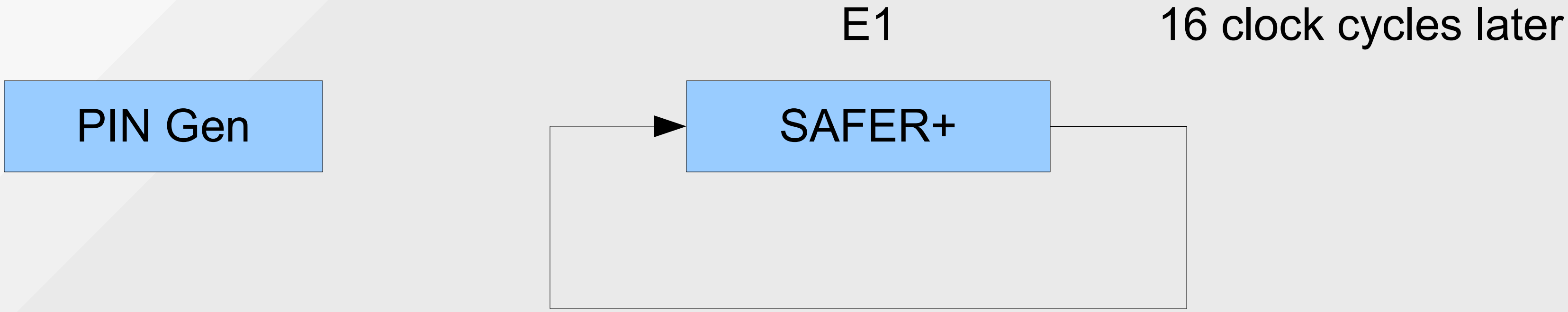
Output loops back and SAFER+ now does
E21 for the Master

Bluetooth PIN Cracking



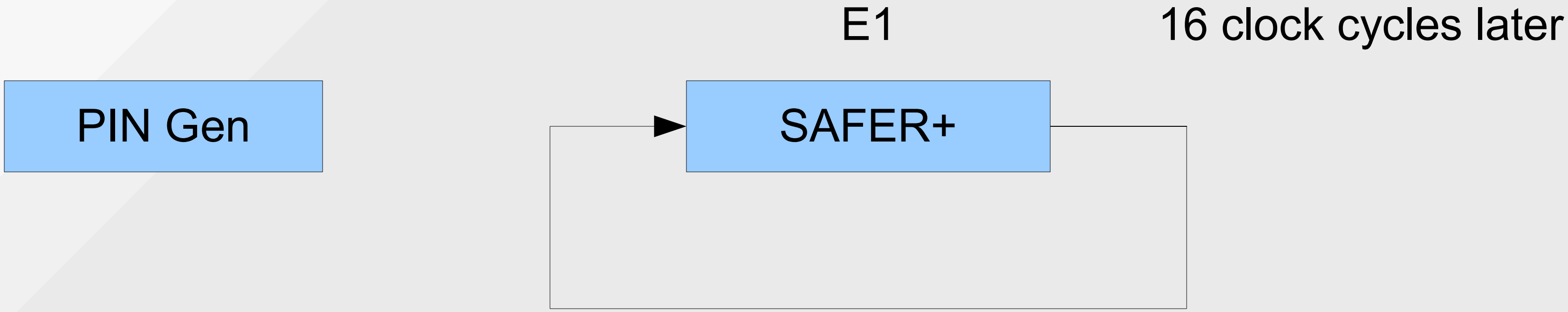
Then does the second E21 for the Slave
and combines the keys to create the link key

Bluetooth PIN Cracking



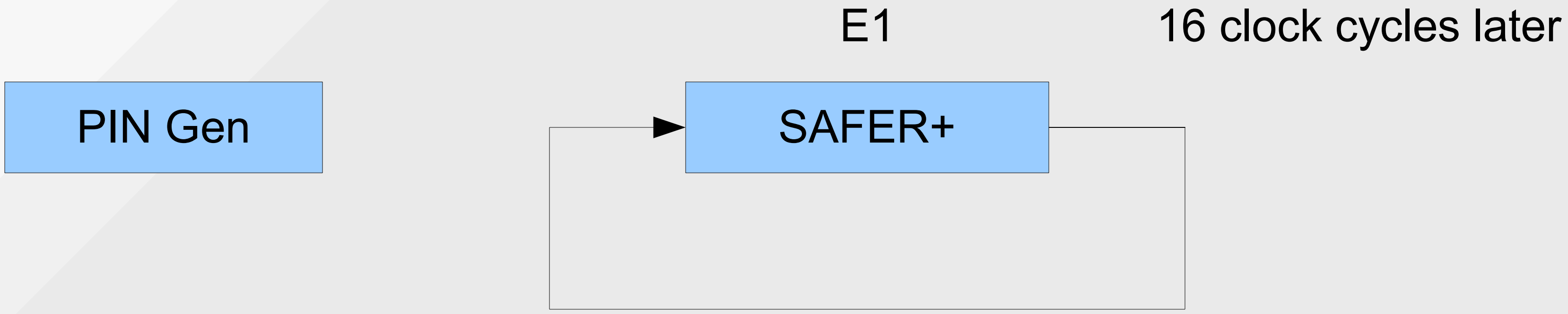
Then the first part of E1 for the Slave

Bluetooth PIN Cracking



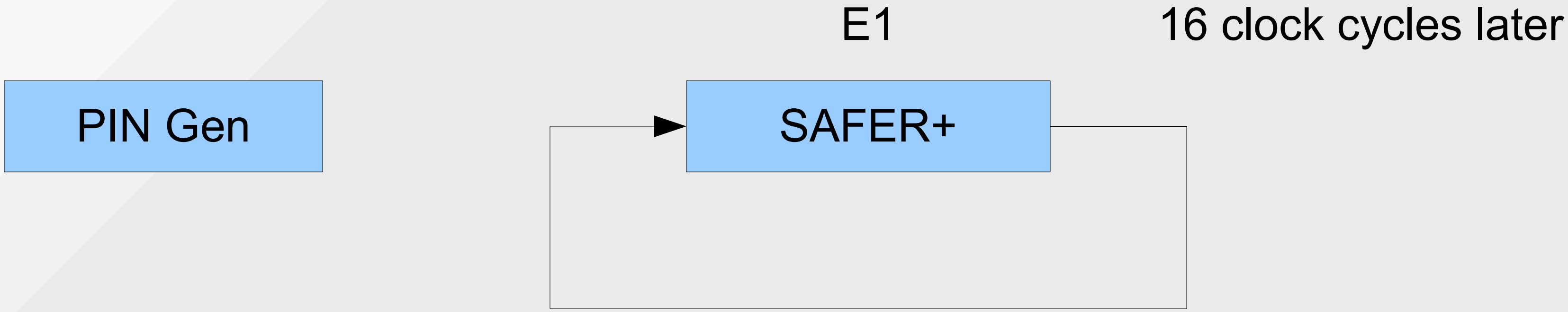
Then the second part of E1 for the Slave

Bluetooth PIN Cracking



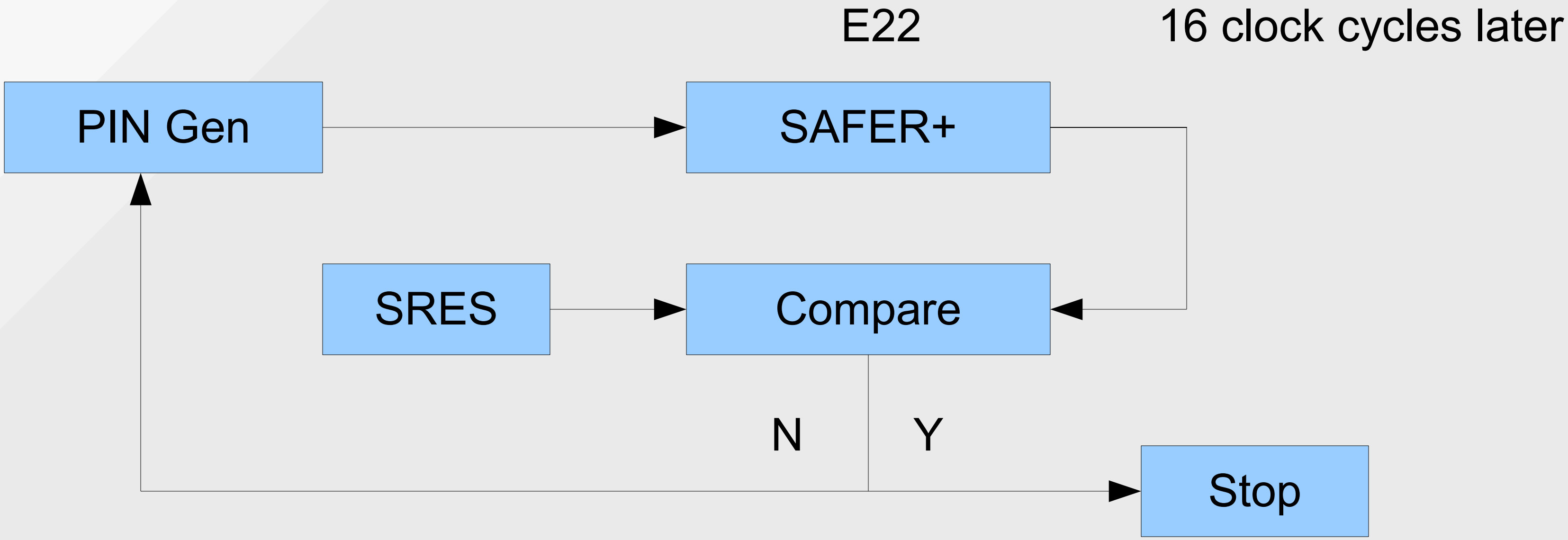
Then the first part of E1 for the Master

Bluetooth PIN Cracking



Then the second part of E1 for the Master

Bluetooth PIN Cracking



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

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
- I built a commandline version
- Thierry Zoller integrated support into BTCrack
- I added some hollywood FX !

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	~7,000,000/sec
15 Cluster	~105,000,000/sec
LX50	~10,000,000/sec

0.001 secs to crack 4 digit

10 secs to crack 8 digit

Demo

WinZip AES Encryption

- Somewhat proprietary standard
- No open source code available (until now!)
- Format
  - Uses the standard ZIP format
  - Adds a new compression type (99)
  - Uses PBKDF2 (1000 iterations) for key derivation
  - Individual files can be encrypted inside the ZIP file
  - Supports 128/192/256-bit key lengths
  - Uses a 16-bit verification value to verify passwords
  - Otherwise you verify by using the checksum
  - Uses a salt (sorry, can't do a dictionary attack!)

WinZip AES Encryption

- Cracking algorithm
  - Scan through ZIP file until you find the encrypted file
  - Get the 16-bit password verification value
  - Hash a password with PBKDF2 and see if the verification value matches
    - No – Try next password
    - Yes – Decrypt file and see if checksum matches
      - No – Try next password
      - Yes – Password found!

WinZip AES Encryption

- Uses the same PBKDF2 core as the WPA and FileVault cracking code
- Requires extra iterations for longer key lengths
- Tool takes a ZIP file, encrypted file name, and dictionary file as input

Performance Comparison

PC		FPGA	
winzipcrack		**winzipcrack**	
800MHz P3	~100/sec	LX25	~2,000/sec
3.6GHz P4	~180/sec	LX50	~6,000/sec
AMD Opteron	~200/sec	15 Cluster	~30,000/sec
2.16GHz IntelDuo	~200/sec		

Demo

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.”

FileVault

Increased security for your computer.

VileFault

- We wanted to know what was happening

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

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

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)

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)

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

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	~6,000/sec
15 Cluster	~30,000/sec

Demo

The A5 Cracking Project

- Capturing GSM Traffic
  - GNU Radio USRP board ($900 USD)
  - We developed software to decode GSM
  - Lets you fire up wireshark on a GSM channel
  - Can sometimes capture SMS messages
  - Couldn't capture voice calls :-(
  - We wanted to change that

The A5 Cracking Project

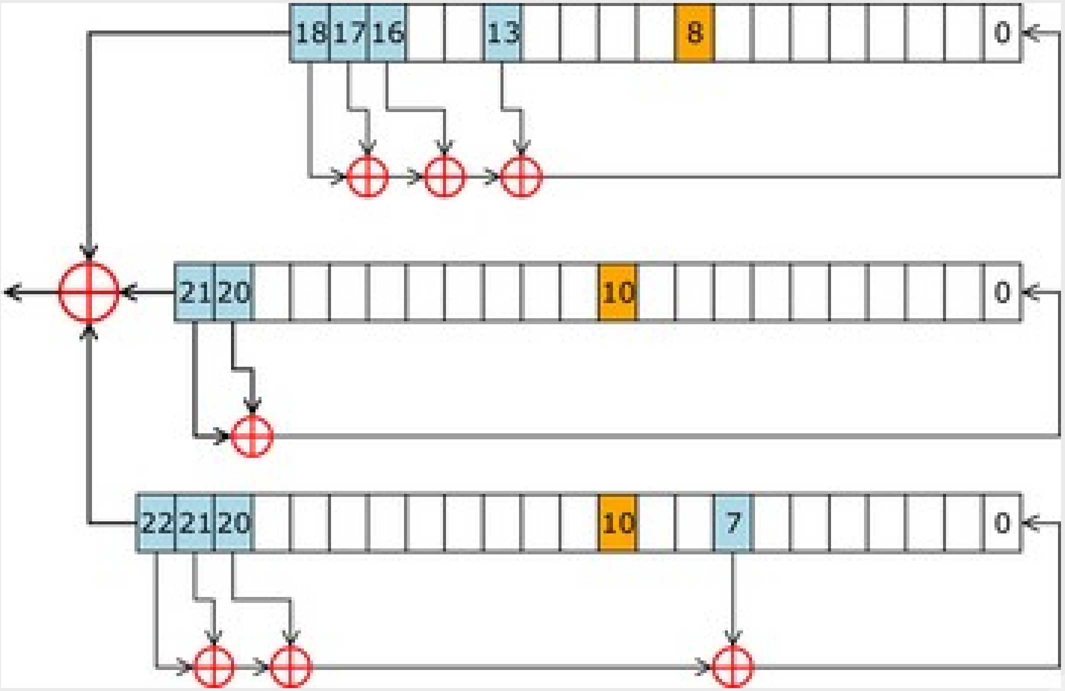
- Luckily you don't need to break crypto
- India
  - IDEA - A5/0
  - AirTel - A5/0
  - Essar - A5/0
  - Orange - A5/0
  - Dolphin - A5/0

The A5 Cracking Project

- Focusing on A5/1
  - Used widely throughout the US, Europe, and some Asian countries
  - The strongest algorithm for GSM (3G is better)
  - Looking at only practical attacks
  - Originally looked at ciphertext only attacks
  - Found out that there is a lot of known-plaintext
  - Known-plaintext attacks are a lot easier
- Researched a few different attacks
  - Real-time attack with known-plaintext + FPGAs
    - Anderson & Roe / Keller + our mods
  - Pre-computation + less FPGAs
    - Biryukov, Shamir, & Wagner + our mods

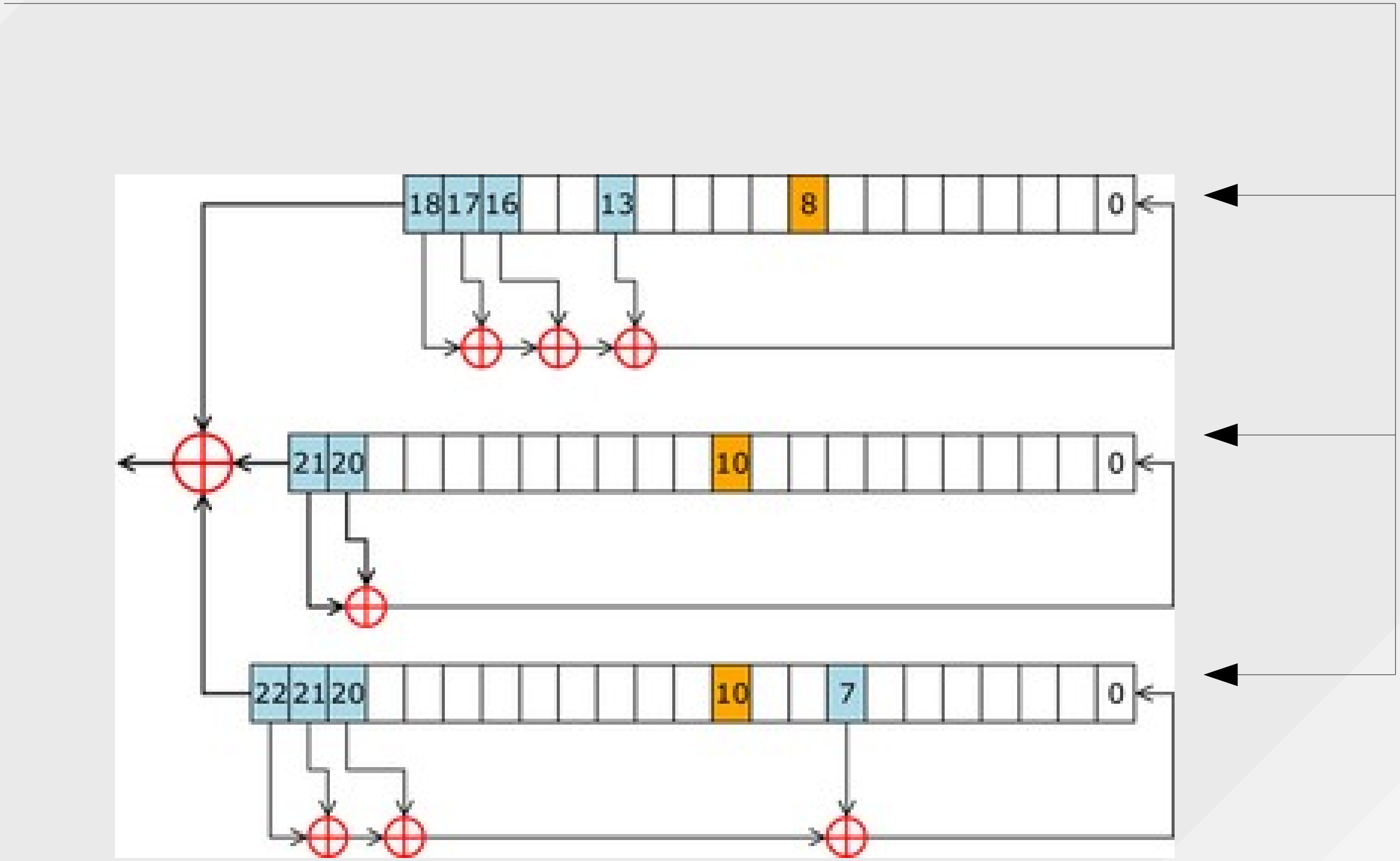
Real-time Attack

- A register is clocked if it's clocking bit agrees with the majority



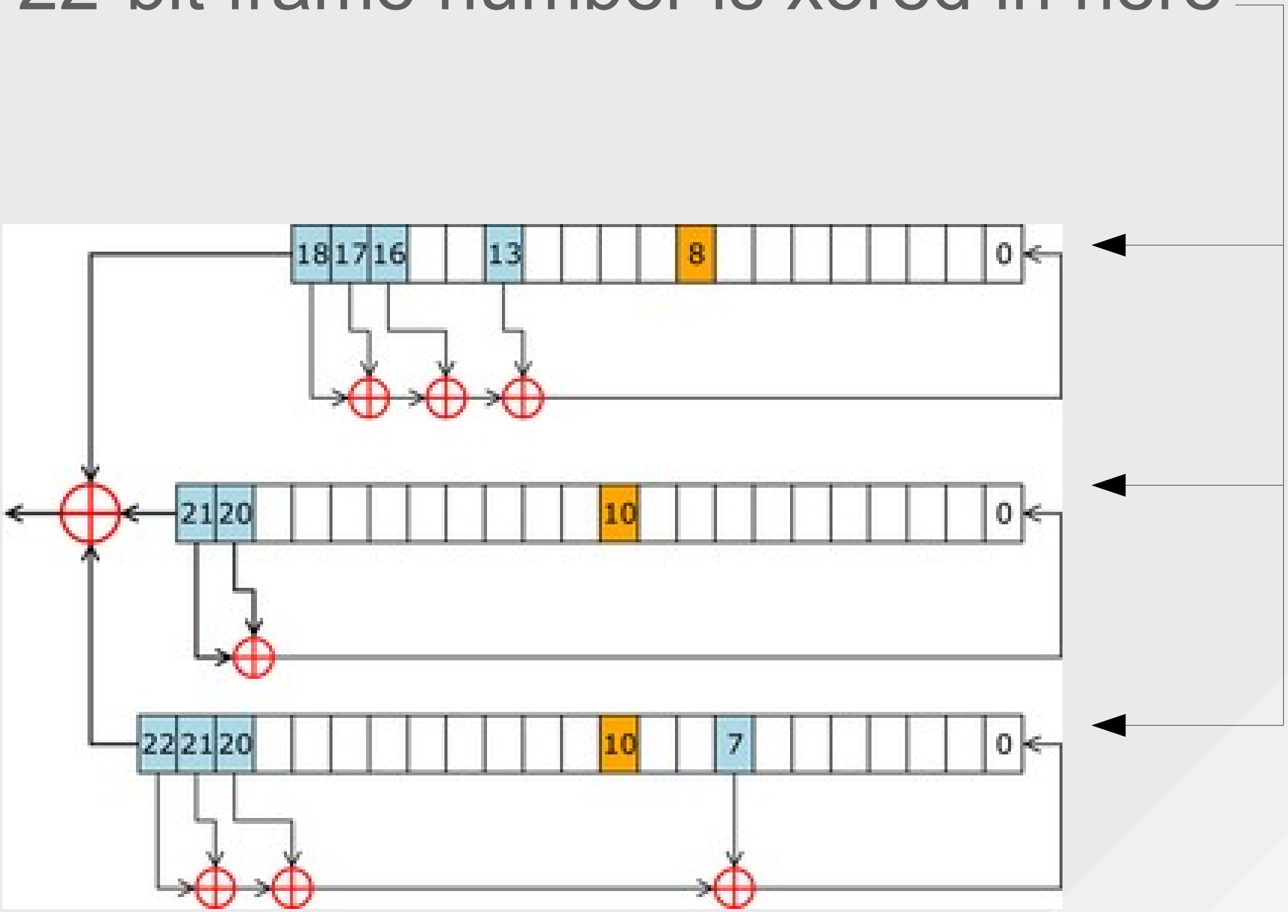
Real-time Attack

- First 64 clock cycles key is xor'ed with registers here



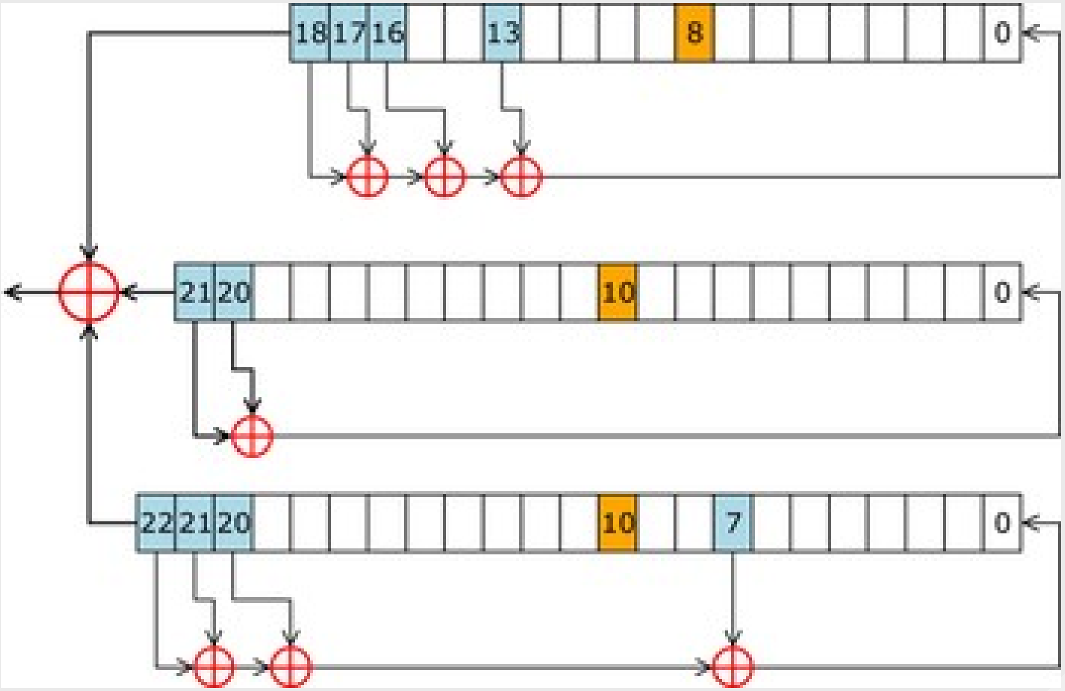
Real-time Attack

- Second, 22-bit frame number is xored in here



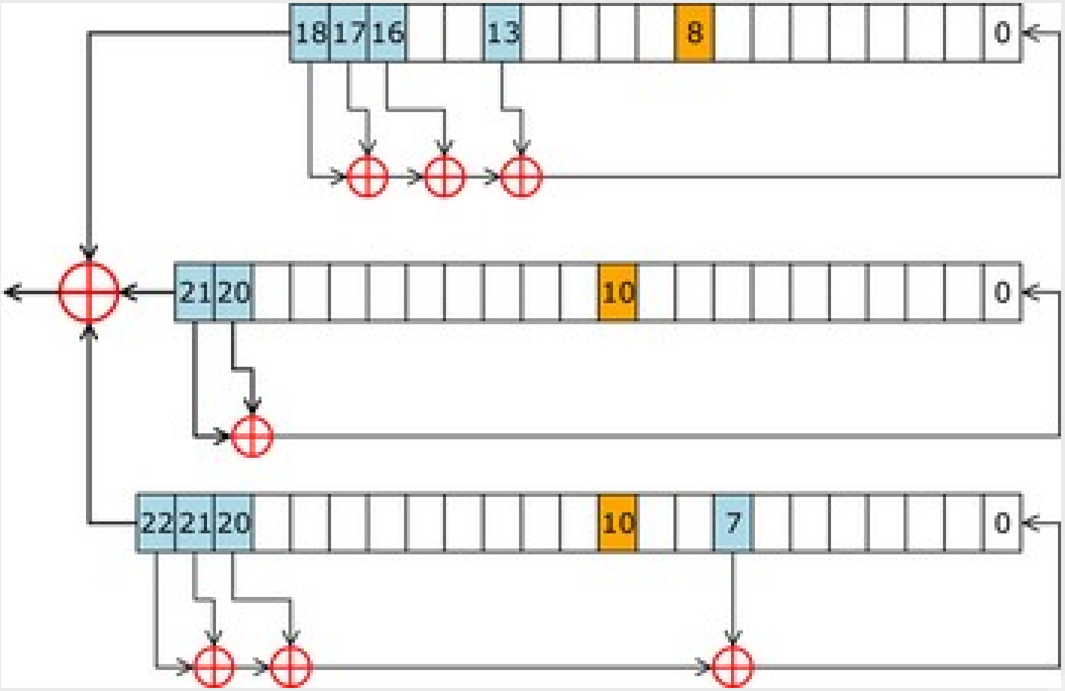
Real-time Attack

- Third, A5/1 is run for 100 clock cycles



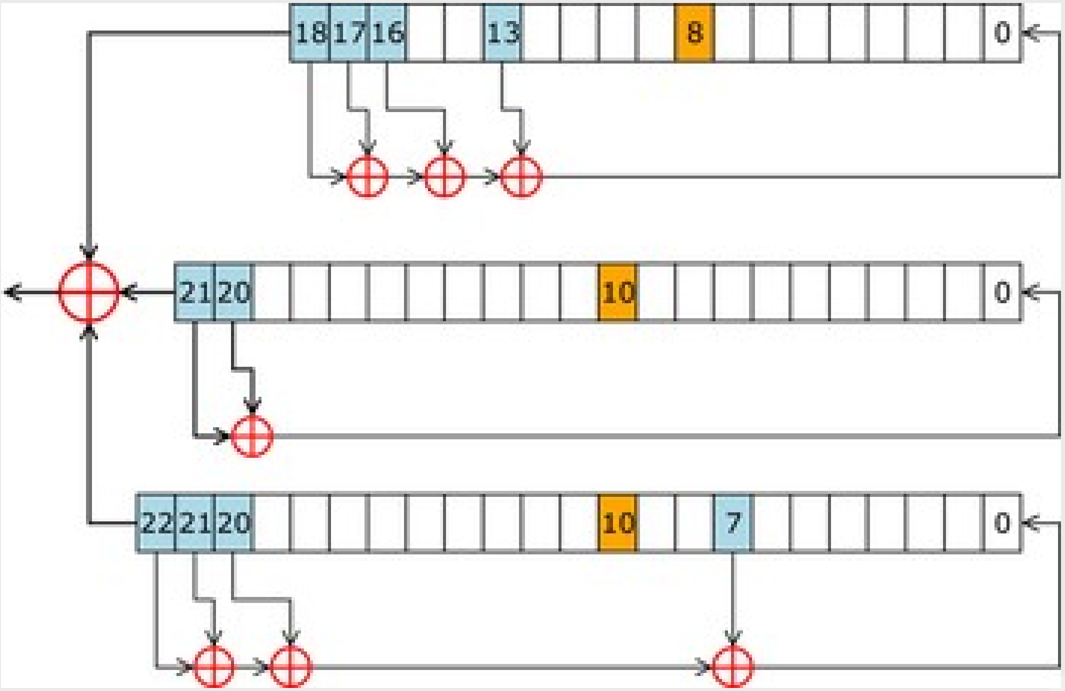
Real-time Attack

- And then the output is xor'ed with the plaintext



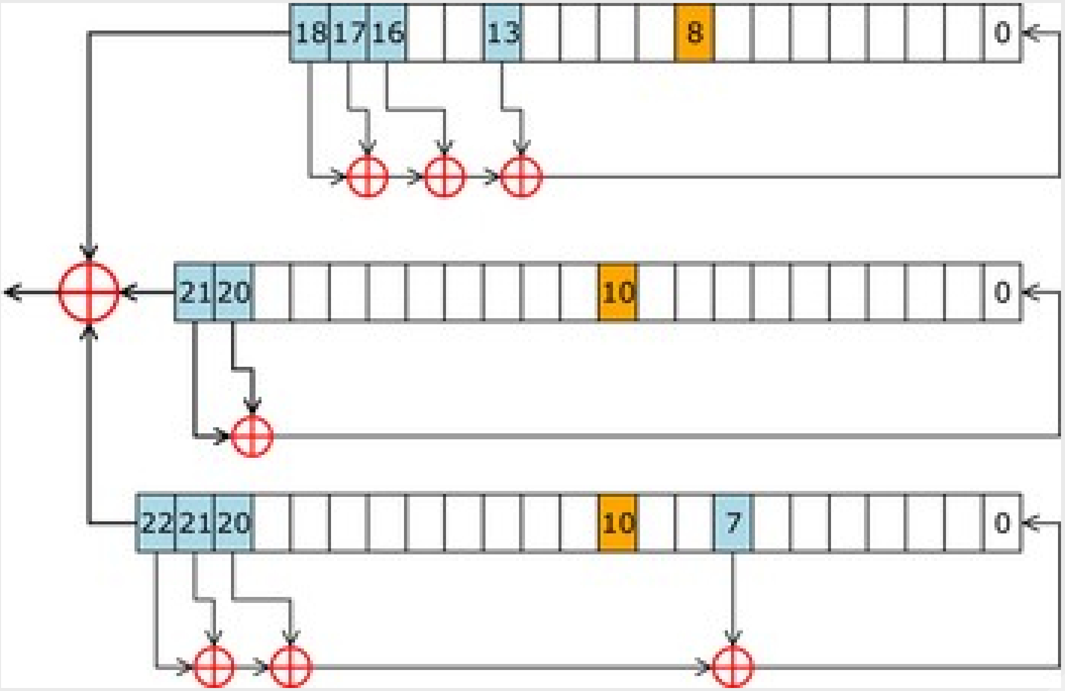
Real-time Attack

- Using known-plaintext you can reduce the keyspace by brute forcing R1/R2 and calculating a matching R3 using the plaintext as parity



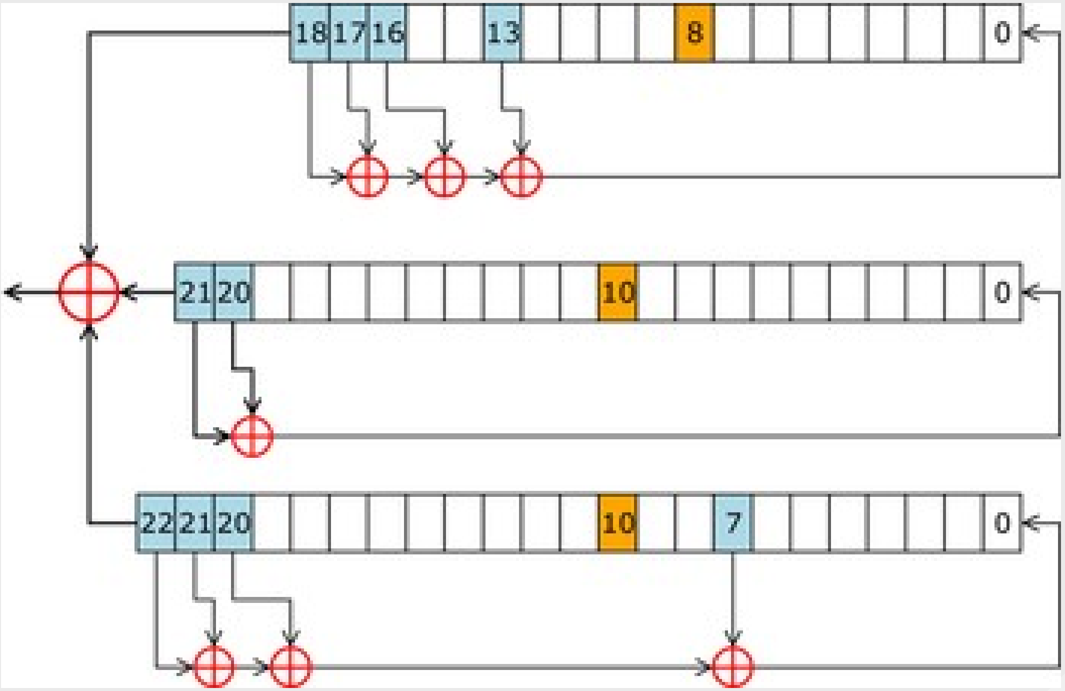
Real-time Attack

- Output bits are related to register bits



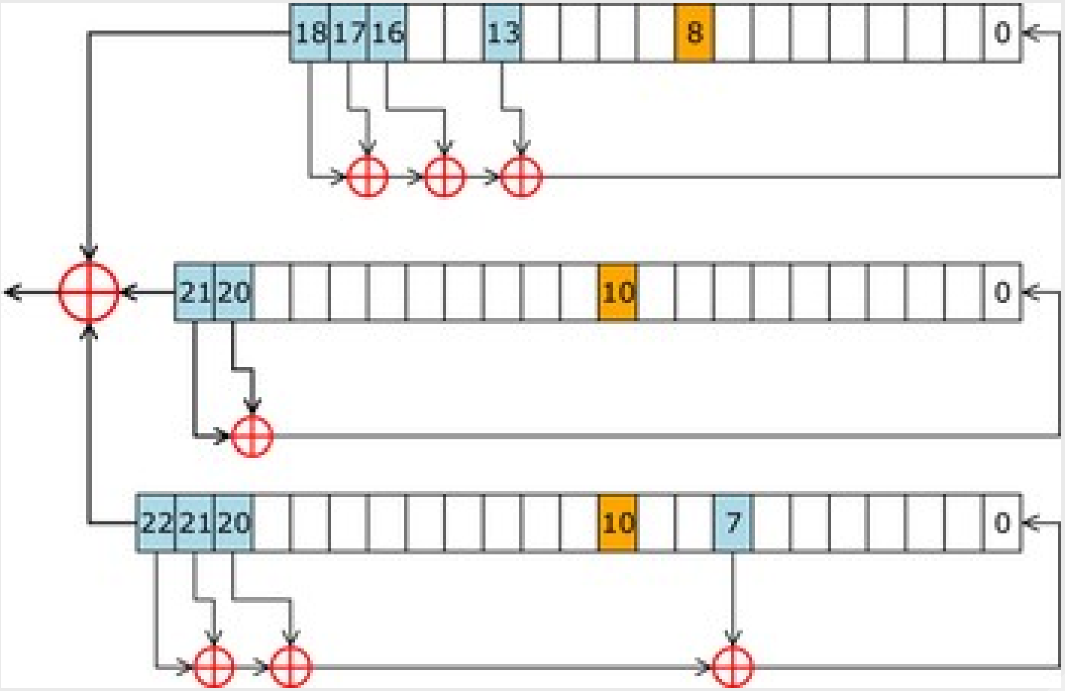
Real-time Attack

- Must essentially brute force the clock bits



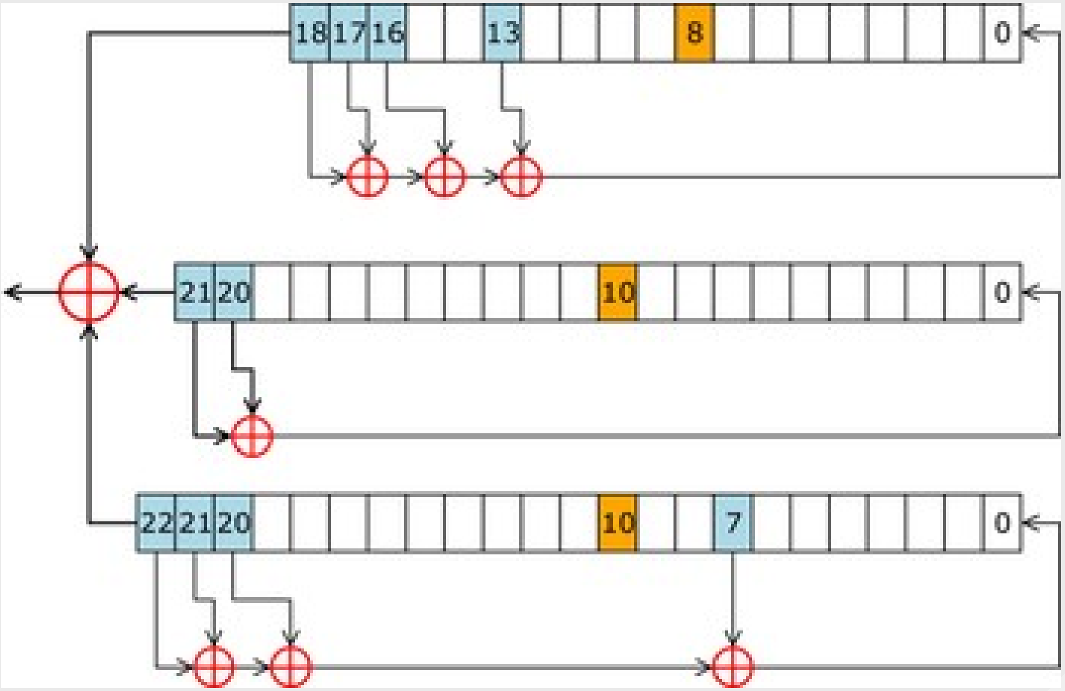
Real-time Attack

- Certain clock bit possibilities can be initially ruled out by looking at registers that don't get clocked and output doesn't match



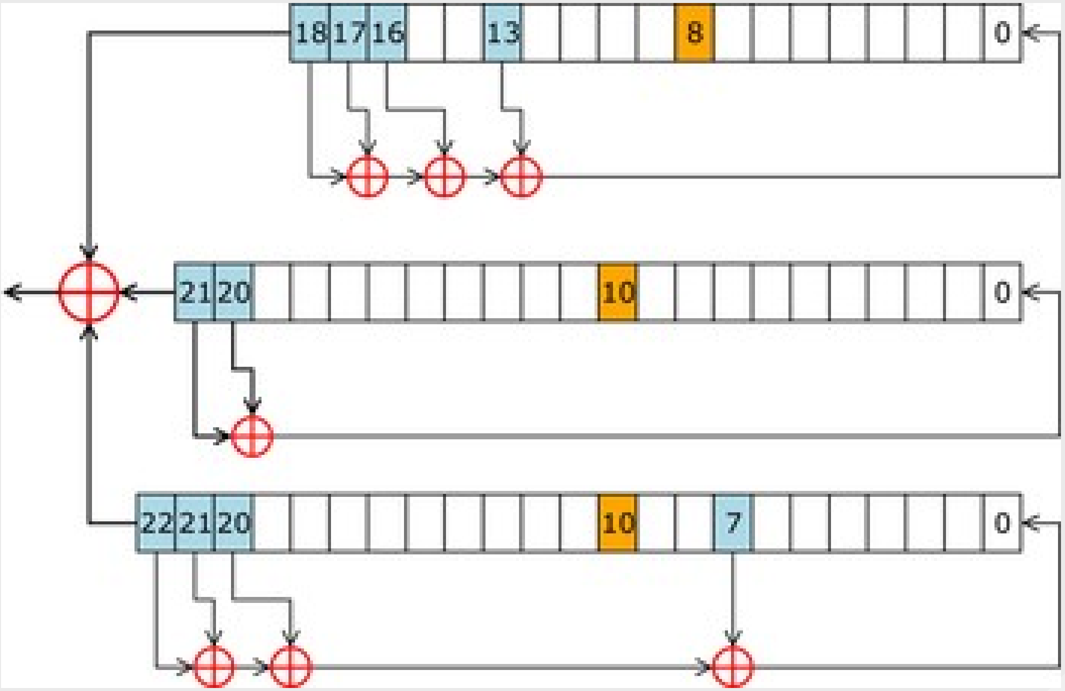
Real-time Attack

- After a while certain possibilities can be ruled out by clock bits not matching output



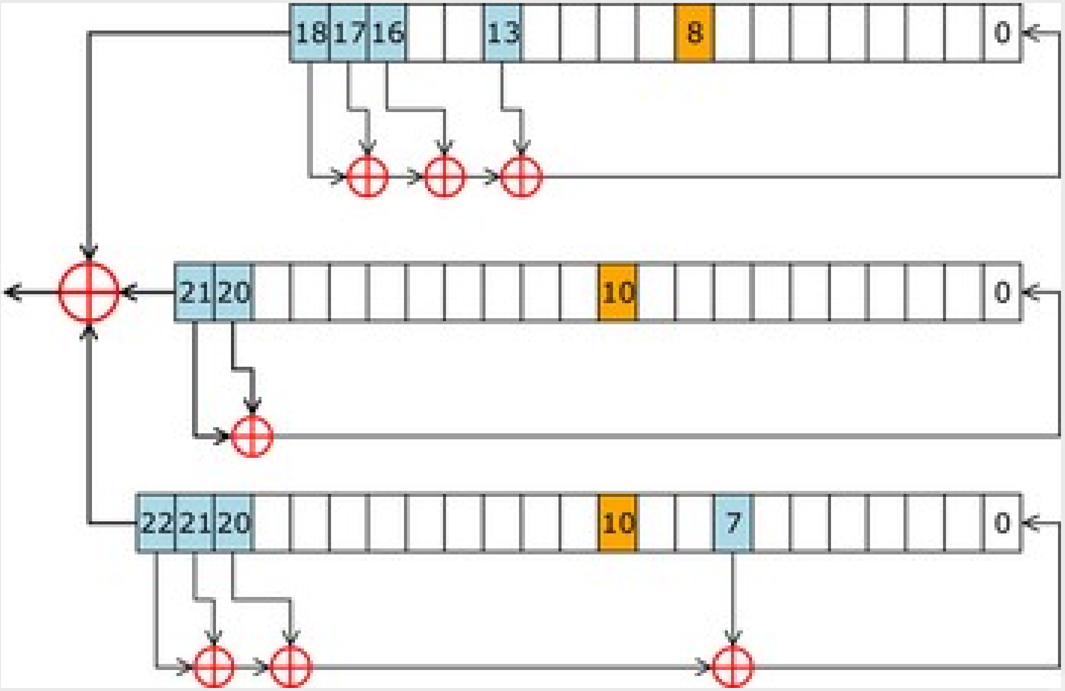
Real-time Attack

- And possibilities can be ruled out once the tap bits are computed and propagate up the registers



Real-time Attack

- With enough known-plaintext you will be able to resolve all possible R1/R2 down to a valid R3

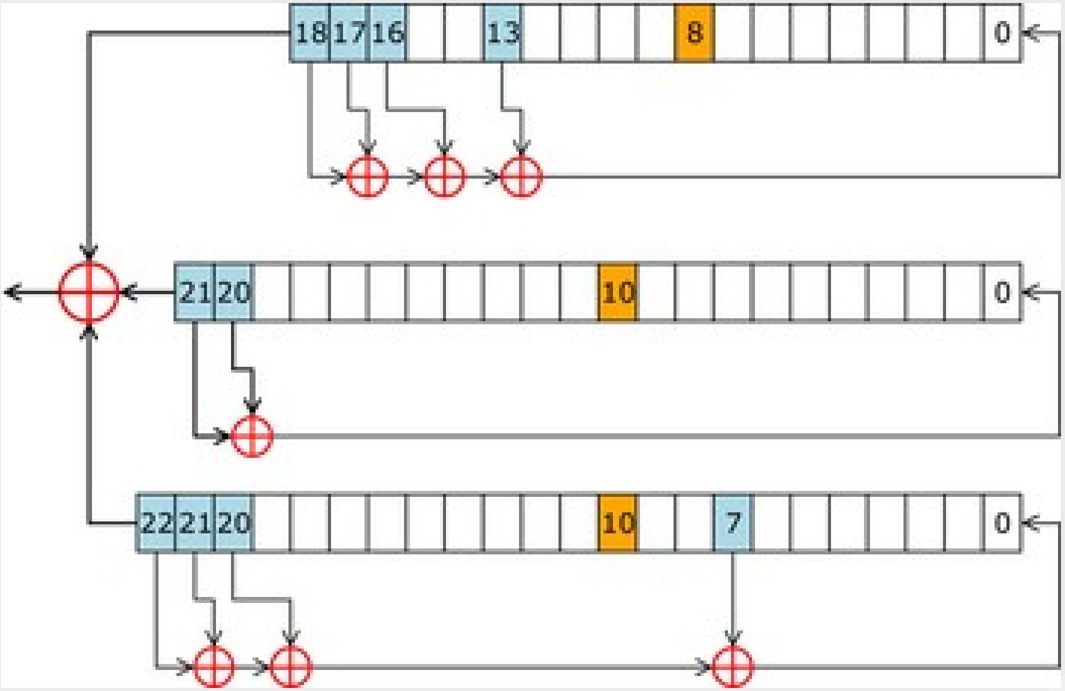


Results

- FPGA code requires around 6000 clock cycles for each R1/R2 state
- 100 cores at 100MHz will do 1.6M per sec
- One FPGA will crack key in 15 days
- 100 FPGAs will crack in 3.6 hours
- PCs will take a really long time (~2,000 times slower)
- (code is free and available if you want to offload it to your botnet ;-)

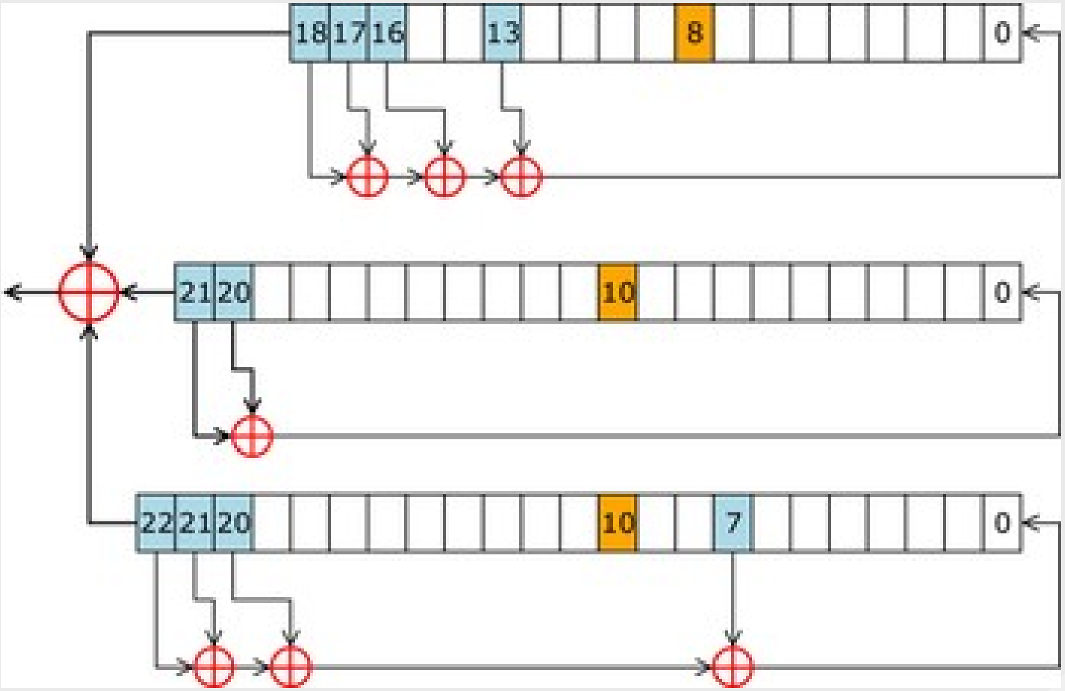
Real-time Attack

- Once the internal state of A5/1 is derived you can reverse clock A5/1 back to the state after the key is mixed in



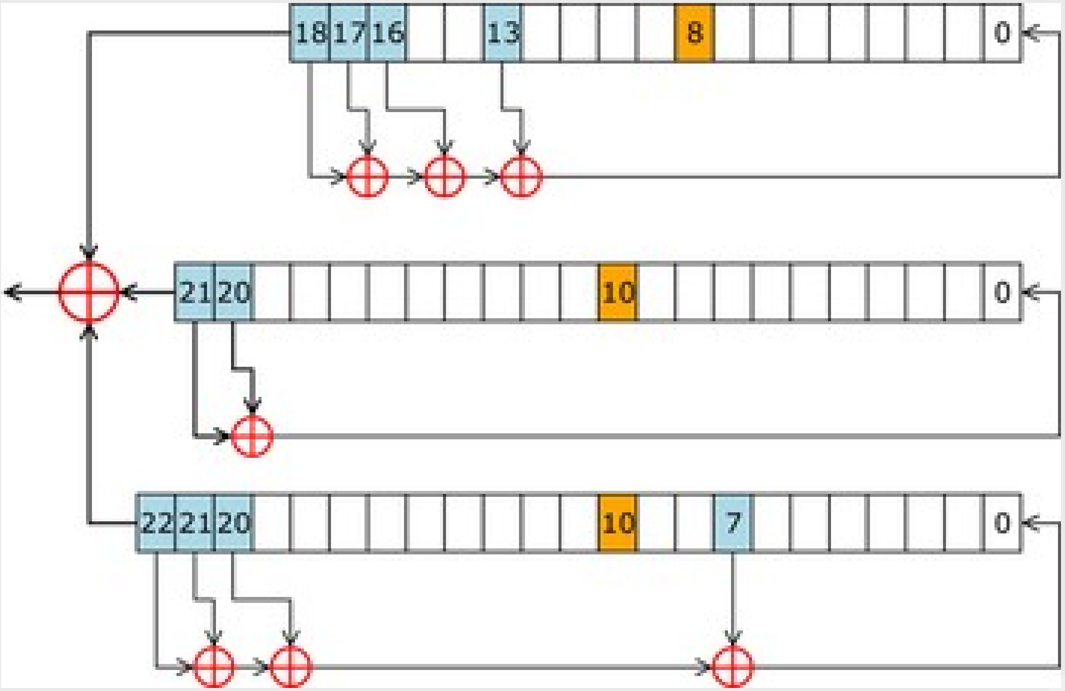
Real-time Attack

- To reverse A5/1 you calculate the only states for the clocking bits that are possible



Real-time Attack

- Eventually there are only a few possibilities left



Real-time attack

- This can be done quickly in software
  - Because there are multiple possible states you need multiple packets to locate the correct state
  - 2 packets is enough, we happen to have 4
  - Calculate the possible initial states for 2 packets
  - Find the possible state that's common between the two

Decrypting Packets

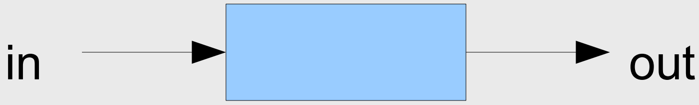
- If you have the A5/1 state after the key is mixed in
- It is the same as having the key
- You can mix in any frame number and generate the proper A5/1 output to xor to any ciphertext or plaintext

Pre-computation Attack

- Reversing 64-bits of A5/1 output to 64-bits of A5/1 internal state
  - Essentially the same concept as a one-way function
  - Rainbowtables are good at reversing one-way functions
  - Decided to focus on building a table of 2^58
  - 1/64 chance of finding the key with a given packet
  - We have 200 different 64-bit A5/1 outputs
  - Good chance that we'll find the key

Pre-computation Attack

- Time-space tradeoff basics
  - You have a one-way function

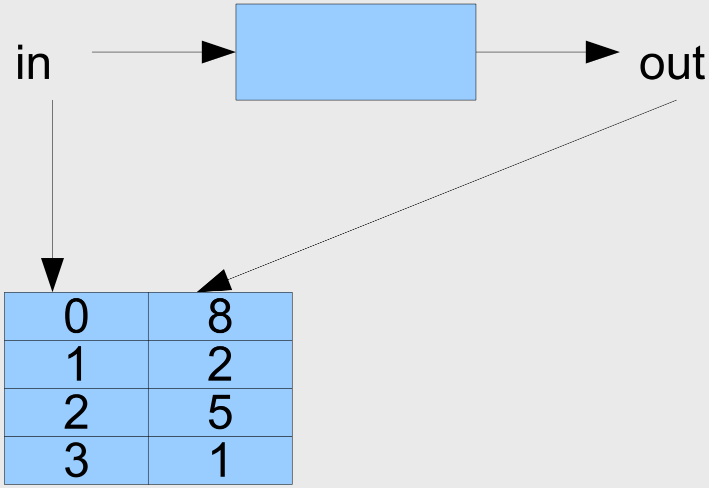


Pre-computation Attack

- Time-space tradeoff basics
  - You have a one-way function
  - You need to find the in that created an out

Pre-computation Attack

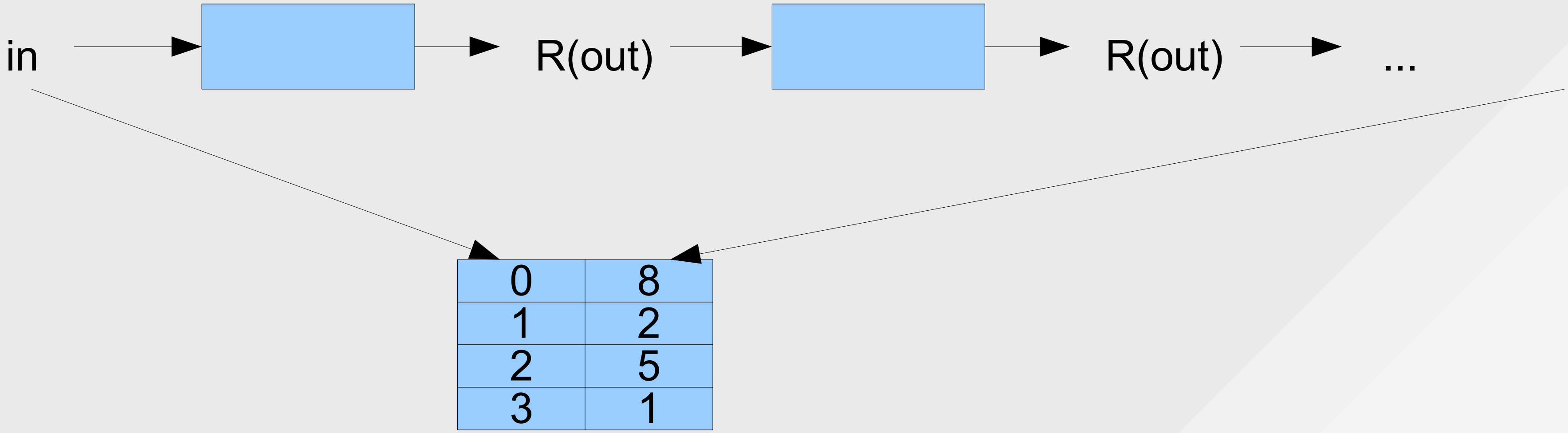
- Naïve implementation
  - Pre-computation
    - Compute and store all possible in/out's
  - Real-time
    - Search through table until you find your out, it's in is the key



0	8
1	2
2	5
3	1

Pre-computation Attack

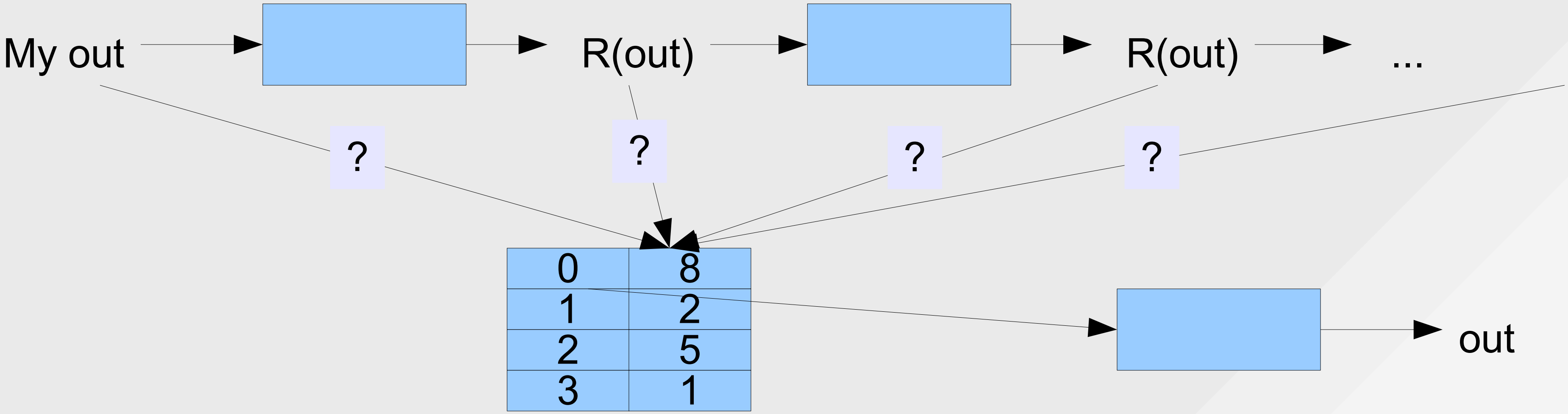
- Basic time-space tradeoff implementation
  - Pre-computation
    - Compute an in -> out, and then take the out and compute another out, etc. (using a “reduction function”)
    - Just store the start and end values of the chain



0	8
1	2
2	5
3	1

Pre-computation Attack

- Basic time-space tradeoff implementation
  - Real-time
    - To reverse a hash, you compute a chain for your out value and compare all out values with all of the end points in your table
    - When you find a matching one, compute a chain from it's start value
    - Your in will be right before your out in its chain



0	8
1	2
2	5
3	1

Pre-computation Attack

- Problems with time-space tradeoff
  - Algorithms have collisions (especially when you're mapping output to input which have differing entropy)
  - Collisions cause chains to merge or loop
  - Different algorithms are used to mitigate this

Pre-computation Attack

- Different time-space tradeoff algorithms
  - Basic
    - Use a different reduction function for different tables
    - Increase the amount of tables you have depending on how collision prone your algorithm is
  - Distinguished Points
    - Instead of having all of your chains be the same length you stop when you see a certain pattern of bits
    - You can detect collisions and reject chains by looking for other chains that end in the same distinguished point
  - Rainbow
    - Use a different reduction function for each stage of the chain
    - Requires more real-time computation

Pre-computation Attack

- Distinguished Points
  - We first assumed that it wasn't very collision prone
  - Turned out we were wrong
  - Very efficient on FPGAs because it requires really low bandwidth and table lookups
  - Can be used to speed up the real-time lookup phase
  - Requires lots of tweaking to find the right parameters to provide the least number of collisions and loops

Pre-computation Attack

- Rainbow tables
  - Provides the best collision resistance
  - Requires a lot of real-time computation

    n(n+1)/2 * chain_length
  - The real-time computation can be done on an FPGA (not as well as DP attack)
  - Best attack parameters we could come up with
    - Requires 5TB of disk storage
    - Can reverse a key in 5 min with 1 FPGA
    - Multiple FPGAs can be used to parallelize cracking multiple keys
  - You can adjust table to use less storage and require more time

OpenCiphers

Pre-computation Attack

- Final Analysis
  - Talked to Elad Barkan
  - Distinguished Points was the best solution
  - Had to reduce chain length to provide the best collision/loop resistance (average length of 2^19)
  - Used a different reduction function for each table
  - Provides around 40% coverage (rainbow table provided only ~20%)
  - Still deciding on the best parameters for the tables

Pre-computation Attack

- Limiting factor right now is pre-computation
  - Computing 2^58 requires ~ 100 FPGAs running for 2 months (6,000 times slower on PCs)
  - We have a cluster of 70 FPGAs ready to start computing
  - Once the parameters are finalized we'll be able to compute a full table in ~3 months
  - Will be the largest rainbowtable ever built
    - Typical Lanman rainbowtables cover 2^36 at most
    - This will be 4 million times larger
  - Resulting table will be 2TB

Real-time Attack

- Result
  - Because of hard drive access time it will need to be spread across multiple hard drives
  - 6 hard drives and 1 FPGA will crack a conversation in ~30 min
  - Double the hard drives and FPGAs to halve the time
  - 32 FPGAs and a network of 200 PCs will crack a conversation in < 1 minute

FPGA Implementation

- Implementation
  - Implemented A5/1 as a 64-stage pipeline
  - Much more efficient than state-machines
  - Get 1 A5/1 per clock cycle
  - Output is looped back into the input (after reduction function is applied) until the last 19 bits are 0
  - Results are written to BlockRAM which is polled by the PC
  - Each core runs at 200MHz and 5 cores fit on an LX50
  - Total A5/1's/sec is 200,000,000 * 5 = 1,000,000,000
  - Single CPU does around 150,000/sec

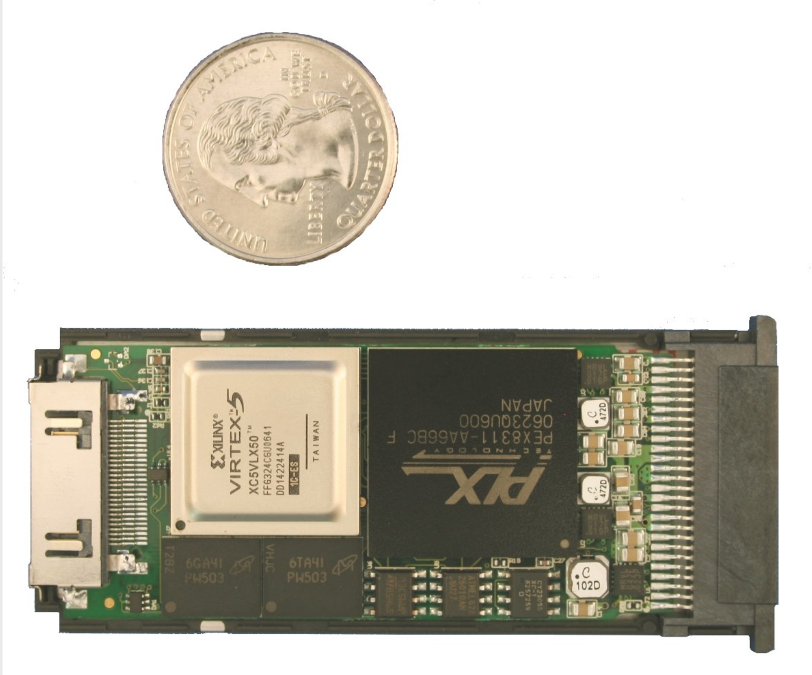
Demo

The A5 Cracking Project

- Currently building 2TB table
- Will eventually build the 28TB table
- If you want to help, check out wiki.thc.org/gsm

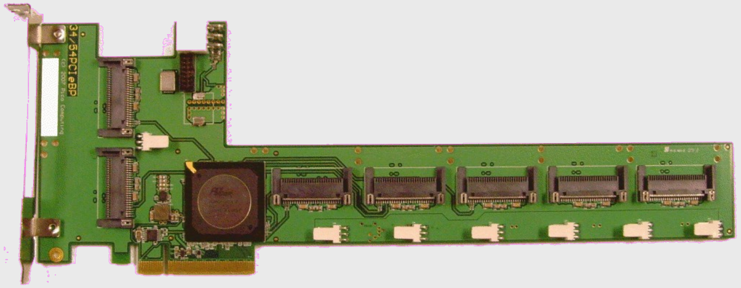
Hardware

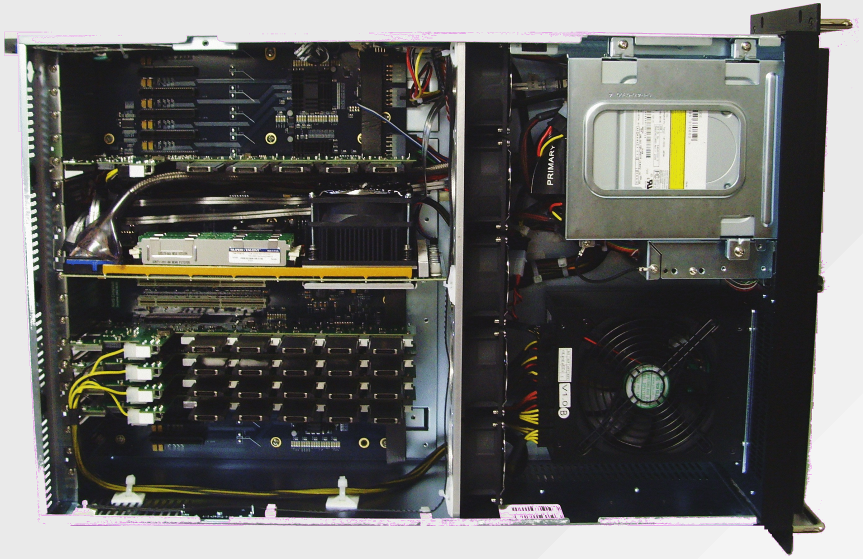
- Pico E-16
  - ExpressCard 34
    - 2.5Gbps full-duplex
  - Virtex-5 LX50
  - 32MB SRAM
  - External ExpressCard Chip



Hardware

- E-16 SuperCluster
  - Up to 77 E-16's
  - 2 Quad-core Xeon's
  - 8GB of RAM
  - 6TB HDD Space





Equivalent computing power of ~400,000 CPUs for cracking A5/1

Conclusion

- Get an FPGA and start cracking!
- Make use if your hardware to break crypto
- <64-bit just doesn't cut it anymore
- Choose bad passwords (please!)

Thanks

- Aaron Peterson (wicrawl)
- The Church of WiFi (CoWPAtty)
- Jacob Appelbaum & Ralf-Philip Weinmann (FileVault)
- Thierry Zoller & Eric Sesterhenn (BTCrack)
- Steve, Josh, & The Hacker's Choice Cr3w (A5 Cracking Project)
- The Shmoo Group (bittorrent seeding)
- Viewers like you

OpenCiphers

Questions?

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