Ανεύρεση Κωδικού με Χρήση Μαρκοβιανών Αλυσίδων

Ασφάλεια Συστημάτων Υπολογιστών
Γεώργιος Δροσάτος

Αναστασιάδου Μαγδαλινή 56678
email: magdanas@ee.duth.gr

Γκαντίδης Χρήστος 56483
email: chrigkan@ee.duth.gr

Δεκέμβριος 2017

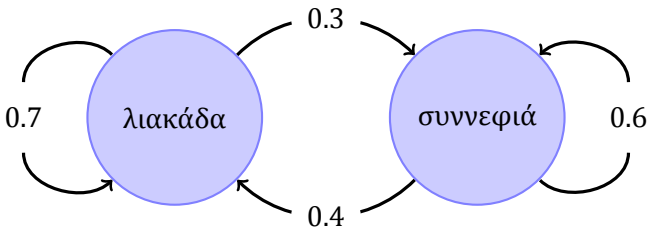
Περιεχόμενα

1 Περίληψη Μαρκοβιανών Αλυσίδων

Οι Μαρκοβιανές αλυσίδες, οι οποίες παίρνουν το όνομά τους από τον Andrey Markov (1856-1922), είναι μαθηματικές κατασκευές οι οποίες περιγράφουν την μετάβαση ενός συστήματος από μια κατάσταση σε όλες τις υπόλοιπες δυνατές καταστάσεις του συστήματος [Wik17f].

Για παράδειγμα, ας θεωρήσουμε τον καιρό μιας περιοχής ως ένα σύστημα, με το σύστημα αυτό να έχεις τις δυνατές καταστάσεις {*λιακάδα*, *συννεφιά*, *βροχή*, *χιόνι*, *χαλάζι*}, στις οποίες αναφερόμαστε ως *πεδίο καταστάσεων*.

Επιπλέον, μπορούμε να ορίσουμε μια Μαρκοβιανή αλυσίδα, η οποία περιγράφει την πιθανότητα μετάβασης από την μια κατάσταση στην άλλη, για παράδειγμα από την κατάσταση *λιακάδα* στην κατάσταση *συννεφιά*. Ο πιο εύκολος τρόπος να αναπαραστήσουμε γραφικά μια Μαρκοβιανή αλυσίδα είναι σε μορφή γράφου όπως φαίνεται στο Σχήμα 1.



Σχήμα 1: Αναπαράσταση Μαρκοβιανής Αλυσίδας με Μορφή Γράφου.

Για δύο δυνατές καταστάσεις, υπάρχουν τέσσερις δυνατές μεταβάσεις, καθώς από κάθε κατάσταση υπάρχει η πιθανότητα να μεταβούμε στην άλλη ή να παραμείνουμε στην ίδια. Γενικά για κάθε n καταστάσεις υπάρχουν n^2 δυνατές μεταβάσεις.

Επειδή για μεγαλύτερα n οι αναπαραστάσεις με μορφή γράφου αρχίζουν να γίνονται χαοτικές, προτιμούμε την αναπαράσταση των Μαρκοβιανών αλυσίδων ως έναν τετραγωνικό πίνακα A, με το $A[i, j]$ να αναπαριστά την πιθανότητα μετάβασης από την κατάσταση i στην j, όπως φαίνεται στον Πίνακα 1 [Pow14].

	λιακάδα	συννεφιά
λιακάδα	0.7	0.3
συννεφιά	0.4	0.6

Πίνακας 1: Αναπαράσταση Μαρκοβιανής Αλυσίδας με Μορφή Πίνακα.

Παρατηρούμε, και εξάγουμε εύκολα το συμπέρασμα, πως το άθροισμα των στοιχείων μιας γραμμής πρέπει να είναι 1, όπως και η συνολική πιθανότητα μετάβασης.

Οι Μαρκοβιανές αλυσίδες είναι πολύ χρήσιμες για συστήματα τα οποία δεν μεταβαίνουν με την ίδια πιθανότητα σε όλες τις καταστάσεις, αλλά δείχνουν μια προτίμηση προς κάποιες από αυτές. Για το παραπάνω παράδειγμα, όταν μια μέρα έχει *λιακάδα*, τότε η πιθανότητα να έχει και την επόμενη μέρα *λιακάδα* είναι μεγαλύτερη από την πιθανότητα να έχει *συννεφιά*. Ετσι η πιθανότητα μετάβασης από *λιακάδα* σε *λιακάδα* είναι μεγαλύτερη από την πιθανότητα μετάβασης από *λιακάδα* σε *συννεφιά*.

2 Κατακερματισμός Κωδικών

2.1 Συναρτήσεις Κατακερματισμού

Οι *Συναρτήσεις Κατακερματισμού (hash functions)*, είναι συναρτήσεις οι οποίες δέχονται ως είσοδο δεδομένα οποιουδήποτε μεγέθους, και τα αντιστοιχίζουν ντετερμινιστικά σε κάποια άλλα δεδομένα σταθερού μεγέθους. Δεν πρόκειται για μεθόδους κρυπτογράφησης, καθώς δεν υπάρχει καθορισμένος τρόπος να μεταβούμε από τα δεδομένα εξόδου της συνάρτησης στα δεδομένα εισόδου. Ετσι, χρησιμοποιούνται για την επιβεβαίωση της ακεραιότητας των δεδομένων μετά από την μετάδοσή τους, ή για την επαλήθευση γνώσης των δεδομένων εισόδου [Wik17e].

Μια από τις κύριες χρήσεις των Συναρτήσεων Κατακερματισμού, είναι η αποθήκευση κωδικών των χρηστών μιας σελίδας, σε μια βάση δεδομένων, με στόχο την προστασία των προσωπικών δεδομένων των χρηστών. Ετσι, σε περίπτωση που η βάση δεδομένων διαρρεύσει και αποκτήσουν πρόσβαση σε αυτήν τρίτοι, οι κωδικοί δεν βρίσκονται σε μορφή *απλού κειμένου (plain text)*, και οι τρίτοι δεν μπορούν να χρησιμοποιήσουν τους κωδικούς για να αποκτήσουν πρόσβαση σε λογαριασμούς των χρηστών σε άλλες σελίδες [Sec17].

Μια από τις πιο γνωστές και ευρέως χρησιμοποιούμενη Συνάρτηση Κατακερματισμού, η οποία όμως δεν θεωρείται πλέον κρυπτογραφικά ασφαλής, είναι η MD5 [Wik17g]. Δέχεται ως είσοδο ένα μπλοκ δεδομένων, που στην δική μας περίπτωση είναι ο κωδικός απλού κειμένου, και παράγει στην έξοδό της μια τιμή $128\,\text{bits}$, δηλαδή $16\,\text{Bytes}$, η οποία αναπαριστάται συνήθως από 32 εκαεξαδικούς χαρακτήρες. Η έξοδος της MD5 αλλάζει δραματικά, ακόμα και με μια μικρή αλλαγή στην είσοδο της, όπως φαίνεται στον Πίνακα 2.

κωδικός	MD5 hash
`apple`	`1f3870be274f6c49b3e31a0c6728957f`
`Apple`	`9f6290f4436e5a2351f12e03b6433c3c`
`apple␣`	`e307c4bc0265c934316dbd59f86336fd`

Πίνακας 2: Μια μικρή μεταβολή της εισόδου, προκαλεί μεγάλη μεταβολή της εξόδου.

Λόγω του ότι οι Συναρτήσεις Κατακερματισμού δεν μπορούν να αντιστραφούν, δεν μπορούμε δηλαδή να παράγουμε την είσοδο από την έξοδο, οι μόνοι τρόποι να βρούμε ποιος κωδικός απλού κειμένου αντιστοιχεί στον κωδικό κατακερματισμού, είναι είτε να γνωρίζουμε εξαρχής τον κωδικό απλού κειμένου (επαλήθευση γνώσης των δεδομένων εισόδου), είτε να δοκιμάσουμε όλους του δυνατούς κωδικούς απλού κειμένου ως εισόδους της Συνάρτησης Κατακερματισμού μέχρι να βρούμε την έξοδο που είναι ίδια με τον κατακερματισμένο κωδικό, οπότε και έχουμε βρει τον ζητούμενο κωδικό απλού κειμένου. Η διαδικασία αυτή λέγεται *Ανεύρεση Κωδικού (Password Guessing/Cracking)* [Wik17h] και θα περιγραφεί παρακάτω.

2.2 Ανεύρεση Κωδικού Απλού Κειμένου από Κατακερματισμένο Κωδικό

Οι σελίδες που αποθηκεύουν τους κωδικούς των χρηστών τους σε κατακερματισμένη μορφή, επαληθεύουν πως ο χρήστης που προσπαθεί να εισέλθει είναι αυτός που δημιούργησε τον λογαριασμό, περνώντας τον κωδικό απλού κειμένου που εισήγαγε στην σελίδα εισόδου από την συνάρτηση κατακερματισμού, και ελέγχοντας αν το αποτέλεσμα είναι ίδιο με την τιμή που υπάρχει αποθηκευμένη στην βάση. Σε περίπτωση που είναι ίδιο, τότε ο χρήστης είχε γνώση των αρχικών δεδομένων, δηλαδή του κωδικού απλού κειμένου που όρισε όταν δημιούργησε τον λογαριασμό του στην σελίδα, και έτσι του επιτρέπεται η πρόσβαση.

Παρακάτω, παρουσιάζουμε μια μέθοδο για την ανεύρεση του κωδικού απλού κειμένου, όταν δεν έχουμε δηλαδή από πριν γνώση των αρχικών δεδομένων, αλλά μόνο του κατακερματισμένου κωδικού. Αν και παρόμοιες μέθοδοι μπορούν να χρησιμοποιηθούν από κακόβουλους χρήστες, με σκοπό για παράδειγμα την πρόσβαση σε κάποιον άλλο λογαριασμό του χρήστη με τον ίδιο κωδικό απλού κειμένου, ο λόγος που παρουσιάζουμε την μέθοδο εμείς είναι ερευνητικός, και με απώτερο σκοπό να ευαισθητοποιήσουμε τους χρήστες ώστε να επιλέγουν κατάλληλους κωδικούς, οι οποίοι δεν είναι ευάλωτοι σε αυτού του είδους τις επιθέσεις.

3 Μέθοδοι Ανεύρεσης

3.1 Μέθοδος Ωμής Βίας (Brute Force)

Ο πιο απλός τρόπος για την ανεύρεση ενός κωδικού, είναι η επίθεση ωμής βίας [Wik17a], κατά την οποία δοκιμάζουμε όλους τους πιθανούς συνδυασμούς των χαρακτήρων που ανήκουν στο *πεδίο χαρακτήρων (character space)*, συνήθως κατά αύξουσα σειρά.

Αν υποθέσουμε πως το πεδίο χαρακτήρων αποτελείται από όλους τους πεζούς χαρακτήρες (a–z), και θέλουμε να δοκιμάσουμε όλους τους κωδικούς με μήκος τρεις χαρακτήρες, τότε η μέθοδος ωμής βίας θα δοκιμάσει τους κωδικούς με την σειρά που φαίνεται στον Πίνακα 3.

aaa	aba	⋯	baa	bba	⋯	zza
aab	abb	⋯	bab	bbb	⋯	zzb
aac	abc	⋯	bac	bbc	⋯	zzc
⋮	⋮	⋱	⋮	⋮	⋱	⋮
aaz	abz	⋯	baz	bbz	⋯	zzz

Πίνακας 3: Παράδειγμα εκτέλεση μεθόδου ωμής βίας.

Στην πραγματικότητα, και στο παρόν έργο, το πεδίο χαρακτήρων αποτελείται συνήθως από όλους τους εκτυπώσιμους χαρακτήρες του κώδικα ASCII (κωδικοί 32–126) [Wik17c]:

lowercase a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

uppercase A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

punctuation ␣(space), !, ”, #, $, %, &, ’, (,), *, +, „ -, ., /, :, ;, <, =, >, ?, @, [, \,], ^, _, ‘, {, |, }, ~

Σε πιο επαγγελματικές βάσεις δεδομένων και για σελίδες με διεθνές κοινό (που έχουν χρήστες για παράδειγμα και στην Ασία), μπορεί να χρησιμοποιηθούν όλοι οι εκτυπώσιμοι χαρακτήρες του κώδικα Unicode [Wik17d] κάτι που ξεφύγει όμως από το πεδίου αυτού του έργου.

Οπως ίσως είναι εύκολο να αντιληφθεί κανείς, αν το μήκος του κωδικού απλού κειμένου είναι n χαρακτήρες, και το πεδίο χαρακτήρων αποτελείται από s χαρακτήρες, θα πρέπει να δοκιμάσουμε s^n διαφορετικούς κωδικούς. Η μέθοδος ωμής βίας δηλαδή έχει υπολογιστική πολυπλοκότητα $\mathcal{O}(s^n)$.

3.2 Μέθοδος Λεξικού (Dictionary)

Η μέθοδος λεξικού (dictionary attack) βασίζεται στην δοκιμή λέξεων από μια υπάρχουσα λίστα. Σε αντίθεση με την μέθοδο ωμής βίας, η μέθοδος λεξικού περιορίζεται μόνο σε συνηθισμένες ακολουθίες χαρακτήρων που είναι εύκολο να απομνημονευθούν. Η επιτυχία της οφείλεται στο γεγονός ότι πολλοί χρήστες επιλέγουν μικρούς σε μέγεθος κωδικούς, οι οποίοι είναι είτε μια καθημερινή λέξη είτε απλή παραλλαγή μιας λέξης, όπως για παράδειγμα την μετατροπή κάποιων χαρακτήρων από πεζούς σε κεφαλαίους, την προσθήκη κάποιου αριθμού στο τέλος της λέξης, την αντικατάσταση κάποιων χαρακτήρων με κάποια σύμβολα (l33t-speak) ή κάποιον συνδυασμό των παραπάνω [Wik17b].

Για παράδειγμα, ένας κωδικός πεζών χαρακτήρων, με μήκος οκτώ, με την μέθοδο ωμής βίας θα χρειαζόταν στην χειρότερη περίπτωση $26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 = 26^8$ προσπάθειες για να εντοπιστεί. Με έναν ρυθμό δοκιμής $5\,000\,000$ ωδικών το δευτερόλεπτο (ρυθμός που πετυχαίνεται εύκολα από σύγχρονο υλικό), θα απαιτούνταν περίπου $11.6\,\mathrm{h}$.

Από την άλλη μεριά, ο αριθμός των λέξεων στο Oxford English Dictionary είναι περίπου $600\,000$, θεωρώντας ότι ο μέσος άνθρωπος χρησιμοποιεί ένα μόνο μέρος αυτών των λέξεων, από $10\,000$ ως $40\,000$, θα απαιτούνταν γύρω στα $8\,\mathrm{ms}$ για να δοκιμαστούν όλες οι λέξεις!

Ακόμη, είναι σχετικά εύκολο ο επιτιθέμενος να δημιουργήσει και να αποθηκεύσει μια λίστα με κατακερματισμένους κωδικούς από όλες τις λέξεις σε ένα λεξικό. Εάν γνωρίζει τον κατακερματισμένο κωδικό κάποιου τότε μπορεί σχεδόν ακαριαία να τον αναζητήσει και να τον εντοπίσει στην λίστα που δημιούργησε.

Αν το λεξικό περιέχει l λέξεις, τότε η μέθοδος λεξικού θα δοκιμάσει l διαφορετικούς κωδικούς, έχει δηλαδή υπολογιστική πολυπλοκότητα $\mathcal{O}(l)$. Αν και πρόκειται για σημαντική βελτίωση από την εκθετική πολυπλοκότητα της μεθόδου ωμής βίας, είναι αρκετά περιορισμένη από το μήκος του λεξικού, και από το γεγονός πως μπορεί ο κωδικός να μην βρίσκεται μέσα σε κανένα λεξικό.

3.3 Μέθοδος με Χρήση Μαρκοβιανών Αλυσίδων (Markov Chains)

Με την χρήση των Μαρκοβιανών αλυσίδων, μπορούμε να βελτιώσουμε την μέθοδο ωμής βίας [Tur12]. Η όλη λογική βασίζεται στο γεγονός ότι στις λέξεις της φυσική γλώσσας, η συχνότητα και η ακολουθία των γραμμάτων δεν είναι ισοπίθανη.

Ενα παράδειγμα που δείχνει την διαφορά στην συχνότητα των χαρακτήρων στην Αγγλική γλώσσα[1], είναι το ότι υπάρχουν περισσότερες λέξεις που ξεκινάνε με τον χαρακτήρα `'c'` παρά με τον χαρακτήρα `'b'`. Ετσι, είναι λογικό, όταν προσπαθούμε να βρούμε τον κωδικό απλού κειμένου, να δοκιμάσουμε όλους τους κωδικούς που ξεκινάνε με `'c'` πριν αυτούς που ξεκινάνε με `'b'`, και όχι με καθαρά αλφαβητική σειρά, καθώς υπάρχει μεγαλύτερη πιθανότητα να βρούμε τον σωστό κωδικό στην πρώτη περίπτωση, γλιτώνοντας έτσι πολύτιμο χρόνο και υπολογιστικό έργο.

Ενα άλλο παράδειγμα, που δείχνει αυτή τη φορά την διαφορά στην ακολουθία των χαρακτήρων στην Αγγλική γλώσσα[1], είναι το ότι υπάρχουν περισσότερες περιπτώσεις κατά τις οποίες ο χαρακτήρας `'a'` ακολουθείται από τον χαρακτήρα `'b'`, παρά από τον χαρακτήρα `'a'`. Και πάλι, είναι λογικό, όταν ο προηγούμενος χαρακτήρας του κωδικού είναι `'a'`, να δοκιμάσουμε στην τωρινή θέση τον χαρακτήρα `'b'` πριν από τον χαρακτήρα `'a'`, και όχι με καθαρά αλφαβητική σειρά, τακτική που μας επιφέρει και πάλι τα κέρδη που περιγράφηκαν πριν.

Το παραπάνω παράδειγμα αποκτά ακόμα μεγαλύτερη ισχύ, αν σκεφτούμε πως η ακολουθία των χαρακτήρων αλλάζει ανάλογα με την θέση μέσα στην λέξη[1]. Ετσι, στην δεύτερη θέση, είναι πιο πιθανό ο χαρακτήρας `'a'` να ακολουθείται από τον χαρακτήρα `'s'` παρά από τον χαρακτήρα

`'t'`. Οι πιθανότητες αυτές όμως αντιστρέφονται στην τρίτη θέση της λέξης, όπου ο χαρακτήρας `'a'` είναι πιο πιθανό να ακολουθείται από τον χαρακτήρα `'t'` παρά από τον χαρακτήρα `'s'`.

Εκμεταλλευόμαστε τις παραπάνω ιδιαιτερότητες με την χρήση των Μαρκοβιανών αλυσίδων, ώστε να δοκιμάζουμε τους πιθανούς κωδικούς απλού κειμένου κατά φθίνουσα πιθανότητα εμφάνισης των χαρακτήρων στην αντίστοιχη θέση, ανάλογα με τον χαρακτήρα στην θέση που προηγείται.

Ενα ακόμα προτέρημα έναντι της μεθόδου ωμής βίας, είναι ότι μπορούμε να περιορίσουμε την πολυπλοκότητα του αλγορίθμου, ορίζοντας ένα όριο στο πλήθος των διαφορετικών χαρακτήρων που θα δοκιμαστούν σε κάθε θέση. Μπορούμε για παράδειγμα να δοκιμάσουμε μόνο τους 30 πιο πιθανούς χαρακτήρες σε κάθε θέση, έναντι του να δοκιμάσουμε όλους τους 95 δυνατούς χαρακτήρες.

Για παράδειγμα, αν το μήκος του κωδικού απλού κειμένου είναι n χαρακτήρες, και το πεδίο χαρακτήρων έχει πληθικότητα s, μπορούμε να ορίσουμε ένα *όριο (threshold)* $t; t \leq s$, έχοντας έτσι να δοκιμάσουμε t^n αντί για s^n διαφορετικούς κωδικούς. Φαίνεται λοιπόν, πως η υπολογιστική πολυπλοκότητα της μεθόδου Μαρκοβιανών αλυσίδων είναι $\mathcal{O}(t^n)$, με την δυνατότητα να κάνουμε το t όσο μικρό θέλουμε. Σε περίπτωση που $t = s$, τότε η μέθοδος Μαρκοβιανών αλυσίδων εκφυλλίζεται στην μέθοδο ωμής βίας, με το πλεονέκτημα της σειράς με την οποία θα δοκιμαστούν οι κωδικοί όμως να παραμένει.

3.4 Ιριδίζοντες Πίνακες (Rainbow Tables)

Οι ιριδίζοντες πίνακες έρχονται να μετατρέψουν το φορτίο, από υπολογιστικό σε αποθηκευτικό. Αντί να παράγουμε κάθε φορά τους κατακερματισμένους κωδικούς που αντιστοιχούν στους κωδικούς απλού κειμένου, μπορούμε να τους αποθηκεύσουμε μαζί σε ένα αρχείο, και αργότερα να πραγματοποιήσουμε απλά μια αναζήτηση για να εντοπίσουμε τον κωδικό απλού κειμένου που αντιστοιχεί στον κατακερματισμένο κωδικό. Η τακτική αυτή υπάρχει εδώ και πολλά χρόνια, και έχει χρησιμοποιηθεί με επιτυχία [Wik17j], αλλά έχει ένα βασικό μειονέκτημα: περιορίζεται από το μήκος του κωδικού.

Αν υποθέσουμε πως το πεδίο χαρακτήρων αποτελείται από τους 95 εκτυπώσιμους χαρακτήρες του κώδικα ASCII, πως κάθε χαρακτήρας αναπαριστάται από $8\,\text{bits}$, και θυμηθούμε πως το αποτέλεσμα της συνάρτηση κατακερματισμού MD5 είναι μια τιμή $128\,\text{bits}$, τότε στον Πίνακα 4 παρουσιάζονται τα μεγέθη των αρχείων που απαιτούνται για να αποθηκεύσουν τα ζεύγη κατακερματισμένου κωδικού και κωδικού απλού κειμένου. Οπως βλέπουμε το μέγεθος των αρχείων αυξάνεται εκθετικά.

# χαρακτήρων	# πιθανών κωδικών	μέγεθος αρχείου
1	95 ($10^{1.98}$)	1.58 KiB
2	9 025 ($10^{3.96}$)	158.64 KiB
3	857 375 ($10^{5.93}$)	15.54 MiB
4	81 450 625 ($10^{7.91}$)	1.52 GiB
5	7 737 809 375 ($10^{9.89}$)	151.33 GiB
6	735 091 890 625 ($10^{11.87}$)	14.71 TiB
7	69 833 729 609 375 ($10^{13.84}$)	1.43 PiB
8	6 634 204 312 890 625 ($10^{15.82}$)	141.42 PiB
9	630 249 409 724 609 375 ($10^{17.80}$)	13.67 EiB

Πίνακας 4: Μέγεθος αρχείου αποθήκευσης Ιριδίζοντος Πίνακα.

[1] Σύμφωνα με το λεξικό `/usr/share/dict/american-english` που βρίσκεται σε όλα τα λειτουργικά *nix

Η μέθοδος βέβαια καθίσταται σχεδόν αδύνατη σε περιπτώσεις που οι κωδικοί είναι αποθηκευμένοι μαζί με κάποιο μεγάλου μήκους τυχαία ακολουθία χαρακτήρων, που ονομάζεται *salt*. Δηλαδή όταν ο κατακερματισμένος κωδικός που γνωρίζει εκ των προτέρων ο επιτιθέμενος δεν προέκυψε απλά από το αποτέλεσμα της συνάρτησης κατακερματισμού για τον κωδικό του χρήστη, αλλά από το αποτέλεσμα αυτής της συνάρτησης για την συνένωση του κωδικού του χρήστη με το salt.

κωδικός	salt	MD5 hash
`hello`	-	`5d41402abc4b2a76b9719d911017c592`
`hello`	`QxLUF1bgIAdeQX`	`61819a2e9b721831243eba1aeaba8486`
`hello`	`bv5PehSMfV11Cd`	`3c3fbfc82061e60fb6c11a5e0307058e`
`hello`	`YYLmfY6IehjZMQ`	`104f4f46d8222b57cbd543d75a3b8947`

Πίνακας 5: Αποτέλεσμα της συνάρτησης κατακερματισμού MD5 με χρήση salt.

Σε μια τέτοια περίπτωση, χρήστες με τον ίδιο κωδικό δεν θα είχαν τους ίδιους κατακερματισμένους κωδικούς, όπως φαίνεται στον Πίνακα 5. Ο επιτιθέμενος θα έπρεπε τότε να παράγει και να αποθηκεύσει τους ιριδίζοντες πίνακες που προκύπτουν για κάθε salt συνδυασμένο με κάθε δυνατή λέξη που δημιουργείται με n χαρακτήρες. Οι συνδυασμοί συνεπώς πολλαπλασιάζονται, και η αποθήκευση ενός τέτοιου πίνακα είναι πρακτικά αδύνατη.

4 Υλοποίηση Μεθόδου Μαρκοβιανών Αλυσίδων

Ο πηγαίος κώδικας που υλοποιεί την μέθοδο Μαρκοβιανών αλυσίδων όπως παρουσιάζεται στο παρόν έργο, μπορεί να βρεθεί στο GitHub στην διεύθυνση `https://github.com/christosg88/markov_guessing`.

4.1 Παραγωγή Μαρκοβιανών Πινάκων

Αφού είδαμε στις προηγούμενες ενότητες την χρησιμότητα των Μαρκοβιανών αλυσίδων στην ανεύρεση κωδικών και τα προτερήματά τους σε σχέση με άλλες μεθόδους, ας δούμε τώρα πώς υλοποιούνται.

Στο παρόν έργο δημιουργείται ένας Μαρκοβιανός πίνακας για κάθε θέση του κωδικού (per position Markov table), με σκοπό να εκμεταλλευτούμε την συνολική ισχύ που μας προσφέρουν, όπως περιγράφηκε στην Υποενότητα 3.3.

Σημαντικό παράγοντα στην ποιότητα των παραγόμενων πινάκων έχει η λίστα των λέξεων από την οποία εξάγουμε τις συχνότητες και τις ακολουθίες των χαρακτήρων. Αν για παράδειγμα χρησιμοποιούσαμε ένα απλό λεξικό της Αγγλικής γλώσσας, τότε οι Μαρκοβιανοί πίνακες θα ήταν πολύ καλοί στο να παράγουν λέξεις της Αγγλική γλώσσας. Εμείς όμως θέλουμε να χρησιμοποιήσουμε τους πίνακες για να παράγουμε πραγματικούς κωδικούς, που όπως είδαμε στην Υποενότητα 3.2 μπορεί να έχουν στο τέλος της λέξης διάφορους αριθμούς ή χαρακτήρες στίξης, ή άλλες μετατροπές.

Ο κώδικας έχει γραφεί με τέτοιον τρόπο, ώστε οι Μαρκοβιανοί πίνακες να μπορούν να παραχθούν από οποιαδήποτε λίστα κωδικών απλού κειμένου *(wordlist)*. Φυσικά όσο πιο γενική είναι η λίστα και όσους περισσότερους κωδικούς απλού κειμένου περιέχει, τόσο πιο αποδοτικοί θα είναι οι πίνακες. Αφήνεται στην ηθική κρίση του αναγνώστη το πού θα βρει μια τέτοια λίστα και αν θα την χρησιμοποιήσει, καθώς αυτές οι λίστες στην πλειοψηφία τους προέρχονται από διαρρεύσεις βάσεων δεδομένων σελίδων που δεν κράτησαν με επιτυχία ασφαλείς τους κωδικούς των χρηστών τους.

Στο παρόν έργο περιορίσαμε τους κωδικούς σε μήκος μέχρι 10 χαρακτήρες, αλλά αυτό μπορεί εύκολα να αλλάξει με μια μικρή μετατροπή σε μια σταθερά στον πηγαίο κώδικα. Επίσης, θεωρήσαμε ως πεδίο χαρακτήρων το πλήρες διάστημα εκτυπώσιμων χαρακτήρων του πίνακα ASCII (κωδικοί 32–126), με πληθικότητα 95. Αφού από κάθε μια κατάσταση (προηγούμενος χαρακτήρας), μπορούμε να πάμε σε οποιαδήποτε κατάσταση (τρέχων χαρακτήρας), δημιουργούμε έναν πίνακα με διαστάσεις 95×95 για κάθε θέση, δηλαδή 10 στο πλήθος.

Για την πρώτη θέση του κωδικού, καθώς δεν προηγείται από κανέναν χαρακτήρα, χρησιμοποιήσαμε μόνο την συχνότητα των χαρακτήρων που εμφανίζονται στην θέση αυτή, καθώς η λογική της ακολουθίας δεν ορίζεται. Οι συχνότητες αυτές αποθηκεύτηκαν στην πρώτη γραμμή του Μαρκοβιανού πίνακα για την θέση αυτή.

Μια μικρή μετατροπή που εφαρμόσαμε στους πίνακες, είναι πως δεν χρησιμοποιήσαμε την *πιθανότητα* μετάβασης από τον έναν χαρακτήρα στον άλλο, αλλά την *συχνότητα* μετάβασης. Ετσι, στους πίνακες αυτούς, οι γραμμές δεν αθροίζονται στην μονάδα. Με αυτόν τον τρόπο όμως εξοικονομούμε υπολογιστικό κόστος, καθώς δεν χρειάζεται να διαιρέσουμε την εκάστοτε συχνότητα με το πλήθος όλων των κωδικών για να βρούμε την πιθανότητα μετάβασης. Το μόνο που χρειαζόμαστε από τους πίνακες αυτούς είναι η φθίνουσα ταξινόμηση των χαρακτήρων, η οποία είναι η ίδια είτε χρησιμοποιήσουμε τις πιθανότητες ή τις συχνότητες.

Ο τρόπος που αναπαριστώνται αυτοί οι πίνακες στην μνήμη, είναι υπό την μορφή ενός τρισδιάστατου πίνακα (με όνομα markov στο παράδειγμα που ακολουθεί), με μήκος πρώτης διάστασης το μέγιστο μήκος κωδικού (4 στο παράδειγμα), και μήκος δεύτερης και τρίτης διάστασης την πληθικότητα του πεδίου χαρακτήρων (5 στο παράδειγμα). Ετσι, για κάθε μια από τις θέσεις του κωδικού με μέγιστο μήκος, έχουμε έναν τετραγωνικό πίνακα που κρατάει τις συχνότητες εμφάνισης των χαρακτήρων.

Για να παραχθούν οι Μαρκοβιανοί πίνακες, αρχικοποιούμε όλες τις συχνότητες στο 0. Επειτα, διατρέχουμε όλη την λίστα με τους κωδικούς που δίνεται ως είσοδος στο πρόγραμμα. Για κάθε κωδικό, ξεκινάμε από τον πρώτο χαρακτήρα (έστω s_0) και αυξάνουμε κατά 1 την τιμή στην θέση $\mathrm{markov}[0][0][s_0]$. Συνεχίζουμε στον δεύτερο χαρακτήρα (έστω s_1) και αυξάνουμε κατά 1 την τιμή στην θέση $\mathrm{markov}[1][s_0][s_1]$. Συνεχίζουμε για κάθε χαρακτήρα s_i μέχρι τον τελευταίο, αυξάνοντας σε κάθε περίπτωση κατά 1 την τιμή στην θέση $\mathrm{markov}[i][s_{i-1}][s_i]$.

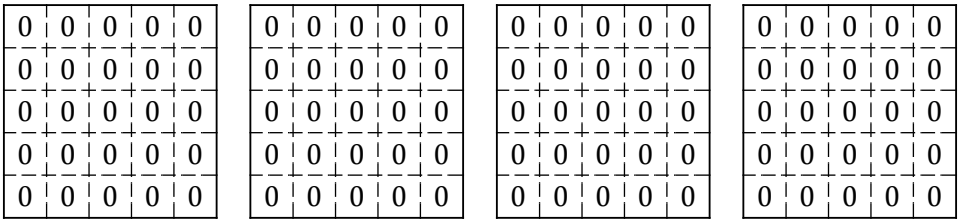
Ας δούμε ένα παράδειγμα παραγωγής πινάκων για να γίνει καλύτερα αντιληπτή η διαδικασία. Θα υποθέσουμε πως το πεδίο χαρακτήρων είναι το {a, b, c, d, e} και η λίστα με τους κωδικούς φαίνεται στον Πίνακα 6, με μέγιστο μήκος κωδικού 4.

```
ace  bee
add  cab
bad  dad
bed  dead
```

Πίνακας 6: Παράδειγμα λίστας με κωδικούς.

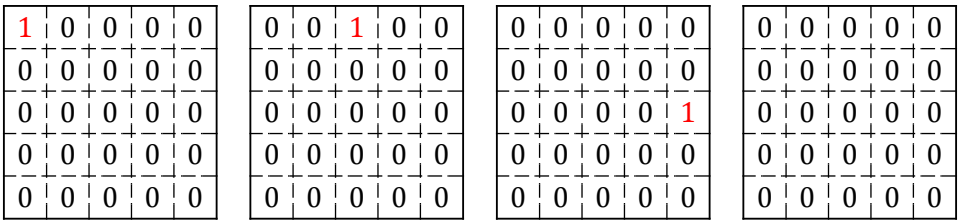
Οι πίνακες που παράγονται, και η παραγωγή τους Βήμα προς Βήμα φαίνονται στον Πίνακα 7.

Βήμα 0: Αρχικοποίηση



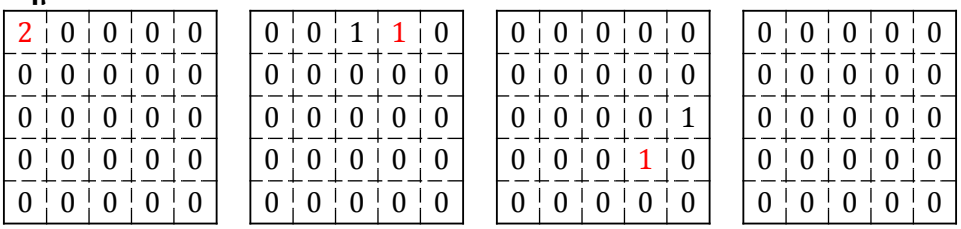
markov[0] markov[1] markov[2] markov[3]

Βήμα 1: ace



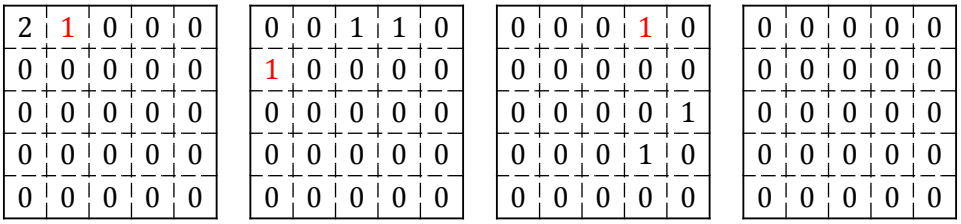
markov[0] markov[1] markov[2] markov[3]

Βήμα 2: add



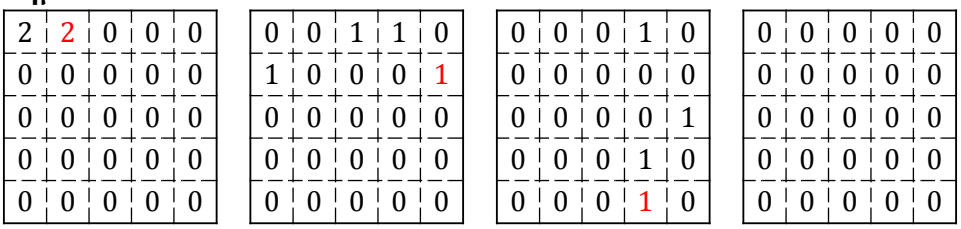
markov[0] markov[1] markov[2] markov[3]

Βήμα 3: bad



markov[0] markov[1] markov[2] markov[3]

Βήμα 4: bed



markov[0] markov[1] markov[2] markov[3]

Βήμα 5: bee

2	3	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[0]

0	0	1	1	0
1	0	0	0	2
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[1]

0	0	0	1	0
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1

markov[2]

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[3]

Βήμα 6: cab

2	3	1	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[0]

0	0	1	1	0
1	0	0	0	2
1	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[1]

0	1	0	1	0
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1

markov[2]

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[3]

Βήμα 7: dad

2	3	1	1	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[0]

0	0	1	1	0
1	0	0	0	2
1	0	0	0	0
1	0	0	0	0
0	0	0	0	0

markov[1]

0	1	0	2	0
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1

markov[2]

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[3]

Βήμα 8: dead

2	3	1	2	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[0]

0	0	1	1	0
1	0	0	0	2
1	0	0	0	0
1	0	0	0	1
0	0	0	0	0

markov[1]

0	1	0	2	0
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
1	0	0	1	1

markov[2]

0	0	0	1	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

markov[3]

Πίνακας 7: Παράδειγμα παραγόμενων Μαρκοβιανών πινάκων.

Τέλος, ο τρισδιάστατος πίνακας που παράχθηκε, γράφεται σε ένα αρχείο στον δίσκο, ώστε να μπορεί να χρησιμοποιηθεί επανειλημμένα για την παραγωγή κωδικών, χωρίς να παράγεται σε κάθε εκτέλεση.

4.2 Παραγωγή Κωδικών με Βάση τους Μαρκοβιανούς Πίνακες

Αφού παραχθούν οι Μαρκοβιανοί πίνακες, περνάμε στην φάση παραγωγής των κωδικών με βάση τους πίνακες αυτούς.

Αρχικά, πρέπει να τοποθετήσουμε τον τρισδιάστατο πίνακα πίσω στην μνήμη, διαβάζοντας τον από το αρχείο στο οποίο είναι αποθηκευμένος στον δίσκο. Αφού τον διαβάσουμε, δημιουργούμε έναν πίνακα ίδιων διαστάσεων, που όμως αντί να κρατάει μόνο την συχνότητα εμφάνισης του

χαρακτήρα σε κάθε κελί, κρατάει ένα ζευγάρι που ως πρώτο στοιχείο έχει τον ίδιο τον χαρακτήρα, και ως δεύτερο στοιχείο έχει την συχνότητα εμφάνισης αυτού του χαρακτήρα.

Συνεχίζοντας το παράδειγμα της προηγούμενη υποενότητας, ο νέος πίνακας με τα ζευγάρια φαίνεται στον Πίνακα 8.

('a',2)	('b',3)	('c',1)	('d',2)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)

pairs[0]

('a',0)	('b',0)	('c',1)	('d',1)	('e',0)
('a',1)	('b',0)	('c',0)	('d',0)	('e',2)
('a',1)	('b',0)	('c',0)	('d',0)	('e',0)
('a',1)	('b',0)	('c',0)	('d',0)	('e',1)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)

pairs[1]

('a',0)	('b',1)	('c',0)	('d',2)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',1)
('a',0)	('b',0)	('c',0)	('d',1)	('e',0)
('a',1)	('b',0)	('c',0)	('d',1)	('e',1)

pairs[2]

('a',0)	('b',0)	('c',0)	('d',1)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)

pairs[3]

Πίνακας 8: Παράδειγμα Μαρκαβιανού πίνακα με ζευγάρια.

Στην συνέχεια, ταξινομούμε τα ζεύγη κάθε γραμμής του πίνακα, για κάθε τετραγωνικό πίνακα, κατά φθίνουσα σειρά με βάση την συχνότητα, σπάζοντας τις ισοβαθμίες κατά αύξουσα σειρά με βάση τον χαρακτήρα. Ετσι ο πίνακας του παραδείγματος θα έχει την τελική μορφή που φαίνεται στον Πίνακα 9.

Για την παραγωγή κάθε κωδικού με μήκος n, επιλέγουμε τον χαρακτήρα pairs$[0][0][0]$ ως πρώτο χαρακτήρα του κωδικού, έστω p_0. Επειτα, επιλέγουμε τον χαρακτήρα pairs$[1][p_0][0]$ ως τον δεύτερο χαρακτήρα, έστω p_1. Συνεχίζουμε με τον ίδιο τρόπο μέχρι να συμπληρώσουμε έναν κωδικό με το ζητούμενο μήκος, επιλέγοντας ως χαρακτήρα για την i θέση, αυτόν που βρίσκεται στο κελί pairs$[i][p_{i-1}][0]$.

Αφού δοκιμάσουμε αυτόν τον κωδικό, αλλάζουμε τον τελευταίο χαρακτήρα του κωδικού στον αμέσως επόμενο πιο πιθανό χαρακτήρα για αυτήν την θέση, δηλαδή αυτόν που βρίσκεται στο κελί pairs$[n-1][p_{n-2}][1]$, και δοκιμάζουμε τον καινούριο κωδικό. Συνεχίζουμε με τον ίδιο τρόπο μέχρι να δοκιμάσουμε και τον χαρακτήρα που βρίσκεται στο κελί pairs$[n-1][p_{n-2}][t-1]$, οπότε και έχουμε δοκιμάσει τους *threshold* πρώτους χαρακτήρες για την τελευταία θέση.

Επειτα αλλάζουμε τον προτελευταίο χαρακτήρα του κωδικού με τον αμέσως επόμενο πιο πιθανό χαρακτήρα, δηλαδή αυτόν που βρίσκεται στο κελί pairs$[n-2][p_{n-3}][1]$, και προχωρώντας στην

('b',3)	('a',2)	('d',2)	('c',1)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)

pairs[0]

('c',1)	('d',1)	('a',0)	('b',0)	('e',0)
('e',2)	('a',1)	('b',0)	('c',0)	('d',0)
('a',1)	('b',0)	('c',0)	('d',0)	('e',0)
('a',1)	('e',1)	('b',0)	('c',0)	('d',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)

pairs[1]

('d',2)	('b',1)	('a',0)	('c',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('e',1)	('a',0)	('b',0)	('c',0)	('d',0)
('d',1)	('a',0)	('b',0)	('c',0)	('e',0)
('a',1)	('d',1)	('e',1)	('b',0)	('c',0)

pairs[2]

('d',1)	('a',0)	('b',0)	('c',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)
('a',0)	('b',0)	('c',0)	('d',0)	('e',0)

pairs[3]

Πίνακας 9: Παράδειγμα Μαρκαβιανού πίνακα με ταξινομημένα ζευγάρια.

τελευταία θέση αλλάζουμε τον χαρακτήρα με τον πρώτο πιο πιθανό για αυτήν την θέση, δηλαδή αυτόν που βρίσκεται στο κελί pairs$[n-1][p_{n-2}][0]$.

Η όλη διαδικασία θυμίζει μια αύξουσα μέτρηση, όπως φαίνεται στον Πίνακα 10, ή την μέθοδο ωμής βίας όπως φαίνεται στον Πίνακα 3, όπου ξεκινώντας από το τελευταίο ψηφίο, και πηγαίνοντας προς τα αριστερά, αυξάνεται το ψηφίο μέχρι να εξαντληθεί το εύρος τιμών για το ψηφίο αυτό, μετά αυξάνεται το προηγούμενο ψηφίο και το παρόν ψηφίο πηγαίνει πάλι στην αρχική τιμή, με την διαδικασία να συνεχίζει μέχρι να εξαντληθεί το εύρος τιμών των ψηφίων για κάθε θέση. Μόνο που στην περίπτωση των Μαρκοβιανών πινάκων, η σειρά των ψηφίων δεν είναι ίδια για κάθε θέση, όπως στις δύο παραπάνω περιπτώσεις, και εξαρτάται επίσης και από την τιμή του αμέσως προηγούμενου ψηφίου.

000	010	$\cdots$	100	110	$\cdots$	990
001	011	$\cdots$	101	111	$\cdots$	991
002	012	$\cdots$	102	112	$\cdots$	992
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
009	019	$\cdots$	109	119	$\cdots$	999

Πίνακας 10: Παράδειγμα με αριθμητικό ισοδύναμο.

Ας δούμε τώρα πως εφαρμόζεται η παραπάνω διαδικασία στους Μαρκοβιανούς πίνακες του παραδείγματός μας. Για διευκόλυνση θα θεωρήσουμε το threshold $t = 3$, δηλαδή για κάθε θέση θα δοκιμάζουμε κάθε φορά μόνο τους 3 πιο πιθανούς χαρακτήρες. Θεωρούμε επίσης πως το μή-

κος των κωδικών που θέλουμε να παράγουμε είναι $n = 3$. Η ακολουθία των παραγόμενων κωδικών φαίνεται στον Πίνακα 11. Οπως είχαμε υπολογίσει και στην Υποενότητα 3.3, παράγουμε $t^n = 3^3 = 27$ διαφορετικούς κωδικούς.

bea	ace	dad
bed	aca	dab
bee	acb	daa
bad	add	dea
bab	ada	ded
baa	adb	dee
bba	aad	dba
bbb	aab	dbb
bbc	aaa	dbc

Πίνακας 11: Παράδειγμα παραγωγής κωδικών με βάση τον Μαρκοβιανό πίνακα.

5 Προτάσεις για Επιλογή Καλύτερων Κωδικών

Κλείνοντας, θα προτείνουμε τρόπους με τους οποίους μπορείτε να βελτιώσετε την επιλογή των κωδικών σας, ώστε να είστε προστατευμένοι ενάντια στις επιθέσεις που περιγράφηκαν παραπάνω.

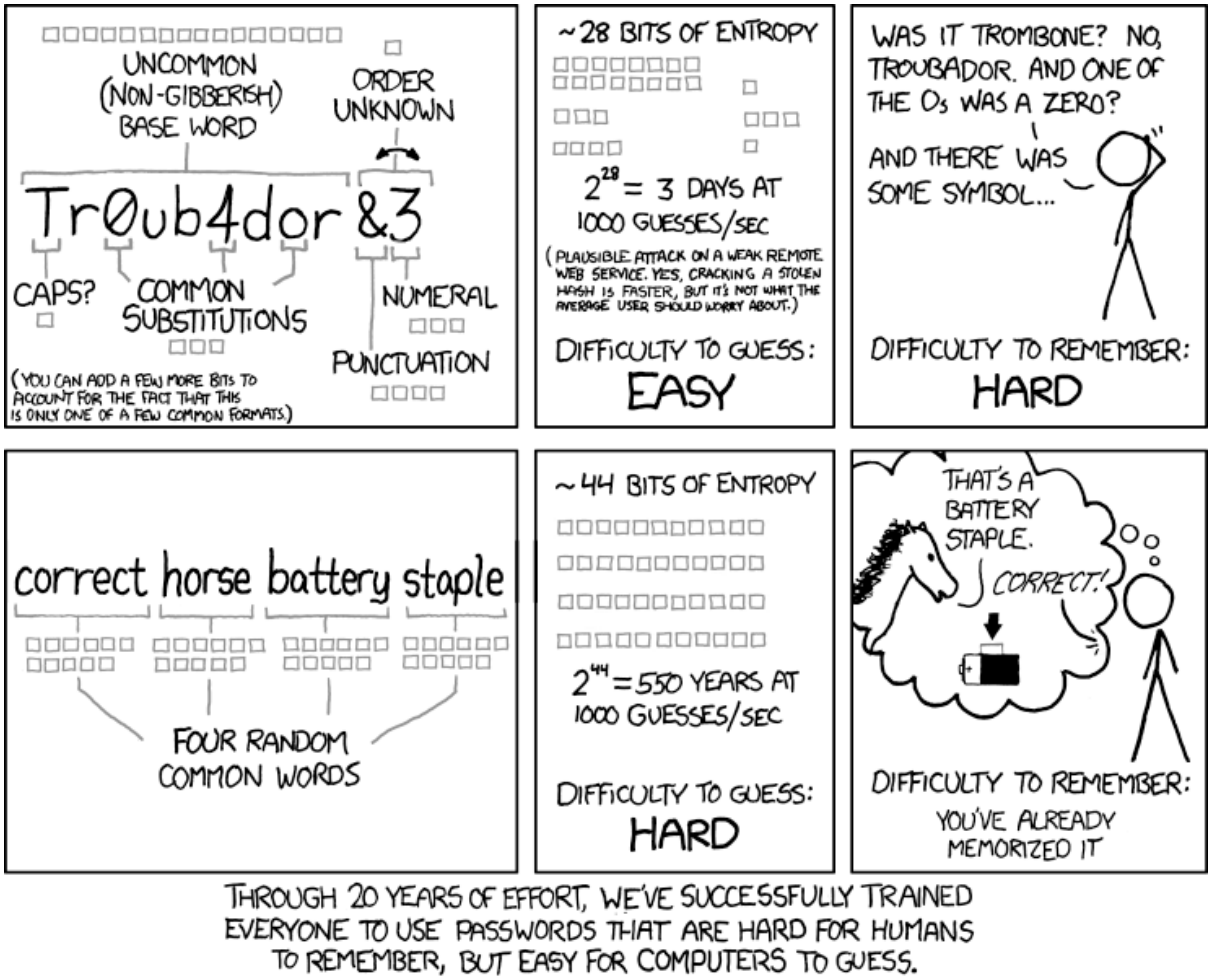
Ως βασική ιδέα θα πρέπει να κρατήσετε πως, "αν μπορείτε να θυμηθείτε τον κωδικό σας, τότε κάποιος άλλος μπορεί να τον μαντέψει εύκολα." Σίγουρα δεν είστε οι πρώτοι που σκεφτήκατε να αντικαταστήσετε ένα γράμμα με κάποιο σύμβολο ή να ξεκινήσετε τον κωδικό με έναν κεφαλαίο χαρακτήρα ή να προσθέσετε στο τέλος του κωδικού σας έναν αριθμό.

Με αυτό κατά νου, προτείνεται η χρήση ενός *διαχειριστή κωδικών (password manager)* ο οποίος μπορεί να παράγει τυχαίους κωδικούς και να τους αποθηκεύει. Επειτα αποκτάτε πρόσβαση σε όλους τους τυχαία παραγόμενους κωδικούς σας, που έχουν παραχθεί και αποθηκευθεί από τον διαχειριστή κωδικών, κάνοντας χρήση μόνο ενός *κυρίως κωδικού (master password)*, ο οποίος θα είναι ο μόνος κωδικός που θα πρέπει να θυμάστε, και μπορείτε να τον κάνετε όσο δυνατό θέλετε.

Σε περιπτώσεις που δεν μπορείτε να χρησιμοποιήσετε έναν διαχειριστή κωδικών, και πρέπει να θυμάστε τον κωδικό, τότε μία τυχαία ακολουθία χαρακτήρων δεν είναι η καλύτερη λύση, καθώς καθιστά τον κωδικό πιο δυσμνημόνευτο και οδηγεί σε λανθασμένες λύσεις, όπως η αποτύπωση του κωδικού σε ένα χαρτάκι κοντά στον υπολογιστή, το οποίο όμως μπορεί να διαβαστεί από όλους όσους έχουν πρόσβαση στον υπολογιστή, ή η χρήση πολύ μικρού μήκους κωδικών. Σε γενικές γραμμές, οι κωδικοί που έχουν μήκος μικρότερο από 12 χαρακτήρες, θεωρούνται πως δεν προσφέρουν καμία ασφάλεια [Wik17i]. Σε αυτές τις περιπτώσεις, προτείνεται η χρήση ενός *συνδυασμού λέξεων (passphrase)*, τυχαία επιλεγμένων, συνήθως 4–5 στο πλήθος, οι οποίες παράγουν έναν κωδικό με αρκετά μεγάλο μήκος, ο οποίος δεν μπορεί να βρεθεί με τις μεθόδους που περιγράφηκαν. Ενα καλό παράδειγμα φαίνεται στο Σχήμα 2, της σελίδας XKCD.

Προτείνεται επίσης να αποφεύγεται η επαναχρησιμοποίηση κωδικών, καθώς αν ο κωδικός σας διαρρεύσει από μία σελίδα, μπορεί να τοποθετηθεί σε ένα λεξικό με κωδικούς, και έπειτα να δοκιμασθεί και σε άλλες σελίδες.

Αποφύγετε επίσης να χρησιμοποιείτε στοιχεία σας, τα οποία είναι γνωστά σε τρίτους, όπως ονόματα κατοικίδιων ή μελών της οικογένειάς σας, ημερομηνίες γέννησης ή επετείους και προσωπικά στοιχεία όπως αριθμούς τηλεφώνων ή αστυνομικής ταυτότητας.



Σχήμα 2: Παράδειγμα Κωδικού με Συνδυασμό Λέξεων

Συνοψίζοντας:

- Χρησιμοποιήστε κωδικούς με μήκος τουλάχιστον 12 με 14 χαρακτήρες.
- Χρησιμοποιήστε κεφαλαίους και μικρούς χαρακτήρες, νούμερα, και ειδικούς χαρακτήρες αν επιτρέπεται.
- Παράγετε τους κωδικούς σας τυχαία όπου μπορείτε.
- Αποφύγετε την επαναχρησιμοποίηση κωδικών.
- Αποφύγετε εύκολους κωδικούς που προκύπτουν από μονοπάτια στο πληκτρολόγιο (π.χ. `asdf`), λέξεις που βρίσκονται αυτούσιες σε λεξικά (π.χ. `football`), αριθμητικές ακολουθίες (π.χ. `123789654`) και προσωπικά στοιχεία (π.χ. `17051994`).

Αναφορές

[Tur12] Daniel Turner. *Hashcat Per Position Markov Chains*. 17 Ιούλ. 2012. URL: https://www.trustwave.com/Resources/SpiderLabs-Blog/Hashcat-Per-Position-Markov-Chains/.

[Pow14] Victor Powell. *Markov Chains. Explained Visually*. 7 Νοέ. 2014. URL: http://setosa.io/ev/markov-chains/.

[Sec17] Defuse Security. *Salted Password Hashing. Doing it Right*. 1 Αύγ. 2017. URL: https://crackstation.net/hashing-security.htm.

[Wik17a] Wikipedia. *Brute-force attack — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Brute-force_attack.

[Wik17b] Wikipedia. *Dictionary attack — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Dictionary_attack.

[Wik17c] Wikipedia. *Graphic character | ISO/IEC 646 — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Graphic_character#ISO/IEC_646.

[Wik17d] Wikipedia. *Graphic character | Unicode — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Graphic_character#Unicode.

[Wik17e] Wikipedia. *Hash function — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Hash_function.

[Wik17f] Wikipedia. *Markov chain — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Markov_chain.

[Wik17g] Wikipedia. *MD5 — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=MD5.

[Wik17h] Wikipedia. *Password cracking — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Password_cracking.

[Wik17i] Wikipedia. *Password strength | Common guidelines — Wikipedia, The Free Encyclopedia*. [Online; accessed 2-January-2018]. 2017. URL: https://en.wikipedia.org/wiki/Password_strength#Common_guidelines.

[Wik17j] Wikipedia. *Rainbow table — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-December-2017]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Rainbow_table.