ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τίτλος Εργασίας

ΥΛΟΠΟΙΗΣΗ SEARCHABLE ENCRYPTION SCHEME

Κομίνης Μιχάλης
Α.Μ: 21444

Αθήνα, 2019

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τριμελής Εξεταστική Επιτροπή

Παναγιώτης Ριζομυλιώτης (Επιβλέπων)
Επίκουρος Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Δημοσθένης Αναγνωστόπουλος
Καθηγητής, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Μαλβίνα Βαμβακάρη
Αναπληρώτρια Καθηγήτρια, Τμήμα Πληροφορικής και Τηλεματικής,
Χαροκόπειο Πανεπιστήμιο

Ο Μιχάλης Κομίνης

δηλώνω υπεύθυνα ότι:

1) Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2) Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Επίκουρο Καθηγητή του Χαροκοπείου Πανεπιστημίου και επιβλέπων καθηγητή μου κ. Παναγιώτη Ριζομυλιώτη για τις συμβουλές και την καθοδήγηση κατά τη διάρκεια υλοποίησης της πτυχιακής μου εργασίας.

Θερμά ευχαριστώ στην οικογένειά μου για την πλήρη στήριξη και κατανόηση που έδειξαν καθ' όλη τη διάρκεια των σπουδών μου.

Να ευχαριστήσω ακόμη τους κοντινούς μου ανθρώπους που μου συμπαραστάθηκαν στα εύκολα και στα δύσκολα.

Περίληψη στα Ελληνικά

Με την ανάπτυξη του υπολογιστικού νέφους, οι χρήστες αποθηκεύουν σε αναξιόπιστους διακομιστές τα αρχεία τους. Εφόσον τα αρχεία είναι κρυπτογραφημένα οποιαδήποτε προσπάθεια αναζήτησης σε αυτά είναι αδύνατη.

Searchable Symmetric Encryption (SSE) είναι μια διαδικασία που επιτρέπει σε μία οντότητα (πελάτης) να αποθηκεύει τα δεδομένα της σε μία άλλη (διακομιστής), διατηρώντας παράλληλα τη δυνατότητα επιλεκτικής αναζήτησης σε αυτά.

Η πρακτικότητα που προσφέρεται έχει το κόστος διαρροής πληροφορίας. Τα σχήματα ανάλογα με τη διαρροή λέμε ότι είναι forward ή/και backward private. Ένα σχήμα είναι forward private όταν τα ερωτήματα ενημέρωσης δεν διαρρέουν καμία πληροφορία σχετικά με τις εμπλεκόμενες λέξεις/κλειδιά και backward private όταν τα ερωτήματα αναζήτησης δεν αποκαλύπτουν καμία πληροφορία σχετικά με διαγραμμένα έγγραφα.

Αντικείμενο της πτυχιακής εργασίας είναι η υλοποίηση ενός σχήματος αναζήτησης σε κρυπτογραφημένα δεδομένα με λέξη κλειδί. Η υλοποίηση του σχήματος έγινε με τη γλώσσα προγραμματισμού C++ και τις βιβλιοθήκες OpenSSL και Libsodium για την κρυπτογράφηση των δεδομένων.

Λέξεις κλειδιά

searchable encryption, κρυπτογραφημένα δεδομένα, forward/backward privacy

Abstract ή Περίληψη στα Αγγλικά

With the development of cloud computing, users store their files on untrusted servers. As long as the files are encrypted any attempt to search them is impossible.

Searchable Symmetric Encryption (SSE) allows a party (client) to outsource the storage of its data to another party (server) in a private manner, while maintaining the ability to selectively search over it.

The practicality offered has the cost of information leakage. The schemes depending on the leakage are called forward and/or backward private. One scheme is forward private when the update queries do not leak any information about the involved keywords and it is backward private when the search queries do not reveal any information regarding deleted documents.

The purpose of the thesis is to implement a searchable encryption scheme. The scheme was implemented with the programming language C++ as well as the libraries OpenSSL and Libsodium to encrypt the data.

Keywords

searchable encryption, encrypted data, forward/backward privacy

Πίνακας Περιεχομένων

Περίληψη στα Ελληνικά.....6
Abstract ή Περίληψη στα Αγγλικά.....7
Πίνακας Περιεχομένων.....8
Κατάλογος Εικόνων.....9
Κατάλογος Σχημάτων.....10
1. Εισαγωγή.....11
1.1 Υπολογιστικό Νέφος και Ιδιωτικότητα.....11
1.2 Symmetric Searchable Encryption (SSE).....12
1.3 Διαρροή (Leakage).....13
1.4 Dynamic SSE (DSSE).....13
1.5 Επισκόπηση.....14
2. Θεωρητικό Υπόβαθρο.....16
2.1 Κρυπτογραφικές Αρχές.....16
2.2 Σχεδίαση MARA.....16
2.3 Σχεδίαση SORAM.....18
2.4 Πλήρες Δυαδικό Δέντρο.....19
3. Βασική Δομή.....20
3.1 Τύποι και Ορίσματα.....20
3.2 Η Κλάση AES.....22
3.2.1 Κρυπτογράφηση του Δέντρου.....22
3.2.2 Υπολογισμός Μονοπατιού.....25
3.3 Η κλάση BinaryTree.....27
3.4 Η Κλάση SORAM.....29
4. Mara DSSE Scheme.....32
4.1 Οι Αλγόριθμοι.....32
4.1.1 Setup.....32
4.1.2 Update.....33
4.1.3 Search.....35
4.2 Ο Αλγόριθμος SORAM.....35
4.2.1 Setup.....35
4.2.2 Insert.....36
4.2.3 Delete.....38
4.3 Βοηθητικές Συναρτήσεις.....40
4.3.1 DelRedund.....40
4.3.2 Evict.....42
4.3.3 Pth.....43
5. Αποτελέσματα.....44
Βιβλιογραφία.....46

Κατάλογος Εικόνων

Εικόνα 2.1 Αναπαράσταση Πλήρους Δυαδικού Δέντρου.....19
Εικόνα 3.1 Βασικοί Τύποι.....20
Εικόνα 3.2 Μέγεθος Κλειδιών και IV.....20
Εικόνα 3.3 Τύπος OP.....20
Εικόνα 3.4 Structure datablocks.....21
Εικόνα 3.5 Structure Block.....21
Εικόνα 3.6 “Κουβάς” Bucket.....21
Εικόνα 3.7 Κλάση Node.....21
Εικόνα 3.8 AES Setup και Cleanup.....22
Εικόνα 3.9 AES Encrypt.....23
Εικόνα 3.10 AES GenerateIV.....23
Εικόνα 3.11 AES EncryptBlock.....23
Εικόνα 3.12 AES EncryptBytes.....24
Εικόνα 3.13 AES Decrypt.....24
Εικόνα 3.14 AES DecryptBlock.....25
Εικόνα 3.15 AES DecryptBytes.....25
Εικόνα 3.16 Μέθοδος ComputeToken.....26
Εικόνα 3.17 Μέθοδος ComputePath.....26
Εικόνα 3.18 Μέθοδος extractNodes.....27
Εικόνα 3.19 Μέθοδος extractRandNodes.....28
Εικόνα 3.20 Μέθοδος insertNodes.....28
Εικόνα 3.21 Μέθοδος SerialiseBucket.....29
Εικόνα 3.22 Μέθοδος DeserialiseBucket.....30
Εικόνα 3.23 Μέθοδος EncryptNode.....30
Εικόνα 3.24 Μέθοδος ReadBucket.....31
Εικόνα 3.25 Μέθοδος ReadBlock.....31
Εικόνα 3.26 Μέθοδος SetNode.....31
Εικόνα 4.1 Client-Server Setup.....33
Εικόνα 4.2 Client Header File.....33
Εικόνα 4.3 Client Update.....34
Εικόνα 4.4 Client Search.....35
Εικόνα 4.5 SORAM WriteBucket.....36
Εικόνα 4.6 SORAM setup.....36
Εικόνα 4.7 Μέθοδος SORAM Insert.....37
Εικόνα 4.8 Μέθοδος SORAM Delete.....39
Εικόνα 4.9 Μέθοδος Server Remove.....40
Εικόνα 4.10 Μέθοδος DelRedund.....41
Εικόνα 4.11 Μέθοδος Evict.....42
Εικόνα 4.12 Μέθοδος Pth.....43

Κατάλογος Σχημάτων

Σχήμα 1 - Χρόνος Αναζήτησης για |DB|=100K και 100 δεδομένα....44

Σχήμα 2 – Χρόνος Αναζήτησης για |DB|=100K και 20K δεδομένα...45

1. Εισαγωγή

1.1 Υπολογιστικό Νέφος και Ιδιωτικότητα

Με τη ραγδαία ανάπτυξη των τεχνολογιών cloud computing, οι χρήστες τείνουν να αποθηκεύουν τα δεδομένα τους σε απομακρυσμένους διακομιστές (νέφος). Το κύριο πλεονέκτημα της αποθήκευσης στο νέφος είναι η πανταχού παρούσα προσβασιμότητα του χρήστη και οι πρακτικά απεριόριστες δυνατότητες αποθήκευσης δεδομένων.

Παρά τα οφέλη που προσφέρει το σύννεφο, η μεγάλη πρόκληση που παραμένει είναι η ανησυχία για την εμπιστευτικότητα και την ιδιωτικότητα των δεδομένων, ενώ υιοθετούνται οι υπηρεσίες αποθήκευσης νέφους. Για παράδειγμα, τα μη κρυπτογραφημένα δεδομένα χρηστών που είναι αποθηκευμένα στον απομακρυσμένο διακομιστή μπορεί να είναι ευάλωτα σε εξωτερικές επιθέσεις που προκαλούνται από μη εξουσιοδοτημένους τρίτους και εσωτερικές επιθέσεις που ξεκινούν από τους αναξιόπιστους παρόχους υπηρεσιών νέφους. Υπάρχουν αρκετές αναφορές που επιβεβαιώνουν παραβιάσεις δεδομένων που σχετίζονται με διακομιστές νέφους, λόγω κακόβουλων επιθέσεων, κλοπής ή εσωτερικών σφαλμάτων. Αυτό δημιουργεί ανησυχίες σε πολλούς χρήστες και οργανισμούς, καθώς τα δεδομένα που ανατίθενται σε εξωτερικούς συνεργάτες ενδέχεται να περιέχουν πολύ ευαίσθητες προσωπικές πληροφορίες.

Αρκετές έρευνες έχουν αντιμετωπίσει το ζήτημα της διασφάλισης της εμπιστευτικότητας και της ιδιωτικότητας των δεδομένων του cloud χωρίς να διακυβεύεται η λειτουργικότητα του χρήστη. Εδώ, η εμπιστευτικότητα αναφέρεται στην μυστικότητα των αποθηκευμένων δεδομένων, έτσι ώστε μόνο ο πελάτης να μπορεί να διαβάσει τα περιεχόμενα των αποθηκευμένων δεδομένων. Για την επίλυση του προβλήματος της εμπιστευτικότητας, τα συστήματα κρυπτογράφησης δεδομένων παρέχουν στους χρήστες κάποιον έλεγχο σχετικά με το απόρρητο των αποθηκευμένων δεδομένων τους. Αυτό επιτρέπει στους χρήστες να κρυπτογραφούν τα δεδομένα τους πριν την εξωτερική ανάθεση στο νέφος. Ωστόσο, τα τυπικά προγράμματα κρυπτογράφησης αφαιρούν πλήρως την ικανότητα αναζήτησης των χρηστών πάνω στα αποθηκευμένα δεδομένα, αφού μετά την κρυπτογράφηση ο χρήστης απλά δεν μπορεί να χρησιμοποιήσει μια λέξη-κλειδί για να πραγματοποιήσει μια αναζήτηση και συνεπώς δεν μπορεί να ανακτήσει το περιεχόμενο με αποτελεσματικό τρόπο.

Η αποθήκευση μιας κρυπτογραφημένης βάσης δεδομένων σε έναν απομακρυσμένο διακομιστή, διατηρώντας παράλληλα τη δυνατότητα πρόσβασης και δυναμικής διατήρησής του, είναι θεμελιώδους σημασίας για τον σύγχρονο υπολογισμό. Θεωρείστε, για παράδειγμα, έναν πελάτη που κατέχει μια βάση δεδομένων, την αποθηκεύει σε έναν μη αξιόπιστο διακομιστή και στη συνέχεια κάνει ερωτήματα αναζήτησης της μορφής "ανάκτησε όλα τα έγγραφα που περιέχουν τη λέξη κλειδί ω". Ιδανικά, ο διακομιστής θα πρέπει όχι μόνο να μην μάθει τίποτα σχετικά με το περιεχόμενο των εγγράφων (που μπορεί να επιτευχθεί με παραδοσιακά κρυπτογραφικά συστήματα), αλλά και άλλες πρόσθετες πληροφορίες, π.χ. πόσες φορές έχει γίνει αναζήτηση, τη συχνότητα λέξεων/κλειδιών στην βάση δεδομένων κ.λπ. Αυτό το "τέλειο" επίπεδο προστασίας της ιδιωτικότητας είναι θεωρητικά εφικτό από τις ισχυρές τεχνικές κρυπτογράφησης, όπως η πλήρης ομομορφική κρυπτογράφηση, των οποίων όμως η πολυπλοκότητα περιορίζει την εφαρμογή στην πράξη.

1.2 Symmetric Searchable Encryption (SSE)

Η συμμετρική κρυπτογράφηση με δυνατότητα αναζήτησης (SSE) παρέχει έναν τρόπο για την αποτελεσματική πρόσβαση σε αυτήν την κρυπτογραφημένη βάση δεδομένων, χαλαρώνοντας ελαφρώς τις απαιτήσεις απορρήτου. Συγκεκριμένα, το SSE αποκαλύπτει μερικές πληροφορίες στο διακομιστή κατά την εκτέλεση ερωτημάτων, γνωστή ως διαρροή (leakage). Αυτή η διαρροή περιλαμβάνει συνήθως το μοτίβο αναζήτησης που αποκαλύπτει τα ερωτήματα αναζήτησης που αφορούν την ίδια λέξη/κλειδί καθώς και το πρότυπο πρόσβασης που αποκαλύπτει τα αρχεία που επιστρέφονται για ένα ερώτημα.

Το SSE είναι επικεντρωμένο σε στατικά σύνολα δεδομένων και μέχρι το 2009 δεν υπήρχε κάποιο σχήμα που να υποστηρίζει ενημερώσεις στη βάση δεδομένων (γνωστά ως Dynamic SSE schemes, παράγραφος 1.4). Το Dynamic SSE, ωστόσο, εισάγει επιπρόσθετα προβλήματα εξαιτίας της επιπλέον λειτουργικότητας που παρέχει. Για παράδειγμα, (α) η εισαγωγή ενός αρχείου f στη βάση δεδομένων μπορεί να αποκαλύψει ότι περιέχει λέξεις/κλειδιά για τα οποία έγινε αναζήτηση προηγουμένως, ή (β) η αναζήτηση για μία λέξη/κλειδί ω μπορεί να αποκαλύψει ποια αρχεία από το παρελθόν, τα οποία έχουν αφαιρεθεί από τη βάση δεδομένων τώρα, περιείχαν το ω.

1.3 Διαρροή (Leakage)

Τα σχήματα που αποφεύγουν τη διαρροή που σχετίζεται με την περίπτωση (α) ονομάζονται forward private. Το forward privacy έχει γίνει ένα ουσιώδης συστατικό για τα DSSE σχήματα καθώς χωρίς αυτό, τα σχήματα είναι ευάλωτα σε file injection επιθέσεις.

Τα σχήματα που προσπαθούν να περιορίσουν τη διαρροή από την περίπτωση (β) ονομάζονται backward private. Για το backward privacy υπάρχουν τρεις διαφορετικοί τύποι διαρροής με κατάταξη από τον πιο ασφαλή προς τον λιγότερο ασφαλή.

Type-I leakage (Backward Privacy with Insertion Pattern): Τα σχήματα αυτά αποκαλύπτουν, κατά τη διάρκεια της αναζήτησης για μία λέξη/κλειδί ω, τον αριθμό και τον τύπο των προηγούμενων ενημερώσεων συσχετισμένων με το ω, τα αναγνωριστικά των αρχείων που βρίσκονται στη βάση δεδομένων και περιέχουν το ω καθώς και πότε έγινε η εισαγωγή αυτών των αρχείων.

Type-II leakage (Backward Privacy with Update Pattern): Η πληροφορία που διαρρέεται είναι η ίδια με τα σχήματα από το Type-I, αλλά επιπλέον αποκαλύπτεται πότε έλαβαν χώρα όλες οι ενημερώσεις που σχετίζονται με το ω.

Type-III leakage (Weak Backward Privacy): Τέλος, ο τρίτος τύπος διαρροής αποκαλύπτει ακόμη ποια ενημέρωση διαγραφής ακύρωσε προηγούμενες ενημερώσεις εισαγωγής (π.χ. η διαγραφή που έγινε κατά τη διάρκεια της δέκατης πράξης ακύρωσε την προσθήκη από την πέμπτη πράξη).

Να σημειωθεί πως και οι τρεις τύποι ικανοποιούν τη βασική ιδιότητα της απόκρυψης των πραγματικών αναγνωριστικών των αρχείων που περιείχαν το ω αλλά διαγράφηκαν πριν από την αναζήτηση του ω.

1.4 Dynamic SSE (DSSE)

Ένα DSSE σχήμα Σ αποτελείται από μία τριάδα λειτουργιών (Setup, Search, Update) ανάμεσα σε έναν πελάτη και έναν διακομιστή.

Setup(1^{λ},N): Αυτός ο πιθανοτικός αλγόριθμος λαμβάνει ως είσοδο τον αριθμό των ζεύγων εγγράφων-λέξεων/κλειδιών N και την παράμετρο ασφαλείας λ και δημιουργεί μία κατάσταση σ για τον πελάτη. Επίσης, υπολογίζει το κρυπτογραφημένο δυαδικό δέντρο BT το οποίο στέλνετε στον διακομιστή.

Search(σ,q; BT): Πρόκειται για έναν αλγόριθμο μεταξύ πελάτη και διακομιστή. Ο πελάτης χρησιμοποιώντας την εσωτερική του κατάσταση σ προετοιμάζει ένα token αναζήτησης για το ερώτημα q. Εφόσον το σχήμα μας υποστηρίζει μόνο μεμονωμένες αναζητήσεις λέξεων/κλειδιών, μόνο η λέξη/κλειδί ω είναι απαραίτητη ως είσοδος. Ο διακομιστής χρησιμοποιεί το token για να εντοπίσει το αντίστοιχο υποσύνολο των κρυπτογραφημένων μπλοκ δεδομένων από το δυαδικό δέντρο BT. Ο διακομιστής στέλνει τα κρυπτογραφημένα μπλοκ στον πελάτη και ο πελάτης τα αποκρυπτογραφεί.

Update(σ,Data; BT): Πρόκειται επίσης για έναν αλγόριθμο μεταξύ του πελάτη και του διακομιστή. Υποστηρίζει τις λειτουργίες προσθήκης και διαγραφής μπλοκ δεδομένων στο δυαδικό δέντρο BT. Δηλαδή, μπορεί να προσθέσει ή/και να αφαιρέσει ένα σύνολο μπλοκ δεδομένων Data. Τα δεδομένα είναι ένα σύνολο από αναγνωριστικά αρχείων, λέξεων/κλειδιών και την ένδειξη 0 ή 1 για διαγραφή ή εισαγωγή αντίστοιχα. Ο πελάτης προετοιμάζει ένα token χρησιμοποιώντας την τοπική του κατάσταση σ και ο διακομιστής ενημερώνει το δυαδικό δέντρο BT.

Ένα DSSE σχήμα θεωρείται σωστό, εάν η λειτουργία αναζήτησης επιστρέφει τα σωστά αποτελέσματα για κάθε ερώτημα.

1.5 Επισκόπηση

Εδώ παρουσιάζεται μία γενική εικόνα για το κάθε κεφάλαιο.

Κεφάλαιο 2 – Θεωρητικό Υπόβαθρο. Σε αυτό το κεφάλαιο περιγράφονται οι αλγόριθμοι και οι τεχνικές που χρησιμοποιήθηκαν για τις κρυπτογραφικές ανάγκες του σχήματος. Ακόμη αναλύεται η σχεδίασή του και οι βασικές δομές δεδομένων που χρησιμοποιεί.

Κεφάλαιο 3 – Βασική Δομή. Εδώ παρουσιάζονται οι κλάσεις και τα ορίσματα που υποστηρίζουν την υλοποίηση του σχήματος. Η κλάση AES όπου υλοποιούνται οι κρυπτογραφικές συναρτήσεις, η BinaryTree όπου γίνεται η αλληλεπίδραση με το δυαδικό δέντρο και η κλάση SORAM η οποία λειτουργεί ως μία διεπαφή για την ανάγνωση/εισαγωγή των δεδομένων από και προς το δέντρο.

Κεφάλαιο 4 – MARA DSSE Scheme. Σε αυτό το κεφάλαιο αναλύονται οι μέθοδοι του σχήματος MARA και του αλγορίθμου SORAM και η αλληλεπίδραση μεταξύ του πελάτη και του διακομιστή.

Κεφάλαιο 5 – Αποτελέσματα. Εδώ παρουσιάζονται τα αποτελέσματα που καταμετρήθηκαν για το σχήμα που υλοποιήθηκε.

Ο κώδικας που αναπτύχθηκε και περιγράφεται στην παρούσα πτυχιακή εργασία μπορεί να βρεθεί στην ιστοσελίδα του Github από τον ακόλουθο σύνδεσμο.

https://github.com/kominis/DSSE

2. Θεωρητικό Υπόβαθρο

2.1 Κρυπτογραφικές Αρχές

Για τις κρυπτογραφικές ανάγκες της εργασίας χρησιμοποιείται ο AES σε λειτουργία CBC-256 με κλειδί k_{SO} για την κρυπτογράφηση των κόμβων του δέντρου. Για την χρήση του αλγορίθμου χρησιμοποιείται η βιβλιοθήκη OpenSSL [1]. Ο αλγόριθμος επιλέχθηκε καθώς ο AES είναι υλοποιημένος στους σύγχρονους επεξεργαστές με αποτέλεσμα την επιτάχυνση της κρυπτογράφησης.

Ακόμη γίνεται χρήση του αλγορίθμου Blake2 [2] ο οποίος λειτουργεί ως μία συνάρτηση κατακερματισμού και δέχεται και κλειδί k_h. Ο αλγόριθμος είναι υπεύθυνος για την παραγωγή μονοπατιών στο δέντρο για τις τετράδες που εισάγονται. Η διαδικασία αναλύεται στην παράγραφο 3.2. Ο αλγόριθμος καλείται μέσω της βιβλιοθήκης Libsodium [3]. Η επιλογή του αλγορίθμου έγινε διότι ο Blake2 είναι όσο γρήγορος είναι και ο MD5 αλλά είναι πιο ασφαλής. Ακόμη έχει βελτιστοποιηθεί για 64-bit πλατφόρμες και παράγει digest από 1 έως 64 bytes.

Τέλος, υλοποιείται ένας Oblivious RAM [4] [5] αλγόριθμος, που ονομάζεται SORAM, για να κρύψουμε το πρότυπο πρόσβασης στη μνήμη, δηλαδή τον τύπο και τη λειτουργία που εκτελείται στις θέσεις μνήμης του διακομιστή.

2.2 Σχεδίαση MARA

Μελετώντας πιο προσεκτικά τους τρεις τύπους διαρροής του backward privacy, φαίνεται ότι και οι τρεις κάνουν την υπόθεση ότι σε ένα backward private DSSE σχήμα, η διαδικασία αναζήτησης διαρρέει στο διακομιστή το αναγνωριστικό των αρχείων, καθώς και την αντίστοιχη ώρα προσθήκης. Το DSSE σχήμα που υλοποιείται στην παρούσα πτυχιακή που ονομάζεται Mara βασίζεται πάνω σε αυτήν την παρατήρηση και εκμεταλλεύεται αυτή τη διαρροή με αποτέλεσμα να έχει αξιοσημείωτη βελτίωση στην απόδοση των ερωτημάτων αναζήτησης.

Πιο συγκεκριμένα, διατηρούμε στο διακομιστή μία ORAM δομή που την διαχειρίζεται ο πελάτης. Όλες οι ενημερώσεις αποθηκεύονται σε αυτήν τη δομή.

Όταν ο πελάτης θέλει να εισάγει (op = 1) ή να διαγράψει (op = 0) ένα ζευγάρι (ind,ω), μία τετράδα της μορφής (ind||ω||op||pth) αποθηκεύεται στην ORAM δομή. Όλες οι τετράδες παραμένουν εκεί μέχρι την επόμενη αναζήτηση για το ω. Οι τετράδες εισάγονται μία φορά στη δομή από τον πελάτη και διαγράφονται μία φορά (όταν γίνεται η αναζήτηση για το ω). Βάση αυτής της ιδιότητας υλοποιήθηκε ο αλγόριθμος SORAM, ο οποίος θεωρείται μία απλοποιημένη έκδοση του αλγορίθμου Path-ORAM [5]. Ο αλγόριθμος εισάγει σε ένα πλήρες δυαδικό δέντρο μπλοκ δεδομένων της μορφής (ind||ω||op|| pth), στους κόμβους του δέντρου που ανήκουν στο μονοπάτι pth. Περισσότερες λεπτομέρειες για την σχεδίαση του αλγορίθμου παρουσιάζονται στην ενότητα 2.3.

Ο πελάτης διατηρεί ακόμη δύο μετρητές για κάθε λέξη/κλειδί, $cnt_{srch}^{(\omega)}$ και $cnt_{upt}^{(\omega)}$. Ο πρώτος είναι ο συνολικός αριθμός αναζητήσεων για το ω και ο δεύτερος είναι ο αριθμός ενημερώσεων που σχετίζονται με το ω, από το τελευταίο ερώτημα αναζήτησης για το ω. Κάθε φορά που ο πελάτης ψάχνει για το ω, η τιμή του μετρητή $cnt_{srch}^{(\omega)}$ αυξάνεται κατά ένα και ο μετρητής $cnt_{upt}^{(\omega)}$ μηδενίζεται. Οι δύο αυτοί μετρητές χρησιμοποιούνται για να εντοπιστούν οι τετράδες για το ω στη δομή SORAM, δηλαδή το μονοπάτι. Το μονοπάτι παράγεται ψευδοτυχαία χρησιμοποιώντας τον Blake2b και δίνεται από τον τύπο

$$pth = Blake2b_{k^{(\omega)}}\left(\omega \,||\, cnt_{upd}^{(\omega)}\right),$$

όπου $k^{(\omega)}$ είναι το τωρινό token του ω που δίνεται από

$$k^{(\omega)} = Blake2b_{k^{h}}\left(\omega \,||\, cnt_{srch}^{(\omega)}\right)$$

και k_h είναι το μυστικό κλειδί που έχει ο πελάτης.

Οι δύο μετρητές $cnt_{upt}^{(\omega)}$ και $cnt_{srch}^{(\omega)}$ αποθηκεύονται τοπικά στον πελάτη στο λεξικό D_x. Το λεξικό χρησιμοποιείται για να συνδέει τους μετρητές με τη λέξη/κλειδί ω. Δηλαδή $D_x[\omega][0]$ είναι ο συνολικός αριθμός αναζητήσεων $cnt_{srch}^{(\omega)}$, ενώ $D_x[\omega][1]$ είναι ο μετρητής ενημερώσεων $cnt_{upt}^{(\omega)}$.

2.3 Σχεδίαση SORAM

Ο αλγόριθμος SORAM χρησιμοποιεί ένα πλήρες δυαδικό δέντρο ως κύρια δομή δεδομένων και κάθε κόμβος του δέντρου θεωρείται ένας “κουβάς” (bucket) στον οποίο μπορούν να αποθηκευτούν έως και Z μπλοκ δεδομένων. Κάθε μπλοκ σχετίζεται με ένα μονοπάτι στο δέντρο και μπορεί να βρεθεί σε κάποιον από τους κόμβους αυτού του μονοπατιού. Τα μπλοκ δεδομένων που αποθηκεύονται στο SORAM είναι της μορφής (ind||ω||op||pth), όπου (ind||ω) είναι το ζευγάρι της λέξης κλειδί με το αναγνωριστικό του αρχείου, op είναι η διαδικασία (insert/delete) και pth το μονοπάτι στο δέντρο που αποθηκεύεται το μπλοκ. Οι κόμβοι αποθηκεύονται κρυπτογραφημένοι στο διακομιστή με τη χρήση του AES σε λειτουργία CBC-256 και μυστικό κλειδί το k_{SO}.

Πρακτικά, χρησιμοποιούμε μία απλουστευμένη και πιο αποδοτική έκδοση του Path-ORAM αλγορίθμου. Οι βασικές διαφορές ανάμεσα στο Path-ORAM και στο SORAM είναι δύο.

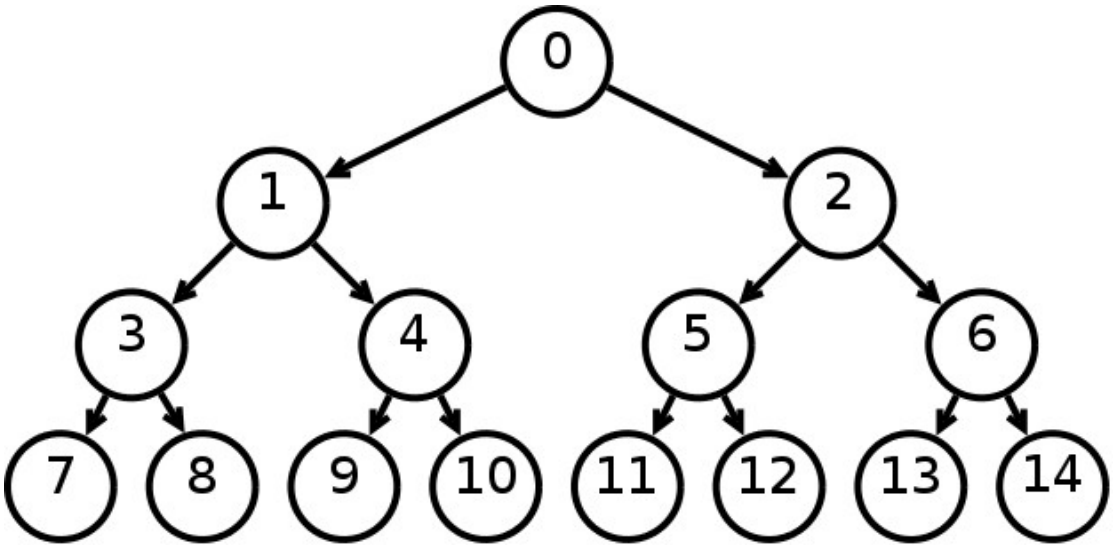
Η πρώτη διαφορά σχετίζεται με τον υπολογισμό του μονοπατιού για τα μπλοκ δεδομένων. Στον αλγόριθμο Path-ORAM, κάθε φορά που γίνεται πρόσβαση σε κάποιο μπλοκ, το μπλοκ αποθηκεύεται σε ένα άλλο, τυχαία επιλεγμένο, μονοπάτι και ο πελάτης πρέπει να διατηρεί ένα position map όπου αποθηκεύονται τα πιο πρόσφατα μονοπάτια. Ενώ αντίθετα στο SORAM, επειδή κάθε μπλοκ εισάγεται και εξάγεται μόνο μία φορά, δεν υπάρχει η ανάγκη διατήρησης του position map καθώς το μονοπάτι μπορεί να υπολογιστεί αποδοτικά με τη βοήθεια του αλγορίθμου Blake2b στη μορφή συνάρτησης κατακερματισμού με μυστικό κλειδί το k_h.

Η δεύτερη διαφορά έχει να κάνει με το γεγονός πως η Mara υποστηρίζει μαζικές ενημερώσεις, γι’ αυτό και ο SORAM είναι σχεδιασμένος να διαβάζει και να γράφει πολλά μπλοκ δεδομένων σε μία παρτίδα. Αντιθέτως ο Path-ORAM επεξεργάζεται ένα μπλοκ δεδομένων τη φορά.

Ο αλγόριθμος υποστηρίζει τρεις λειτουργίες, setup, insert και delete, οι οποίες αναλύονται στην ενότητα 4.2. Η λειτουργία setup είναι υπεύθυνη για την αρχικοποίηση του δέντρου, η insert εισάγει μπλοκ δεδομένων στο δέντρο και η delete αφαιρεί τα μπλοκ που ικανοποιούν τη συνθήκη αναζήτησης.

Ακόμη χρησιμοποιεί κάποιες βοηθητικές συναρτήσεις, την Evict η οποία εισάγει μπλοκ δεδομένων στους κόμβους του δέντρου και την DelRedund η οποία αφαιρεί μπλοκ δεδομένων που ακυρώνουν το ένα το άλλο. Οι συναρτήσεις και οι λειτουργίες τους αναλύονται στην ενότητα 4.3.

Τέλος, ο πελάτης διατηρεί τοπικά μία stash όπου αποθηκεύει τα δεδομένα τα οποία δεν μπορούν να αποθηκευτούν στο δέντρο, όπως και στο Path-ORAM.



Εικόνα 2.1 Αναπαράσταση Πλήρους Δυαδικού Δέντρου

2.4 Πλήρες Δυαδικό Δέντρο

Ένα πλήρες δυαδικό δέντρο ύψους h έχει $2^{h+1}-1$ κόμβους. Οι κόμβοι θεωρούνται “κουβάδες” (buckets) και αποθηκεύουν έως και Z μπλοκ δεδομένων. Κάθε κόμβος έχει ως μοναδικό αναγνωριστικό έναν ακέραιο αριθμό με τη ρίζα να έχει το μηδέν, το αριστερό της παιδί το ένα, το δεξί της παιδί το δύο κ.ο.κ, όπως φαίνεται και στην εικόνα 2.1.

Παρατηρούμε πως κάθε αριστερό παιδί έχει περιττό αναγνωριστικό και κάθε δεξί παιδί άρτιο. Για να βρούμε τον πατέρα ενός κόμβου n εκτελούμε την εξής διαδικασία:

- Αν ο n είναι περιττός, ο πατέρας του θα είναι το πηλίκο της διαίρεσης $\frac{n}{2}$, ενώ
- Αν ο n είναι άρτιος, ο πατέρας θα είναι ο $\frac{n}{2}-1$

Αντίστοιχα για να βρούμε τα παιδιά ενός κόμβου n έχουμε:

- Το αριστερό παιδί του κόμβου n είναι ο κόμβος 2n+1 και
- το δεξί παιδί θα είναι ο κόμβος 2n+2

3. Βασική Δομή

3.1 Τύποι και Ορίσματα

Στην εικόνα 3.1 παραθέτονται κάποιοι βασικοί τύποι που χρησιμοποιούνται σε όλο το πρόγραμμα. Ο τύπος byte_t είναι ένα byte και ο τύπος block αποτελεί μία ακολουθία από bytes. Ο πίνακας bytes μεγέθους N χρησιμοποιείται για την αποθήκευση των κλειδιών κρυπτογράφησης και του IV. Το N παίρνει την τιμή 128 για το μυστικό κλειδί k_{SO}, την τιμή 32 για το κλειδί k_h και την τιμή 16 για το IV όπως φαίνεται στην εικόνα 3.2. Ο σταθερός ακέραιος Z συμβολίζει το πλήθος των μπλοκ δεδομένων που εισάγονται στους κόμβους του δέντρου.

```
7  using byte_t = uint8_t;
8  using block = std::vector<byte_t>;
9
10 template <size_t N>
11 using bytes = std::array<byte_t, N>;
12
13 // A bucket contains a number of Blocks
14 constexpr int Z = 4;
```

Εικόνα 3.1 Βασικοί Τύποι

```
6  // Encryption constants (in bytes)
7  constexpr int IV = 16;
8  constexpr int Key = 128;
9  constexpr int Blake2Key = crypto_generichash_KEYBYTES;
```

Εικόνα 3.2 Μέγεθος Κλειδιών και IV

Ο enum τύπος OP (εικόνα 3.3) παίρνει την τιμή 0 για τη λειτουργία διαγραφής και την τιμή 1 για τη λειτουργία εισαγωγής.

```
13 enum OP{
14     DEL,INS
15 };
```

Εικόνα 3.3 Τύπος OP

Το structure datablocks (εικόνα 3.4) είναι υπεύθυνο για την προσωρινή αποθήκευση των δεδομένων.

```
17  struct dataBlocks{
18      int ind;
19      string keyword;
20      OP op;
21      int pth;
22  };
```

Εικόνα 3.4 Structure datablocks

Το structure Block (εικόνα 3.5) "δένει" τα δεδομένα του dataBlocks για να μπορέσουν να εισαχθούν σε ένα array (εικόνα 3.6) το οποίο έχει χωρητικότητα $Z = 4$ και θεωρείται ο "κουβάς" (bucket) από μπλοκ δεδομένων που αποθηκεύονται στους κόμβους του δέντρου.

```
24  struct Block {
25      dataBlocks data;
26  };
```

Εικόνα 3.5 Structure Block

```
28   using Bucket = std::array<Block, Z>;
```

Εικόνα 3.6 "Κουβάς" Bucket

Η κλάση Node (εικόνα 3.7) δεν έχει κάποια λειτουργικότητα πέρα από την αποθήκευση των κρυπτογραφημένων κόμβων του δέντρου σε έναν πίνακα, με τον κόμβο ρίζα να βρίσκεται στη θέση μηδέν του πίνακα, τον κόμβο ένα στη θέση ένα κ.ο.κ.

```
30  class Node{
31  public:
32      Node(){}
33      ~Node(){}
34      block value;
35  };
```

Εικόνα 3.7 Κλάση Node

3.2 Η Κλάση AES

Η κλάση AES είναι υπεύθυνη για την κρυπτογράφηση των κόμβων του δέντρου καθώς επίσης για τον υπολογισμό του μονοπατιού στο οποίο αποθηκεύονται τα δεδομένα. Η χρήση της κλάσης δεν απαιτεί τη δημιουργία αντικειμένου καθώς όλες οι μέθοδοι είναι static.

3.2.1 Κρυπτογράφηση του Δέντρου

Για την κρυπτογράφηση του δέντρου γίνεται η χρήση της βιβλιοθήκης OpenSSL. Οι μέθοδοι Setup και Cleanup (εικόνα 3.8) δεσμεύουν και αποδεσμεύουν αντίστοιχα τους απαραίτητους πόρους για τη χρήση της βιβλιοθήκης. Η μέθοδος Setup καλείται στον κατασκευαστή της κλάσης SORAM και η Cleanup με την ολοκλήρωση της εκτέλεσης του προγράμματος.

```
11  void AES::Setup(){
12      // Initialise OpenSSL
13      ERR_load_crypto_strings();
14      OpenSSL_add_all_algorithms();
15  }
16
17  void AES::Cleanup(){
18      EVP_cleanup();
19      ERR_free_strings();
20  }
```

Εικόνα 3.8 AES Setup και Cleanup

Η μέθοδος Encrypt (εικόνα 3.9) δέχεται το μυστικό κλειδί k_{SO} και το μήνυμα που αποτελείται από το περιεχόμενο του εκάστοτε κόμβου στη δυαδική του αναπαράσταση. Η μέθοδος υπολογίζει ένα τυχαίο IV μήκους 16 και καλεί την EncryptBlock (εικόνα 3.11). Το IV υπολογίζεται, όπως φαίνεται στην εικόνα 3.10, με τη βοήθεια της συνάρτησης RAND_bytes από τη βιβλιοθήκη OpenSSL, η οποία παράγει τυχαία bytes. Η EncryptBlock δημιουργεί μία μεταβλητή στην οποία θα αποθηκεύσει το κρυπτοκείμενο και στη συνέχεια καλεί την EncryptBytes (εικόνα 3.12). Η EncryptBytes δέχεται ως παραμέτρους το μυστικό κλειδί k_{SO}, το IV, το μήνυμα που θα κρυπτογραφήσει και ένα δείκτη στον οποίο θα αποθηκεύσει το κρυπτοκείμενο.

Η μέθοδος χρησιμοποιώντας τη βιβλιοθήκη EVP από το OpenSSL, η οποία παρέχει μία υψηλού επιπέδου διεπαφή για τη χρήση κρυπτογραφικών συναρτήσεων, κρυπτογραφεί το μήνυμα με τη χρήση του AES_CBC_256 όπως φαίνεται στη γραμμή 34 της εικόνας 3.12.

```
102  block AES::Encrypt(bytes<Key> kSO, block plaintext,size_t clen_size,size_t plaintext_size){
103      block ciphertext;
104      bytes<IV> iv = AES::GenerateIV();
105
106      ciphertext = Encryptblock(kSO, iv, plaintext,clen_size,plaintext_size);
107
108      // Put randomised IV at the front of the ciphertext
109      ciphertext.insert(ciphertext.end(), iv.begin(), iv.end());
110      return ciphertext;
111  }
```

Εικόνα 3.9 AES Encrypt

```
152   bytes<IV> AES::GenerateIV(){
153       bytes<IV> iv;
154
155       if (RAND_bytes(iv.data(), iv.size()) != 1){
156           // Bytes generated aren't cryptographically strong
157           error("Needs more entropy");
158       }
159
160       return iv;
161   }
```

Εικόνα 3.10 AES GenerateIV

```
86  block AES::Encryptblock(bytes<Key> kSO, bytes<IV> iv, block plaintext,size_t clen_size,size_t plaintext_size){
87      block ciphertext(clen_size);
88      EncryptBytes(kSO, iv, plaintext.data(), plaintext_size, ciphertext.data());
89      return ciphertext;
90  }
```

Εικόνα 3.11 AES EncryptBlock

```cpp
int AES::EncryptBytes(bytes<Key> kSO, bytes<IV> iv, byte_t *plaintext, int plen, byte_t *ciphertext){
    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();

    if (!ctx){
        error("Failed to create new cipher");
    }

    // Initialise the encryption operation
    if (EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, kSO.data(), iv.data()) != 1){
        error("Failed to initialise encryption");
    }

    // Encrypt
    int len;
    if (EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plen) != 1){
        error("Failed to complete EncryptUpdate");
    }

    int clen = len;

    if (EVP_EncryptFinal_ex(ctx, ciphertext + len, &len) != 1){
        error("Failed to complete EncryptFinal");
    }
    clen += len;

    EVP_CIPHER_CTX_free(ctx);

    return clen;
}
```

Εικόνα 3.12 AES EncryptBytes

Για την αποκρυπτογράφηση των κόμβων του δέντρου εκτελείται η αντίστροφη διαδικασία. Δηλαδή για έναν κόμβο καλείται η μέθοδος Decrypt (εικόνα 3.13) η οποία διαχωρίζει το IV από το κρυπτοκείμενο και καλείται η DecryptBlock (εικόνα 3.14). Δημιουργεί μία μεταβλητή στην οποία θα αποθηκευτεί το αρχικό μήνυμα και καλεί την DecryptBytes (εικόνα 3.15). Η DecryptBytes χρησιμοποιώντας τη διεπαφή EVP αποκρυπτογραφεί το μήνυμα και το επιστρέφει.

```cpp
block AES::Decrypt(bytes<Key> key, block ciphertext,size_t clen_size){
    // Extract the IV
    bytes<IV> iv;
    std::copy(ciphertext.end() - IV, ciphertext.end(), iv.begin());

    // Perform the decryption
    block plaintext = Decryptblock(key, iv, ciphertext,clen_size);

    return plaintext;
}
```

Εικόνα 3.13 AES Decrypt

```
92  block AES::Decryptblock(bytes<Key> kSO, bytes<IV> iv, block ciphertext,size_t clen_size){
93      block plaintext(clen_size);
94      int plen = DecryptBytes(kSO, iv, ciphertext.data(), clen_size, plaintext.data());
95
96      // Trim plaintext to actual size
97      plaintext.resize(plen);
98
99      return plaintext;
100 }
```

Εικόνα 3.14 AES DecryptBlock

```
56  int AES::DecryptBytes(bytes<Key> kSO, bytes<IV> iv, byte_t *ciphertext, int clen, byte_t *plaintext){
57      EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
58
59      if (!ctx){
60          error("Failed to create new cipher");
61      }
62
63      // Initialise the decryption operation
64      if (EVP_DecryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, kSO.data(), iv.data()) != 1){
65          error("Failed to initialise decryption");
66      }
67
68      // Dencrypt
69      int len;
70      if (EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext, clen) != 1){
71          error("Failed to complete DecryptUpdate");
72      }
73
74      int plen = len;
75
76      if (EVP_DecryptFinal_ex(ctx, plaintext + len, &len) != 1){
77          error("Failed to complete DecryptFinal");
78      }
79      plen += len;
80
81      EVP_CIPHER_CTX_free(ctx);
82
83      return plen;
84  }
```

Εικόνα 3.15 AES DecryptBytes

3.2.2 Υπολογισμός Μονοπατιού

Ο υπολογισμό του μονοπατιού στο οποίο θα αποθηκευτούν τα μπλοκ δεδομένων γίνεται σε δύο στάδια. Πρώτα ο πελάτης καλεί τη μέθοδο ComputeToken (εικόνα 3.16) με παραμέτρους το μυστικό κλειδί k_h, το μέγεθος του κλειδιού, το μήνυμα καθώς και ένα δείκτη στον οποίο θα αποθηκευτεί το token που θα παραχθεί. Η μέθοδος χρησιμοποιώντας τον αλγορίθμου Blake2b από τη βιβλιοθήκη Libsodium παράγει ένα message digest μήκους 32 το οποίο αποθηκεύεται στη μεταβλητή token.

```
129 // Compute a token with Blake2b based on key and specified input
130 void AES::ComputeToken(unsigned char* key, unsigned int key_len, char *plaintext, unsigned char* token){
131     unsigned int md_len = 32;
132     crypto_generichash_blake2b_state state;
133 
134     crypto_generichash_blake2b_init(&state, key, strlen((char *)key), md_len);
135     crypto_generichash_blake2b_update(&state, (unsigned char *)&plaintext[0], strlen(plaintext));
136     crypto_generichash_blake2b_final(&state, token, md_len);
137 }
```

Εικόνα 3.16 Μέθοδος ComputeToken

Στο δεύτερο στάδιο, ο πελάτης καλεί την μέθοδο ComputePath (εικόνα 3.17) με παραμέτρους το token που δημιουργήθηκε στο προηγούμενο στάδιο και το μήνυμα. Το token χρησιμοποιείται ως κλειδί για τον αλγόριθμο Blake2b. Από το παραγόμενο message digest, μήκους 32, επιστρέφουμε στον πελάτη την ακέραια αναπαράσταση του πρώτου bit. Επειδή όμως η τιμή αυτή είναι συχνά πολύ μεγάλη με συνέπεια να ξεπερνάει το εύρος τιμών ακόμα και μιας μεταβλητής τύπου unsigned long, τη διαιρούμε πρώτα με το 100. Ο πελάτης χρησιμοποιεί αυτόν τον ακέραιο για να υπολογίσει το αναγνωριστικό ενός φύλλου από το δέντρο και κατ' επέκταση το μονοπάτι.

```
139 // Return the integer represantation of the first bit of the digest
140 unsigned long AES::ComputePath(unsigned char* key, char *plaintext){
141     unsigned int md_len = 32;
142     unsigned char digest[sizeof(char) * md_len];
143     crypto_generichash_blake2b_state state;
144 
145     crypto_generichash_blake2b_init(&state, key, sizeof(key), md_len);
146     crypto_generichash_blake2b_update(&state, (unsigned char *)&plaintext[0], strlen(plaintext));
147     crypto_generichash_blake2b_final(&state, digest, md_len);
148 
149     return *(unsigned long *)&digest[0] / 100;
150 }
```

Εικόνα 3.17 Μέθοδος ComputePath

3.3 Η κλάση BinaryTree

Η κλάση BinaryTree υποστηρίζει τις εξής τρεις μεθόδους.

Η μέθοδος extraxtNodes(J), που φαίνεται στην εικόνα 3.18, δέχεται ως παράμετρο ένα σύνολο J που αποτελείται από αναγνωριστικά φύλλων του δέντρου. Δημιουργεί ένα map με κλειδί το αναγνωριστικό των κόμβων που ανήκουν στα μονοπάτια από τα φύλλα αυτά μέχρι τη ρίζα και τιμή το κρυπτογραφημένο περιεχόμενο των κόμβων αυτών. Επειδή ο κάθε κόμβος εισέρχεται μόνο μία φορά στο map, όταν βρεθεί ένας κόμβος που έχει εισαχθεί τότε η επανάληψη στην γραμμή 44 σταματάει καθώς όλοι οι κόμβοι πατέρας θα έχουν εισαχθεί επίσης. Στη συνέχεια εκτελείται η ίδια επανάληψη για όλα τα μονοπάτια που βρίσκονται στο J.

```
38  // extract all nodes that belong in the J
39  encrypted_nodes BinaryTree::extractNodes(vector<int> J){
40      encrypted_nodes map;
41
42      for(auto it = J.begin(); it != J.end(); it++){
43          int n = *it;
44          while(n >= 0){
45              auto notInserted = map.insert(make_pair(n, soram->ReadBlock(n)));
46              if(notInserted.second == true){
47                  if(n % 2 == 0){ // right child
48                      n = n / 2 - 1;
49                  }else{ // left child
50                      n = n / 2;
51                  }
52              }else{ // node already inserted
53                  break;
54              }
55          }
56      }
57      return map;
58  }
```

Εικόνα 3.18 Μέθοδος extractNodes

Η μέθοδος extractRandNodes(nb), που βλέπουμε στην εικόνα 3.19, είναι παρόμοια με την extractNodes(J).

Η διαφορά είναι πως η extractRandNodes(nb) δημιουργεί ένα map από nb τυχαία επιλεγμένα μονοπάτια. Πιο αναλυτικά, στη γραμμή 19 ζητάμε από τον SORAM να μας δώσει ένα τυχαίο φύλλο του δέντρου και η μέθοδος εισάγει στο map τους κρυπτογραφημένους κόμβους που ανήκουν στο αντίστοιχο μονοπάτι. Η διαδικασία επαναλαμβάνεται έως ότου εισαχθούν οι κόμβοι από ακριβώς nb μονοπάτια.

```
15  // extract the unique nodes from nb randomly selected tree paths
16  encrypted_nodes BinaryTree::extractRandNodes(int nb){
17      encrypted_nodes map;
18      while(nb > 0){
19          int leaf = soram->RandomPath();
20          int nodeID = leaf;
21          while(nodeID >= 0){
22              auto notInserted = map.insert(make_pair(nodeID, soram->ReadBlock(nodeID)));
23              if(notInserted.second == true){
24                  if(leaf == nodeID) nb--;
25                  if(nodeID % 2 == 0){
26                      nodeID = nodeID / 2 - 1;
27                  }else{
28                      nodeID = nodeID / 2;
29                  }
30              }else{ // node already inserted
31                  break;
32              }
33          }
34      }
35      return map;
36  }
```

Εικόνα 3.19 Μέθοδος extractRandNodes

Τέλος η συνάρτηση insertNodes(nodes), όπως φαίνεται στην εικόνα 3.20, δέχεται ως παράμετρο ένα map που έχει ως κλειδί το αναγνωριστικό των κόμβων και ως τιμή το κρυπτογραφημένο περιεχόμενό τους. Η μέθοδος αντικαθιστά το παλιό περιεχόμενο των κόμβων αυτών με το καινούριο.

```
60  // insert the nodes in the tree
61  // replacing their previous content
62  void BinaryTree::insertNodes(encrypted_nodes nodes){
63      for(auto it = nodes.begin(); it != nodes.end(); it++){
64          soram->setNode(it->first, it->second);
65      }
66  }
```

Εικόνα 3.20 Μέθοδος insertNodes

3.4 Η Κλάση SORAM

Η κλάση SORAM χρησιμοποιείται τόσο από τον πελάτη όσο και από τον διακομιστή. Οι ενέργειες, ωστόσο, που εκτελούνται διαφέρουν ανάμεσα στους δύο.

Ο πελάτης δημιουργεί ένα αντικείμενο της κλάσης SORAM και αρχικοποιεί το δυαδικό δέντρο με “dummy” δεδομένα, όπως περιγράφεται στην παράγραφο 4.2.1.

Χρησιμοποιώντας τη μέθοδο SerialiseBucket (εικόνα 3.21) ο πελάτης μετατρέπει ένα αντικείμενο Bucket σε block, δηλαδή στη δυαδική του αναπαράσταση, για να μπορέσει να κρυπτογραφηθεί στη συνέχεια. Αντίθετα η μέθοδος DeserialiseBucket (εικόνα 3.22) δέχεται ένα block και το μετατρέπει σε Bucket.

```
35 block SORAM::SerialiseBucket(Bucket bucket) {
36     block buffer;
37 
38     for (int z = 0; z < Z; z++) {
39         Block b = bucket[z];
40         string plain = to_string(b.data.ind) + ":" + to_string(b.data.op) + ":" + to_string(b.data.pth) + ":" + b.data.keyword;
41 
42         uint8_t* p = reinterpret_cast<uint8_t*>(&plain[0]);
43         auto r = vector<byte_t>(p, p + sizeof(plain));
44 
45         buffer.insert(buffer.end(), r.begin(), r.end());
46     }
47     assert(buffer.size() == Z * (blockSize));
48 
49     return buffer;
50 }
```

Εικόνα 3.21 Μέθοδος SerialiseBucket

```
52  Bucket SORAM::DeserialiseBucket(block buffer) {
53      assert(buffer.size() == Z * (blockSize));
54
55      Bucket bucket;
56      block b;
57      for (int z = 0; z < Z; z++) {
58          string plain;
59          b.assign(buffer.begin(), buffer.begin() + blockSize);
60          for(auto i = b.begin(); i != b.end(); i++){
61              plain += *i;
62          }
63          bucket[z].data.ind = stoi(plain.substr(0, plain.find(":")));
64          plain.erase(0, plain.find(":") + sizeof(char));
65
66          bucket[z].data.op = (OP)stoi(plain.substr(0, plain.find(":")));
67          plain.erase(0, plain.find(":") + sizeof(char));
68
69          bucket[z].data.pth = stoi(plain.substr(0, plain.find(":")));
70          plain.erase(0, plain.find(":") + sizeof(char));
71
72          bucket[z].data.keyword = &plain[0];
73
74          buffer.erase(buffer.begin(), buffer.begin() + blockSize);
75      }
76
77      return bucket;
78  }
```

Εικόνα 3.22 Μέθοδος DeserialiseBucket

Η μέθοδος EncryptNode (εικόνα 3.23) δέχεται ένα Bucket και το μυστικό κλειδί k_{SO}, το μετατρέπει σε block χρησιμοποιώντας τη μέθοδο SerialiseBucket και το κρυπτογραφεί με την Encrypt όπως περιγράφηκε στην παράγραφο 3.2.1. Για την ανάγνωση ενός Bucket, η μέθοδος ReadBucket (εικόνα 3.24) αποκρυπτογραφεί με την Decrypt (παράγραφος 3.2.1), και μετατρέπει το block που επιστρέφεται σε bucket με την DeserialiseBucket.

```
88  block SORAM::EncryptNode(Bucket bucket, bytes<Key> kSO){
89      block b = SerialiseBucket(bucket);
90      block ciphertext = AES::Encrypt(kSO, b, clen_size, plaintext_size);
91      return ciphertext;
92  }
```

Εικόνα 3.23 Μέθοδος EncryptNode

```
100  Bucket SORAM::ReadBucket(block ciphertext, bytes<Key> kSO) {
101      block buffer = AES::Decrypt(kSO, ciphertext, clen_size);
102      Bucket bucket = DeserialiseBucket(buffer);
103      return bucket;
104  }
```

Εικόνα 3.24 Μέθοδος ReadBucket

Ο διακομιστής αντίθετα, δεν εκτελεί αυτές τις λειτουργίες. Ο διακομιστής χρησιμοποιεί το αντικείμενο της κλάσης SORAM μόνο για να διαβάζει με τη μέθοδο ReadBlock (εικόνα 3.25) και να γράφει με τη μέθοδο setNode (εικόνα 3.26) έναν κρυπτογραφημένο κόμβο από και προς το δυαδικό δέντρο.

```
106  block SORAM::ReadBlock(int n){
107      return nodes[n].value;
108  }
```

Εικόνα 3.25 Μέθοδος ReadBlock

```
110  void SORAM::setNode(int i, block value){
111      nodes[i].value = value;
112  }
```

Εικόνα 3.26 Μέθοδος SetNode

4. Mara DSSE Scheme

4.1 Οι Αλγόριθμοι

Σε αυτήν την ενότητα περιγράφονται αναλυτικά οι αλγόριθμοι του σχήματος Mara (Setup, Update, Search). Ο αλγόριθμος Setup είναι υπεύθυνος για την αρχικοποίηση των απαραίτητων μεταβλητών και του δυαδικού δέντρου. Ο αλγόριθμος Update εισάγει δεδομένα στο δέντρο και ο Search διαγράφει τα δεδομένα που ικανοποιούν τη συνθήκη αναζήτησης.

4.1.1 Setup

Για την αρχικοποίηση του σχήματος (εικόνα 4.1) έχουμε ένα αντικείμενο Client και ένα αντικείμενο Server. Το αντικείμενο Client κρατάει ένα δείκτη στη διεύθυνση μνήμης του αντικειμένου Server καθώς η εκτέλεση του προγράμματος γίνεται στη μνήμη RAM. Η παράμετρος N είναι το μέγεθος που θα έχει το δυαδικό δέντρο.

Ο Client δημιουργεί το δυαδικό δέντρο (παράγραφος 4.2.1), το οποίο με τη σειρά του δημιουργεί ένα αντικείμενο της κλάσης SORAM, τα οποία στέλνονται και στο διακομιστή (αντικείμενο Server). Ακόμη δημιουργεί το λεξικό D_x και τη stash και τα κλειδιά k_{SO} και k_h (εικόνα 4.2). Τέλος, αποθηκεύει μία μεταβλητή με το συνολικό αριθμό φύλλων και μία με το αναγνωριστικό του πρώτου φύλλου.

```
4  Client::Client(Server* server, int N){
5      this->server = server;
6
7      // Initialize a Dictionary Dx and the local stash
8      Dx.clear();
9      stash.clear();
10
11     // Create and send BinaryTree to the Server
12     this->bt = BinaryTree(N, kSO);
13     this->server->setBinaryTree(&bt);
14     this->soram = bt.getSORAM();
15
16     // get first leafID and total number of leaves
17     this->firstLeafID = soram->getFirstLeaf();
18     this->totalLeaves = soram->getNumberOfLeaves();
19 }
```

Εικόνα 4.1 Client-Server Setup

```
18         bytes<Key> kSO {{0}}; // SORAM Key
19         unsigned char kh[Blake2Key] {1}; // PRF Key(Blake2)
20
21         unordered_map<string, pair<int, int>> Dx; //Dictionary
22         map<string, dataBlocks> stash;
```

Εικόνα 4.2 Client Header File

4.1.2 Update

Ο αλγόριθμος για την ενημέρωση (εικόνα 4.3) παίρνει ως παράμετρο το σύνολο των δεδομένων που πρόκειται να εισαχθούν ή να διαγραφούν. Τα δεδομένα είναι της μορφής (ind||ω||op||pth), όπου ind το αναγνωριστικό του αρχείου, ω η λέξη/κλειδί, op ο τύπος της ενημέρωσης, ο οποίος παίρνει την τιμή ένα για την εισαγωγή και την τιμή μηδέν για τη διαγραφή. Το pth αρχικά έχει μία “dummy” τιμή και συμβολίζει το μονοπάτι στο δέντρο που θα αποθηκευτεί η κάθε τετράδα. Παίρνει, δηλαδή, ως τιμή το αναγνωριστικό ενός φύλλου από το δυαδικό δέντρο. Ο πελάτης υπολογίζει την τιμή για το pth και οι ολοκληρωμένες τετράδες αποτελούν την παράμετρο για τον insert αλγόριθμο του SORAM (παράγραφος 4.2.2).

Για τον υπολογισμό του pth γίνεται χρήση των μεθόδων ComputeToken και ComputePath, όπως περιγράφονται αναλυτικότερα στην υποενότητα 3.2.2.

```
27  void Client::update(map<string, dataBlocks> Data){
28      for(auto i = Data.begin();i != Data.end();i++){
29          int len = 32;
30          unsigned char token[sizeof(char) * len];
31          string input = i->second.keyword + to_string(Dx[i->second.keyword].first);
32          char val[input.length()];
33          strcpy(val, input.c_str());
34          AES::ComputeToken(kh, sizeof(kh), val, token);
35
36          Dx[i->second.keyword].second += 1;
37
38          input = i->second.keyword + to_string(Dx[i->second.keyword].second);
39          char val1[input.length()];
40          strcpy(val1, input.c_str());
41          unsigned long pth = AES::ComputePath(token, val1);
42
43          i->second.pth = pth % totalLeaves + firstLeafID;
44
45          Data[i->first] = {i->second.ind, i->second.keyword, i->second.op, i->second.pth};
46      }
47      insert(Data);
48  }
```

Εικόνα 4.3 Client Update

Πρώτα καλείται η μέθοδος ComputeToken η οποία παίρνει σαν παράμετρο το (ω||D_x[ω][0]), όπου ω η λέξη/κλειδί και D_x[ω][0] ο συνολικός αριθμός αναζητήσεων του ω. Έπειτα καλείται η ComputePath που δέχεται ως παραμέτρους το token που παράχθηκε στο προηγούμενο βήμα και το (ω||D_x[ω][1]) με αυξημένο το D_x[ω][1] κατά ένα, όπου D_x[ω][1] ο συνολικός αριθμός ενημερώσεων για το ω. Ο ακέραιος που επιστρέφεται στη γραμμή 41 από την μέθοδο ComputePath επειδή είναι πολύ μεγάλος τον διαιρούμε με τον συνολικό αριθμό των φύλλων του δέντρου και στο υπόλοιπο της διαίρεσης προσθέτουμε το αναγνωριστικό του πρώτου φύλλου. Έτσι, εξασφαλίζουμε ότι η μεταβλητή pth θα έχει σίγουρα την τιμή κάποιου φύλλου από το δέντρο.

Στην συνέχεια ο πελάτης αλληλεπιδρά με το διακομιστή μέσω του SORAM αλγορίθμου, όπως περιγράφεται στην παράγραφο 4.2.2, για την αποθήκευση των δεδομένων στο δυαδικό δέντρο.

4.1.3 Search

Για την αναζήτηση (εικόνα 4.4) ο πελάτης αλληλεπιδρά με το διακομιστή και εξάγει με τη βοήθεια του αλγορίθμου SORAM, όπως περιγράφεται στην παράγραφο 4.2.3, όλα τα μπλοκ δεδομένων που περιέχουν τη λέξη/κλειδί ω, αυξάνει το μετρητή αναζήτησης για το ω κατά ένα και μηδενίζει το μετρητή ενημέρωσης.

```
87  void Client::search(string keyword){
88      remove(keyword);
89      Dx[keyword].first += 1;
90      Dx[keyword].second = 0;
91  }
```

Εικόνα 4.4 Client Search

4.2 Ο Αλγόριθμος SORAM

Σε αυτήν την ενότητα παρουσιάζεται ο αλγόριθμος SORAM.

4.2.1 Setup

Ο κατασκευαστής της κλάσης SORAM αρχικοποιεί το δυαδικό δέντρο, όπως φαίνεται στην εικόνα 4.6, ύψους $h = \log_2 N$. Κάθε κόμβος του δέντρου είναι ένας “κουβάς” (bucket) χωρητικότητας $Z = 4$ μπλοκ δεδομένων. Κάθε κόμβος αρχικοποιείται με “dummy” δεδομένα.

Η μέθοδος WriteBucket (εικόνα 4.5) είναι παρόμοια με την EncryptNode, που περιγράφηκε στην παράγραφο 3.4, με τη διαφορά ότι καλείται μία φορά και εισάγει τους κόμβους στο δέντρο, το οποίο στέλνετε στο διακομιστή από τον πελάτη.

Τέλος αποθηκεύονται σε μεταβλητές ο αριθμός των φύλλων του δέντρου, ποιο φύλλο είναι πρώτο και αρχικοποιείται η μεταβλητή dis η οποία είναι υπεύθυνη για την παραγωγή ενός τυχαίου αριθμού στο εύρος [firstLeaf, lastLeaf].

```
94  void SORAM::WriteBucket(int index, Bucket bucket, bytes<Key> kSO){
95      block b = SerialiseBucket(bucket);
96      block ciphertext = AES::Encrypt(kSO, b, clen_size, plaintext_size);
97      nodes[index].value = ciphertext;
98  }
```

Εικόνα 4.5 SORAM WriteBucket

```
7   SORAM::SORAM(int N, bytes<Key> kSO)
8   : rd(), mt(rd()){
9       AES::Setup();
10      depth = ceil(log2(N));
11      bucketCount = pow(2, depth + 1) - 1;
12      blockSize = sizeof(string);
13      nOfLeaves = bucketCount / 2 + 1;
14      firstLeaf = bucketCount - nOfLeaves;
15      nodes = new Node[bucketCount];
16      dis = std::uniform_int_distribution<int>{firstLeaf, bucketCount - 1};
17      clen_size = AES::GetCiphertextLength(blockSize * Z);
18      plaintext_size = blockSize * Z;
19      for(int i = 0; i < bucketCount; i++){
20          Bucket bucket;
21          for(int z = 0; z < Z; z++){
22              bucket[z].data.ind = -1;
23              bucket[z].data.keyword = "dummydata";
24              bucket[z].data.op = OP::INS;
25              bucket[z].data.pth = -1;
26          }
27          WriteBucket(i, bucket, kSO);
28      }
29  }
```

Εικόνα 4.6 SORAM setup

4.2.2 Insert

Ο αλγόριθμος εισαγωγής (εικόνα 4.7) παίρνει ως παράμετρο ένα σύνολο δεδομένων B και τα αποθηκεύει στο δυαδικό δέντρο της SORAM δομής. Ακόμη αφαιρεί από τη stash τα μπλοκ που ενδεχομένως βρίσκονται ακόμη εκεί και τα αποθηκεύει στο δέντρο.

Πιο αναλυτικά ο πελάτης ζητάει από το διακομιστή $n_b = |B|$ τυχαία μονοπάτια και του επιστρέφονται οι κόμβοι που αντιστοιχούν σε αυτά. Αυτό επιτυγχάνεται με τη μέθοδο ExtractRandNodes όπως αναλύθηκε στην παράγραφο 3.3. Οι κόμβοι αποκρυπτογραφούνται με το κλειδί k_{SO} και δημιουργούνται, ένα σύνολο J που κρατούνται τα αναγνωριστικά των κόμβων και ένα σύνολο S που κρατούνται τα έγκυρα δεδομένα (δηλαδή δεν λαμβάνονται υπόψη τα “dummy” μπλοκ)

που υπάρχουν στους κόμβους και στην stash. Στη συνέχεια καλείται η βοηθητική συνάρτηση DelRedund, η οποία εξαλείφει δεδομένα τα οποία έχουν το ίδιο ω και το ίδιο ind αλλά διαφορετικό op. Η συνάρτηση Evict, η οποία καλείται αμέσως μετά, εισάγει τα δεδομένα στους κόμβους και τα δεδομένα που δεν μπόρεσαν να εισαχθούν αποθηκεύονται στην stash.

Οι συναρτήσεις DelRedund και Evict αναλύονται στις παραγράφους 4.3.1 και 4.3.2 αντίστοιχα. Ακολουθεί η κρυπτογράφηση των πλέον ενημερωμένων κόμβων με το μυστικό κλειδί k_{SO} και τέλος η αποστολή και αποθήκευση των κόμβων στο διακομιστή.

```
48  void Client::insert(map<string, dataBlocks> S){
49      decrypted_nodes dec_nodes;
50      encrypted_nodes enc_nodes = server->getNodesOnRandomPaths(S.size());
51      // Decrypt nodes
52      for(auto itr = enc_nodes.begin(); itr != enc_nodes.end(); itr++){
53          dec_nodes.insert(make_pair(itr->first, soram->ReadBucket(itr->second, kSO)));
54      }
55      vector<int> J;
56
57      for(auto it = dec_nodes.begin(); it != dec_nodes.end(); it++){
58          J.push_back(it->first);
59          for(int z = 0; z < Z; z++){
60              if(it->second[z].data.ind != -1){ // It isn't a dummy block
61                  auto tmp = it->second[z].data;
62                  S[to_string(tmp.op) + to_string(tmp.ind) + tmp.keyword] = tmp;
63              }
64          }
65      }
66      enc_nodes.clear();
67      dec_nodes.clear();
68
69      // Inserts in S the data blocks from the stash (if any)
70      for(auto i = stash.begin();i != stash.end();i++){
71          S[i->first] = i->second;
72      }
73      stash.clear();
74
75      S = Auxiliary::DelRedund(S);
76
77      // Eviction algorithm
78      dec_nodes = Auxiliary::Evict(S, J, &stash);
79
80      // Encrypt nodes
81      for(auto it = dec_nodes.begin(); it != dec_nodes.end(); it++){
82          enc_nodes.insert(make_pair(it->first, soram->EncryptNode(it->second, kSO)));
83      }
84      server->insert(enc_nodes);
85  }
```

Εικόνα 4.7 Μέθοδος SORAM Insert

4.2.3 Delete

Ο αλγόριθμος διαγραφής (εικόνα 4.8, η μέθοδος ονομάζεται remove διότι η λέξη delete είναι δεσμευμένη στη C++) παίρνει ως παράμετρο μία λέξη/κλειδί ω και αφαιρεί όλα τα δεδομένα που περιέχουν το ω. Ακόμη εισάγει στο δέντρο τα δεδομένα που ενδεχομένως βρίσκονται στη stash.

Ο πελάτης υπολογίζει με τη μέθοδο ComputeToken από την κλάση AES (παράγραφος 3.2.2) το token που προκύπτει από το μυστικό κλειδί k_h και είσοδο (ω||D_x[ω][0]), όπου D_x[ω][0] ο συνολικός αριθμός αναζητήσεων για το ω. Στέλνει στο διακομιστή το token, το ω και το D_x[ω][1], δηλαδή τον αριθμό των μπλοκ δεδομένων που σχετίζονται με το ω που βρίσκονται επί του παρόντος στο δέντρο.

Ο διακομιστής (εικόνα 4.9) καλεί την μέθοδο ComputePath (παράγραφος 3.2.2) επαναληπτικά με σταθερό κλειδί το token και ως είσοδο το (ω||i), όπου i = 1 έως D_x[ω][1] και υπολογίζει τα μονοπάτια με τη χρήση της μεθόδου extractNodes (παράγραφος 3.3) στα οποία υπάρχουν τα δεδομένα που περιέχουν το ω και στέλνει στον πελάτη ένα σύνολο το οποίο περιέχει τα αναγνωριστικά των κόμβων που βρίσκονται στα μονοπάτια καθώς και τους κόμβους. Η γραμμή 30 στην εικόνα 3.9 ακολουθεί την ίδια λογική που περιγράφηκε στην παράγραφο 4.1.2 για τον υπολογισμό έγκυρου αναγνωριστικού για τα φύλλα του δέντρου.

Στη συνέχεια ο πελάτης αποκρυπτογραφεί τους κόμβους με το μυστικό κλειδί k_{SO} και δημιουργεί το σύνολο J με τα αναγνωριστικά των κόμβων και το σύνολο S με το έγκυρο περιεχόμενο των κόμβων και της stash εάν δεν είναι άδεια. Καλεί τη συνάρτηση DelRedund (παράγραφος 4.3.1), όπου μετά την εκτέλεσή της θα υπάρχει το πολύ ένα μπλοκ δεδομένων για κάθε ζευγάρι (ind, ω). Τα μπλοκ τα οποία περιέχουν το ω αφαιρούνται από το S και καλείται η συνάρτηση Evict (παράγραφος 4.3.2) για την αποθήκευση των υπόλοιπων μπλοκ στο δέντρο και στη stash όσα δεν μπόρεσαν να εισαχθούν. Οι κόμβοι του δέντρου κρυπτογραφούνται και στέλνονται στο διακομιστή για να αντικαταστήσει το περιεχόμενο των αντίστοιχων κόμβων με το καινούριο.

```
void Client::remove(string keyword){
    map<string, dataBlocks> S;
    vector<int> J;
    int cUpd = Dx[keyword].second;
    int len = 32;
    unsigned char token[sizeof(char) * len];
    string input = keyword + to_string(Dx[keyword].first);
    char val[input.length()];
    strcpy(val, input.c_str());
    AES::ComputeToken(kh, sizeof(kh), val, token);

    // Decrypt nodes
    decrypted_nodes dec_nodes;
    encrypted_nodes enc_nodes = server->remove(keyword, token, cUpd);
    for(auto itr = enc_nodes.begin(); itr != enc_nodes.end(); itr++){
        dec_nodes.insert(make_pair(itr->first, soram->ReadBucket(itr->second, kSO)));
    }

    // insert dataBlocks in S
    for(auto it = dec_nodes.begin(); it != dec_nodes.end(); it++){
        J.push_back(it->first);
        for(int z = 0; z < Z; z++){
            if(it->second[z].data.ind != -1){ // It isn't a dummy block
                auto tmp = it->second[z].data;
                S[to_string(tmp.op) + to_string(tmp.ind) + tmp.keyword] = tmp;
            }
        }
    }
    enc_nodes.clear();
    dec_nodes.clear();

    // Inserts in S the data blocks from the stash (if any)
    for(auto i = stash.begin();i != stash.end();i++){
        S[i->first] = i->second;
    }
    stash.clear();

    S = Auxiliary::DelRedund(S);
    // remove from S the dataBlocks that contain keyword w
    vector<string> found;
    for(auto block = S.begin(); block != S.end();block++){
        if(keyword.compare(block->second.keyword) == 0){
            found.push_back(block->first);
        }
    }
    for(auto block = found.begin(); block != found.end(); block++){
        S.erase(*block);
    }

    // Eviction algorithm
    dec_nodes = Auxiliary::Evict(S, J, &stash);

    // Encrypt nodes
    for(auto it = dec_nodes.begin(); it != dec_nodes.end(); it++){
        enc_nodes.insert(make_pair(it->first, soram->EncryptNode(it->second, kSO)));
    }
    server->insert(enc_nodes);
}
```

Εικόνα 4.8 Μέθοδος SORAM Delete

```
encrypted_nodes Server::remove(string keyword, unsigned char token[], int cUpd){
    vector<int> J;
    int numberOfLeaves = bt->getNumberOfLeaves();
    int firstLeafID = bt->getFirstLeaf();
    for(int i=1; i <= cUpd; i++){
        string input = keyword + to_string(i);
        char val1[input.length()];
        strcpy(val1, input.c_str());
        int pth = AES::ComputePath(token, val1);
        pth = pth % numberOfLeaves + firstLeafID;
        J.push_back(pth);
    }
    return bt->extractNodes(J);
}
```

Εικόνα 4.9 Μέθοδος Server Remove

4.3 Βοηθητικές Συναρτήσεις

Σε αυτήν την ενότητα περιγράφονται οι βοηθητικές συναρτήσεις που χρησιμοποιούνται από τον αλγόριθμο SORAM.

4.3.1 DelRedund

Η συνάρτηση DelRedund (εικόνα 4.10) παίρνει ως όρισμα ένα σύνολο S από μπλοκ δεδομένων (ind||ω||op||pth) και διαγράφει εκείνα που έχουν τις ίδιες τιμές για τα (ind||ω) αλλά έχουν διαφορετική τιμή στο op. Για παράδειγμα, αν εμφανιστούν τα μπλοκ (α||β||1||pth) και (α||β||0||pth) θα αφαιρεθούν και τα δύο από το S.

```
map<string, dataBlocks> Auxiliary::DelRedund(map<string, dataBlocks> S){
    vector<dataBlocks> tempS;
    map<string, int> dictionary;

    for(auto block = S.begin();block != S.end();block++) {
        dataBlocks b;
        b.ind = block->second.ind;
        b.keyword = block->second.keyword;
        b.pth = block->second.pth;
        tempS.push_back({b});
        dictionary[to_string(b.ind) + b.keyword] += 2 * static_cast<int>(block->second.op) - 1;
    }

    S.clear();
    for(int i = 0;i < (int)tempS.size();i++){
        if(dictionary[to_string(tempS[i].ind) + tempS[i].keyword] != 0){
            dataBlocks b;
            b.ind = tempS[i].ind;
            b.keyword = tempS[i].keyword;
            b.pth = tempS[i].pth;
            b.op = static_cast<OP>((dictionary[to_string(b.ind) + b.keyword] + 1) / 2);
            S[to_string(b.op) + to_string(b.ind) + b.keyword] = {b};
        }
    }

    return S;
}
```

Εικόνα 4.10 Μέθοδος DelRedund

Πιο συγκεκριμένα, αποθηκεύει σε ένα προσωρινό σύνολο tempS τα δεδομένα εκτός του op και σε ένα dictionary με κλειδί το (ind||ω) και τιμή έναν ακέραιο αρχικοποιημένο σε μηδέν. Για όλα τα δεδομένα που βρίσκονται στο σύνολο S, προσθέτει στην τιμή του κλειδιού (ind||ω) την τιμή 2 * op – 1. Στη συνέχεια για όλα τα δεδομένα στο σύνολο tempS εξετάζει την τιμή στο dictionary με κλειδί (ind||ω) και αν δεν είναι μηδέν τότε εισάγει ξανά στο σύνολο S το μπλοκ και η τιμή του op δίνεται από τον τύπο

$$\frac{dictionary\left[ind \,||\, \omega\right]+1}{2}$$

4.3.2 Evict

Η συνάρτηση Evict (εικόνα 4.11) παίρνει σαν ορίσματα ένα σύνολο S με τα μπλοκ δεδομένων (ind||ω||op||pth), ένα σύνολο J με τα αναγνωριστικά των κόμβων και ένα δείκτη στη stash. Η συνάρτηση ξεκινάει να γεμίζει τους κόμβους που είναι πιο κοντά στα φύλλα με Z μπλοκ δεδομένων από το S. Η επιλογή των μπλοκ δεδομένων προς εισαγωγή γίνεται με τυχαίο τρόπο έτσι ώστε τα δεδομένα που βρίσκονται στους κόμβους να αλλάζουν συχνά. Θα πρέπει όμως ο κόμβος να ανήκει στο μονοπάτι pth του μπλοκ για να μπορέσει να εισαχθεί (βλέπε παράγραφο 4.3.3). Οι κόμβοι που έχουν λιγότερα από Z μπλοκ δεδομένων συμπληρώνονται με “dummy” μπλοκ. Όσα μπλοκ δεδομένων δεν μπόρεσαν να αποθηκευτούν στους κόμβους, αποθηκεύονται στην stash. Η συνάρτηση επιστρέφει το σύνολο των κόμβων συμπληρωμένο με τα μπλοκ δεδομένων.

```
14  decrypted_nodes Auxiliary::Evict(map<string, dataBlocks> S, vector<int> J, map<string, dataBlocks>* stash){
15      std::random_device rd;
16      std::mt19937 mt(rd());
17      decrypted_nodes nodes;
18
19      for(auto itr = J.rbegin(); itr != J.rend(); itr++){
20          map<string, dataBlocks> temp;
21          for(auto block = S.begin(); block != S.end();block++){
22              if(Pth(block->second.pth, *itr)){
23                  temp[block->first] = {block->second};
24              }
25          }
26          Bucket bucket;
27          for(int z = 0; z < Z; z++){
28              // Insert in the bucket random blocks from S
29              if(temp.size() != 0){
30                  auto dis = std::uniform_int_distribution<int>{0, (int)temp.size() - 1};
31                  int rnd = dis(mt);
32                  auto it = temp.begin();
33                  std::advance(it, rnd);
34                  bucket[z] = {it->second};
35                  S.erase(it->first);
36                  temp.erase(it);
37              }else{ // dummy block
38                  bucket[z].data.ind = -1;
39                  bucket[z].data.keyword = "dummydata";
40                  bucket[z].data.op = OP::INS;
41                  bucket[z].data.pth = -1;
42              }
43          }
44          nodes.insert(make_pair(*itr, bucket));
45      }
46      for(auto block=S.begin();block != S.end(); block++) {
47          (*stash)[block->first] = block->second;
48      }
49      return nodes;
50  }
```

Εικόνα 4.11 Μέθοδος Evict

4.3.3 Pth

Η συνάρτηση Pth (εικόνα 4.12) παίρνει δύο ορίσματα, ένα αναγνωριστικό κόμβου i και ένα μονοπάτι pth. Συγκρίνει το i με το pth και επιστρέφει true ή false αν ο κόμβος i ανήκει στο μονοπάτι pth ακολουθώντας τα παρακάτω βήματα:

- Επιστρέφει true σε περίπτωση όπου το i ισούται με το pth, δηλαδή ο κόμβος i ανήκει στο μονοπάτι pth. Ακόμη επιστρέφει true εάν ο κόμβος i είναι η ρίζα.
- Επιστρέφει false εάν το pth γίνει μικρότερο από το i, δηλαδή ο κόμβος i δεν ανήκει στο μονοπάτι pth.
- Αν το pth έιναι άρτιος αριθμός σημάινει πως είναι το δεξί παιδί του κόμβου πατέρα οπότε γίνεται pth=pth/2–1 και καλείται αναδρομικά η συνάρτηση.
- Παρόμοια, αν το pth είναι περιττός αριθμός σημάινει πως είναι το αριστερό παιδί του κόμβου πατέρα οπότε γίνεται pth=pth/2 και καλείται αναδρομικά η συνάρτηση.

```
52  // return true if node i exists in path pth
53  bool Auxiliary::Pth(int pth, int i){
54      if(i == 0 || pth == i) return true;
55      if(pth < i) return false;
56      if(pth % 2 == 0) return Pth(pth / 2 - 1, i); // right child
57      if(pth % 2 == 1) return Pth(pth / 2, i); // left child
58
59      return false;
60  }
```

Εικόνα 4.12 Μέθοδος Pth

5. Αποτελέσματα

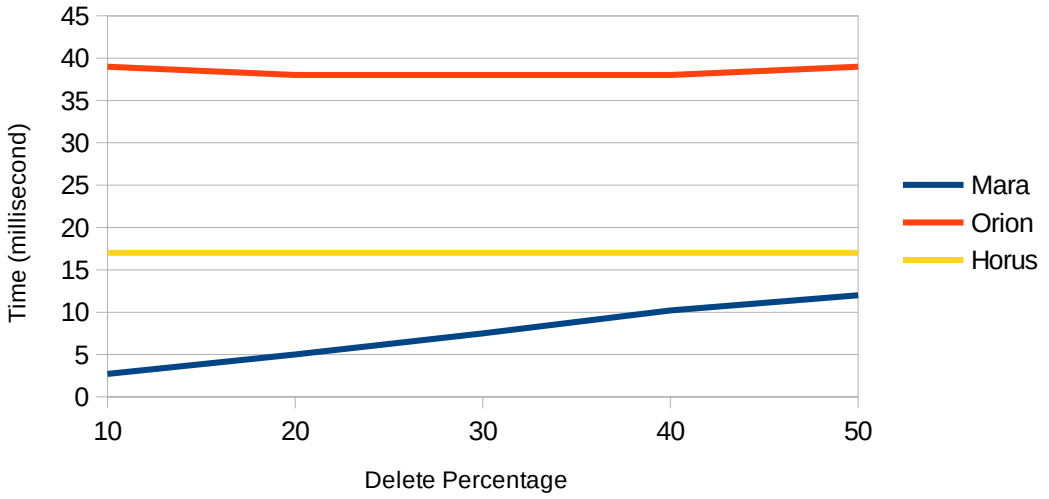
Για τους σκοπούς της πτυχιακής το πρόγραμμα υλοποιήθηκε στη γλώσσα προγραμματισμού C++ μαζί με την βοήθεια της βιβλιοθήκης OpenSSL και Libsodium για τις κρυπτογραφικές λειτουργίες. Πιο συγκεκριμένα χρησιμοποιήθηκε ο AES-256 για την κρυπτογράφηση του δέντρου και ο Blake2b για τον υπολογισμό των μονοπατιών.

Για τα πειράματα χρησιμοποιήθηκε ο υπολογιστής του Χαροκοπείου Πανεπιστημίου με διπύρηνο επεξεργαστή Intel Xeon CPU E5-2650 2.00GHz, λειτουργικό σύστημα Ubuntu 18.04.2 LTS και μνήμη RAM 16GB. Σε όλα τα πειράματα τα δεδομένα αποθηκεύονται στη μνήμη RAM. Όλα τα πειράματα επαναλήφθηκαν 10 φορές και παραθέτονται οι μέσοι όροι.

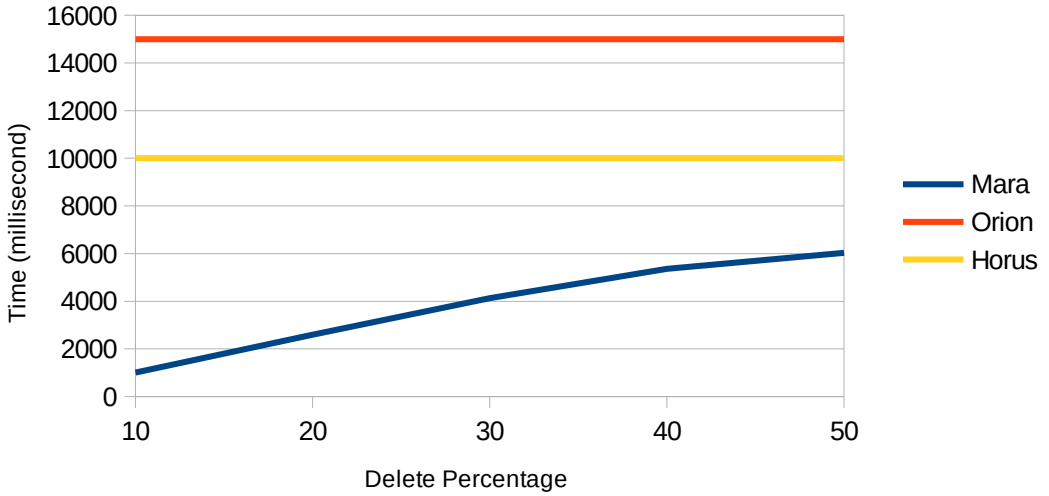
Αυτό που μας ενδιαφέρει να μετρήσουμε είναι το υπολογιστικό κόστος της διαδικασίας αναζήτησης με βάση το ποσοστό διαγραφής. Στα σχήματα 1 και 2 παρουσιάζονται οι χρόνοι αναζήτησης για δυαδικό δέντρο 100.000. Στο πρώτο σχήμα τα δεδομένα είναι 100 και στο δεύτερο είναι 20.000 και το ποσοστό διαγραφής κυμαίνεται από 10-50%.

Οι χρόνοι συγκρίθηκαν με τα σχήματα Orion και Horus από το paper New Constructions for Forward and Backward Private Symmetric Searchable Encryption [6] το οποίο πήρε μέρος στο συνέδριο ACM CCS το 2018.

Σχήμα 1 - Χρόνος Αναζήτησης |DB|=100K για 100 δεδομένα



Σχήμα 2 - Χρόνος Αναζήτησης |DB|=100K για 20K δεδομένα



Όπως φαίνεται ο χρόνος αναζήτησης για το σχήμα Orion και Horus μπορεί να μην μεταβάλλεται για σταθερό όγκο δεδομένων αλλά και τα δύο παραμένουν πιο αργά από το σχήμα Mara.

Βιβλιογραφία

[1] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. 2003. https://www.openssl.org/.

[2] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5.

[3] The Libsodium library: https://libsodium.gitbook.io/doc.

[4] Oden Goldreich, Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. 1993.

[5] Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu and Srinivas Devadas. Path-ORAM: An extremely simple Oblivious RAM protocol.

[6] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. 2018. New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In proceedings of the 22nd ACM SIGSAC conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. 668-679