



Πανεπιστήμιο Μακεδονίας  
Πρόγραμμα Μεταπτυχιακών Σπουδών  
Τμήματος Εφαρμοσμένης Πληροφορικής

**Εργασία στο μάθημα της Κρυπτογραφίας**

**Θέμα: Δημιουργία Ασφαλών Κατανεμημένων Εφαρμογών με την  
Χρήση του Ολοκληρωμένου Πλαισίου «Vanadium»**

Ζήσης Δημήτριος (mai21014)  
Μαϊκαντής Θεόδωρος (mai21030)

Υπεύθυνη Καθηγήτρια: Πετρίδου Σοφία

# Περιεχόμενα

Περίληψη.....	3
Ευρετήριο Εικόνων .....	1
Ευρετήριο Πινάκων.....	1
1   Εισαγωγή .....	1
2   Ερευνητικό Υπόβαθρο .....	1
2.1 Κρυπτογραφία .....	1
2.2 Αυθεντικοποίηση.....	3
2.2.1 Αναλυτική Περιγραφή του πρωτοκόλλου SIGMA .....	3
2.3 Δομή Πελάτη – Διακομιστή.....	4
2.4 Κλήση Απομακρυσμένης Διαδικασίας.....	5
3.1 Μοντέλο Επικοινωνίας .....	7
3.1.1 Μοντέλο Πελάτη – Διακομιστή στο Vanadium .....	7
3.1.2 Κλήση Απομακρυσμένης Διαδικασίας στο Vanadium .....	8
3.2 Μοντέλο Ασφάλειας.....	14
3.2.1 Γενικά.....	14
3.2.2 Principals & Blessings.....	14
3.2.3 Αναθέσεις.....	18
3.2.4 Περιορισμοί.....	21
3.2.5 Επαλήθευση των Blessings.....	23
3.2.6 Αυθεντικοποίηση.....	24
3.2.6 Εξουσιοδότηση .....	25
4   Υλοποίηση.....	27
4.1 Εφαρμογή Ανταλλαγής Άμεσων Μηνυμάτων .....	27
4.1.1 Κώδικας.....	27
4.1.2 Εκτέλεση .....	31
5   Συμπεράσματα .....	33
Βιβλιογραφία.....	34

# Περίληψη

Προκειμένου να υφίσταται η ασφάλεια σε ένα καταναμεμημένο σύστημα, είναι άκρως σημαντικό να αξιολογηθεί η αξιοπιστία των συμμετεχόντων οντοτήτων, καθώς η εμπιστοσύνη είναι, γενικότερα, βασική προϋπόθεση για επικοινωνία και συνεργασία. Σε αυτή την εργασία, παρουσιάζουμε ένα ολοκληρωμένο πλαίσιο (framework), το οποίο παρέχει ένα σύνολο εργαλείων, βιβλιοθηκών και υπηρεσιών για την ανάπτυξη ασφαλών καταναμεμημένων συστημάτων και εφαρμογών, ονόματι *Vanadium*.

Η διάρθρωση της παρούσας εργασίας εστιάζει στην εξερεύνηση του συγκεκριμένου framework. Το *Vanadium* αποτελεί ένα open-source framework, με το οποίο είναι δυνατή η κατασκευή ασφαλών και καταναμεμημένων cross-platform εφαρμογών. Πιο συγκεκριμένα, μελετώνται το πρωτόκολλο κλήσης απομακρυσμένης διαδικασίας *Vanadium* (RPC), το μοντέλο ασφάλειας του *Vanadium*, το οποίο καθορίζει μηχανισμούς αναγνώρισης (identification mechanisms), ελέγχου ταυτότητας (authentication mechanisms), υποστηρίζοντας, έτσι, μία πλήρως αποκεντρωμένη, λεπτομερή και ελεγχόμενη εξουσιοδότηση όταν επικοινωνούν δύο ή και περισσότερες οντότητες (π.χ. αυτόνομες διεργασίες). Σε ό,τι αφορά τους προαναφερθέντες μηχανισμούς, γίνεται περιγραφή των ιδιοτήτων τους, των υλοποιήσεων που παρέχονται από το *Vanadium Framework* και ορισμένων λεπτομερειών σχεδίασης που σχετίζονται με τα πρωτόκολλα αυτά καθ' αυτά.

Επιπλέον, εξηγείται εκτενώς η έννοια και η υλοποίηση της ονοματοδοσίας στο framework και ο τρόπος με τον οποίο το σύστημα ονομάτων του *Vanadium* επιτρέπει την ανακάλυψη συσκευών, ανεξάρτητα από τη φυσική τους θέση - με ή χωρίς σύνδεση στο διαδίκτυο. Τέλος, προκειμένου να αναδειχθούν στην πράξη ορισμένες από τις λειτουργίες και δυνατότητες του *Vanadium Framework*, παρέχεται συμπληρωματικά και μία εφαρμογή επίδειξης (σε γλώσσα προγραμματισμού *Go*), η οποία κάνει χρήση των περισσότερων λειτουργιών του framework, και συγκεκριμένα εκείνες που αφορούν την επικοινωνία, την αυθεντικοποίηση, αλλά και την αναγνώριση οντοτήτων.

Συνοψίζοντας, η συμβολή της παρούσας εργασίας εντοπίζεται, κυρίως, στο επίπεδο της μελέτης της δομής και της λειτουργίας του *Vanadium Framework*. Επεκτείνεται, όμως, και στο επίπεδο της προγραμματιστικής υλοποίησης μίας demo εφαρμογής, με την οποία μπορεί να γίνει αντιληπτό το πώς μπορεί κανείς να χρησιμοποιήσει τις δυνατότητες που προσφέρει το *Vanadium Framework* στην πράξη.

# Ευρετήριο Εικόνων

<b>Εικόνα 1:</b> Μοντέλο Πελάτη – Διακομιστή (Πολλαπλοί Πελάτες, Ένας διακομιστής) .....	5
<b>Εικόνα 2:</b> Κλήση Απομακρυσμένης Διαδικασίας .....	6
<b>Εικόνα 3:</b> Βασική Δομή του Μοντέλου .....	7
<b>Εικόνα 4:</b> Η περίπτωση δικτύου εφαρμογών .....	8
<b>Εικόνα 5:</b> Λειτουργία Εργαλείου VDL .....	11
<b>Εικόνα 6:</b> Διαπιστευτήρια (Certificates), Ιδιότητες (Blessings) και Περιορισμοί (Caveats) [3] .....	15
<b>Εικόνα 7:</b> Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος .....	16
<b>Εικόνα 8:</b> Άρνηση Πρόσβασης ανάλογα με την Ιδιότητα (Blessing) .....	17
<b>Εικόνα 9:</b> Απόδοση Ιδιότητας και Άρση της Απαγόρευσης Πρόσβασης .....	18
<b>Εικόνα 10:</b> Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος .....	18
<b>Εικόνα 11:</b> Άρνηση Πρόσβασης ανάλογα με την Ιδιότητα (Blessing) .....	19
<b>Εικόνα 12:</b> Απόδοση Ιδιότητας και Άρση της Απαγόρευσης Πρόσβασης .....	20
<b>Εικόνα 13:</b> Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος .....	21
<b>Εικόνα 14:</b> Ιεράρχηση Περιορισμών (Caveats) .....	22
<b>Εικόνα 15:</b> Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος .....	22
<b>Εικόνα 16:</b> Παράδειγμα Αναγνώρισης Blessing από τον Διακομιστή .....	24
<b>Εικόνα 17:</b> Παράδειγμα Επιλεκτικής Διαμοίρασης των Blessings .....	27
<b>Εικόνα 18:</b> Στιγμιότυπο Οθόνης Γραμμής Εντολών Διακομιστή .....	32
<b>Εικόνα 19:</b> Στιγμιότυπο Οθόνης Γραμμής Εντολών Πελάτη .....	33

# Ευρετήριο Πινάκων

<b>Πίνακας 1:</b> Κωδικοποίηση var-128 .....	13
<b>Πίνακας 2:</b> Δεσμευμένες Καταστάσεις Ελέγχου στο VOM .....	13

# 1 Εισαγωγή

Το *Vanadium* αποτελεί ένα ολοκληρωμένο πλαίσιο (framework) κατασκευής ασφαλών κατανεμημένων εφαρμογών, οι οποίες μπορούν να εκτελούνται οπουδήποτε. Συγκεκριμένα, προσφέρει ένα μοντέλο ασφάλειας, το οποίο βασίζεται σε κρυπτογραφία δημοσίου κλειδιού, που υποστηρίζει λεπτομερείς άδειες και εξουσιοδοτήσεις. Επίσης, παρέχει μια ασφαλή υποδομή επικοινωνιών που μπορεί να χρησιμοποιηθεί για εφαρμογές κέντρων δεδομένων (datacenter applications), καθώς και για εφαρμογές μικρότερης κλίμακας για επιχειρήσεις και καταναλωτές. Αυτό επιτυγχάνεται μέσω της υποστήριξης του συμμετρικού ελέγχου ταυτότητας και της κρυπτογραφημένης κλήσης απομακρυσμένης διαδικασίας (*RPC*), με υποστήριξη για αμφίδρομη ανταλλαγή μηνυμάτων, ροή και διακομιστή μεσολάβησης, που λειτουργεί σε μια ποικιλία πρωτοκόλλων δικτύου, όπως το πρωτόκολλο *Bluetooth*.

## 2 Ερευνητικό Υπόβαθρο

### 2.1 Κρυπτογραφία

Η κρυπτογραφία έχει βασικό ρόλο στην λειτουργικότητα του δικτύου *Vanadium*, καθώς χωρίς αυτήν δεν είναι δυνατόν να λειτουργήσει σωστά και με ασφάλεια η δομή. Η κύρια αρμοδιότητα της κρυπτογραφίας είναι σωστή λειτουργία της εξουσιοδότησης μεταξύ δύο οντοτήτων. Αυτό επιτυγχάνεται μέσω της διασφάλισης της αυθεντικοποίησης και της εμπιστευτικότητας. Η εμπιστευτικότητα εξασφαλίζεται αυτομάτως, καθώς οντότητες που δεν έχουν περάσει την διαδικασία της αυθεντικοποίησης δεν αποκτούν πρόσβαση. Όσον αφορά τον τρόπο αυθεντικοποίησης, εξετάζονται δύο συνηθισμένες μέθοδοι.

Η πρώτη μέθοδος αυθεντικοποίησης είναι η παροχή ενός διακριτικού πρόσβασης (θα αναφέρεται και ως *Access Token*). Η χρήση των *Access Token* συναντάται σε μηχανισμούς όπως το ολοκληρωμένο πλαίσιο *OAuth2* [1] το οποίο χρησιμοποιείται από μεγάλες εταιρίες όπως η *Google* [2]. Τα διακριτικά πρόσβασης είναι διαπιστευτήρια που χρησιμοποιούνται για την

απόκτηση κάποιας μορφής πρόσβασης, π.χ. σε μία υπηρεσία. Αυτά παράγονται για λογαριασμό του πελάτη χρησιμοποιώντας τεχνικές συμμετρικής κρυπτογραφίας. Η ακολουθία χαρακτήρων του διακριτικού πρόσβασης συνήθως δεν προσφέρει κάποια πληροφορία στον χρήστη, με την έννοια ότι δεν του γνωστοποιεί το πώς παράχθηκε και τι δυνατότητες του δίνει [1]. Αυτή η μέθοδος είναι ευρέως διαδεδομένη, κυρίως στην χρήση σε εφαρμογές διαδικτύου, καθώς είναι απλή στην υλοποίηση της, αποδοτική και εύκολο να υιοθετηθεί σε κάποιο σύστημα. Παρά τα θετικά στοιχεία της, αυτή η μέθοδος δεν κρίθηκε κατάλληλη για την υλοποίηση του Vanadium, καθώς έχει κάποιους περιορισμούς. Η επιβεβαίωση και αποκωδικοποίηση των διαπιστευτηρίων, μπορεί να γίνει μόνο από την μεριά του εκδότη τους, επομένως ο ίδιος πρέπει να κληθεί με κάθε νέα αίτηση πρόσβασης [3].

Η δεύτερη μέθοδος είναι χρήση ασύμμετρης κρυπτογραφίας, η οποία πραγματοποιείται συνδυάζοντας διαφόρων ειδών τεχνολογιών, αλλά σαν κύριο χαρακτηριστικό της είναι η χρήση του ζεύγους Δημόσιου-Ιδιωτικού κλειδιού. Αυτή η μέθοδος ξεκίνησε με την δημιουργία του Pretty Good Privacy (PGP) [4], και στην συνέχεια αναβαθμίστηκε και εξελίχθηκε μέσω πολλών μεταγενέστερων εκδόσεων. Κάποιες από τις πιο πρόσφατες εκδόσεις είναι το OpenPGP και το GnuPGP, τα οποία αποτελούν κομμάτι του "Internet Official Protocol Standards" [5]. Με την χρήση αυτής της μεθόδου, αποφεύγουμε τον περιορισμό που συναντάτε με την χρήση των διακριτικών πρόσβασης, καθώς κάθε οντότητα κατέχει ένα ζεύγος Δημόσιου και Ιδιωτικού κλειδιού. Αυτό σημαίνει πως δεν χρειάζεται η παροχή και η επιβεβαίωση κάποιου Access Token από έναν τρίτο, για να ξεκινήσει η διαδικασία επικοινωνίας. Για τους παραπάνω λόγους, η μέθοδος αυτή είναι ιδιαίτερα διαδεδομένη στο διαδίκτυο και χρησιμοποιείται σε διάφορες εφαρμογές όπως για παράδειγμα την ηλεκτρονική αλληλογραφία.

Οι επικοινωνίες που δρομολογούνται μέσω του δικτύου Vanadium είναι σημαντικό να έχουν τα χαρακτηριστικά της μεθοδολογίας Δημόσιου και Ιδιωτικού κλειδιού, για τον λόγο αυτό και υιοθετήθηκε παρόμοια τακτική. Το πλαίσιο που χρησιμοποιεί το Vanadium, είναι ο συνδυασμός δύο πρωτοκόλλων του rfc2693, του διαδεδομένου μοντέλου Simple Distributed Security Infrastructure (SDSI) [6] και του Simple Public Key Infrastructure (SPKI) [6]. Το μοντέλο που προκύπτει ως αποτέλεσμα αναφέρεται ως SPKI/SDSI [3].

## 2.2 Αυθεντικοποίηση

Η αυθεντικοποίηση στο Vanadium βασίζεται στο πρωτόκολλο *SIGMA-I*. Διασφαλίζει ότι κάθε άκρο, τελικώς, θα είναι σίγουρο ότι το άλλο άκρο, με το οποίο επικοινωνεί, κατέχει το αντίστοιχο μυστικό κλειδί. Στο τέλος του πρωτοκόλλου, δημιουργείται ένα κρυπτογραφημένο κανάλι (βασισμένο στο κοινόχρηστο κλειδί) μεταξύ του πελάτη και του διακομιστή για περαιτέρω επικοινωνία.

### 2.2.1 Αναλυτική Περιγραφή του πρωτοκόλλου SIGMA

**Αρχικές πληροφορίες:** Οι πρώτοι αριθμοί  $p$ ,  $q$ ,  $q / p - 1$  και  $g$  της σειράς  $q$  στο  $Z^*$ . Κάθε οντότητα έχει ένα ιδιωτικό κλειδί για έναν αλγόριθμο υπογραφής  $SIG$ , και όλοι διαθέτουν τα δημόσια κλειδιά επαλήθευσης των άλλων οντοτήτων. Το πρωτόκολλο χρησιμοποιεί επίσης ένα μήνυμα, το οποίο ανήκει στην κατηγορία ελέγχου ταυτότητας  $MAC$  και μια ψευδοτυχαία συνάρτηση  $PRF$  [7].

**Τα μηνύματα του πρωτοκόλλου:**

- Έναρξη μηνύματος ( $I \rightarrow R$ ):  $s, g^x$
- Μήνυμα απάντησης ( $R \rightarrow I$ ):  $s, g^y, ID_r, SIG_r("1", s, g^x, g^y), MAC_{k1}("1", s, ID_r)$
- Μήνυμα Ολοκλήρωσης ( $I \rightarrow R$ ):  $s, ID_i, SIG_i("0", s, g^y, g^x), MAC_{k1}("0", s, ID_i)$

**Οι ενέργειες του πρωτοκόλλου:**

1. Το αρχικό μήνυμα αποστέλλεται από τον εκκινητή  $ID_i$  κατά την ενεργοποίηση με αναγνωριστικό συνεδρίας  $s$  (αφού ελέγξει ότι δεν ξεκίνησε καμία προηγούμενη συνεδρία στο  $ID_i$  με αναγνωριστικό  $s$ ). Το DH εκθετικό  $g^x$  υπολογίζεται με  $x \xleftarrow{R} Z_q$  και το  $x$  αποθηκεύεται στην κατάσταση συνεδρίας  $(ID_i, s)$  [7].
2. Όταν ένα αρχικό μήνυμα με αναγνωριστικό συνεδρίας  $s$  παραδίδεται σε μια οντότητα  $ID_r$  (εάν το  $s$  δεν υπήρχε πριν στο  $ID_r$ ), το  $ID_r$  ενεργοποιεί μια τοπική συνεδρία (ως ανταποκριτής). Πλέον, είναι σε θέση να δημιουργήσει το μήνυμα απόκρισης, όπου το εκθετικό DH υπολογίζεται με  $y$



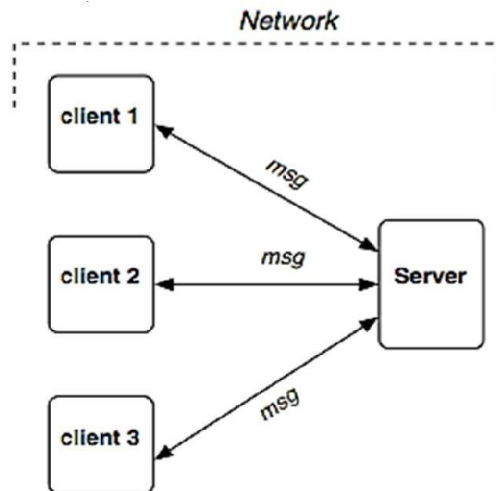
$\xleftarrow{R} Z_q$ , η υπογραφή  $SIG_r$  υπολογίζεται υπό την υπογραφή κλειδιού του  $ID_r$  και η τιμή  $g^x$  που τοποθετείται κάτω από την υπογραφή είναι ο εκθέτης DH που λαμβάνεται από τον  $ID_r$  στο εισερχόμενο μήνυμα έναρξης. Η τιμή  $MAC_{k_1}$  παράγεται με  $k_1 = PRF_{g^{xy}}(1)$  όπου το  $g^{xy}$  υπολογίζεται από  $ID_r$  ως  $(g^y)^x$ . Τέλος, η τιμή  $k_0 = PRF_{g^{xy}}(0)$  υπολογίζεται και διατηρείται στη μνήμη και οι τιμές  $y$  και  $g^{xy}$  διαγράφονται [7].

3. Μόλις ληφθεί ένα μήνυμα απόκρισης με το αναγνωριστικό συνεδρίας  $s$ , το  $ID_i$  ανακτά το δημόσιο κλειδί του μέρους του οποίου η ταυτότητα  $ID_r$  εμφανίζεται σε αυτό το μήνυμα και χρησιμοποιεί το συγκεκριμένο κλειδί, προκειμένου να επαληθεύσει την υπογραφή του τετραπλού  $(\text{"1"}, s, g^x, g^y)$ , όπου  $g^x$  είναι η τιμή που αποστέλλεται από το  $ID_r$  στο αρχικό μήνυμα και  $g^y$  η τιμή, η οποία ελήφθη σε αυτό το μήνυμα απάντησης. Το  $ID_i$  ελέγχει επίσης το ληφθέν MAC με το κλειδί  $k_1 = PRF_{g^{xy}}(1)$ , όπου το  $g^{xy}$  υπολογίζεται ως  $(g^y)^x$  και στο ταυτότητα  $ID_r$  όπως εμφανίζεται στο μήνυμα απόκρισης. Εάν κάποιο από αυτά τα βήματα επαλήθευσης αποτύχει, η συνεδρία ακυρώνεται και η έξοδος (output) "ακυρώθηκε ( $ID_i, s$ )" δημιουργείται. Η κατάσταση συνεδρίας διαγράφεται. Εάν η επαλήθευση πετύχει τότε το  $ID_i$  ολοκληρώνει τη συνεδρία με δημόσια έξοδο (public output)  $(ID_i, s, ID_r)$  και μυστικό κλειδί συνεδρίας  $k_0$  υπολογιζόμενο ως  $k_0 = PRF_{g^{xy}}(0)$ . Το μήνυμα ολοκλήρωσης στέλνεται και η κατάσταση συνεδρίας διαγράφεται [7].
4. Μόλις ληφθεί το μήνυμα ολοκλήρωσης της συνεδρίας, το  $ID_r$  επαληθεύει την υπογραφή (υπό το δημόσιο κλειδί της οντότητας  $ID_i$  και με το  $g^y$  να είναι η τιμή DH), η οντότητα  $ID_r$  στέλνει το μήνυμα απόκρισης και επαληθεύει το MAC υπό το κλειδί  $k_1$ , το οποίο υπολογίστηκε στο βήμα 2. Εάν κάποιο από τα βήματα επαλήθευσης αποτύχει, η συνεδρία ματαιώνεται (με την έξοδο "ματαιώθηκε ( $ID_r, s$ )"). Διαφορετικά, το  $ID_r$  ολοκληρώνει την συνεδρία με δημόσια έξοδο  $(ID_r, s, ID_i)$  και μυστικό κλειδί συνεδρίας  $k_0$ . Η κατάσταση συνεδρίας διαγράφεται [7].

## 2.3 Δομή Πελάτη – Διακομιστή

Το μοντέλο πελάτη – διακομιστή (ή πελάτη – εξυπηρετητή, client – server model) αποτελεί μια υποδομή για κατανεμημένες εφαρμογές, δηλαδή συστήματα στα οποία διάφορες συσκευές συνδέονται μεταξύ τους μέσω ενός δικτύου (ή και του διαδικτύου). Με αυτό τον τρόπο

οι πελάτες (υπολογιστής, έξυπνο τηλέφωνο, IoT συσκευή κλπ.), έχουν την δυνατότητα να ζητούν υπηρεσίες από έναν διακομιστή, ο οποίος προσφέρει πληροφορίες ή/και επιπρόσθετη υπολογιστική ισχύ. Το συγκεκριμένο μοντέλο, χρησιμοποιείται από τις περισσότερες (κατανεμημένες) εφαρμογές σήμερα. [8]



Εικόνα 1: Μοντέλο Πελάτη – Διακομιστή (Πολλαπλοί Πελάτες, Ένας διακομιστής)

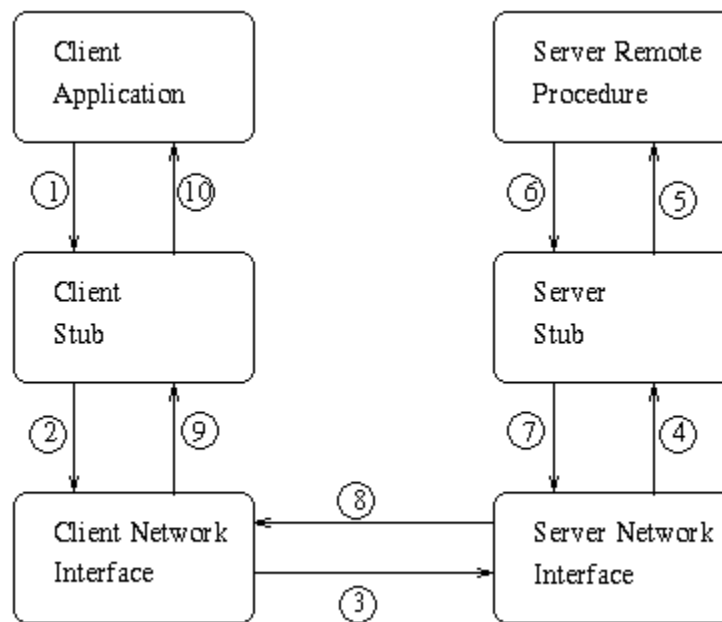
Πηγή: [Architecture of Network and Client-Server model](#)

Στην παρούσα εργασία, θα ασχοληθούμε με **μία από τις σημαντικότερες πτυχές του μοντέλου πελάτη – διακομιστή: Την ασφάλεια των δεδομένων που ρέουν σε ένα δίκτυο μέσω μιας κατανεμημένης εφαρμογής**, λόγω των αιτημάτων των πελατών (client requests) και των απαντήσεων του διακομιστή (server replies). Η ανταλλαγή των δεδομένων αυτών, γίνεται μέσω διαφόρων πρωτοκόλλων μεταφοράς (transport protocols), όπως το *TCP* στην περίπτωση του *Vanadium*.

## 2.4 Κλήση Απομακρυσμένης Διαδικασίας

Η Κλήση Απομακρυσμένης Διαδικασίας (*Remote Procedure Call*) αποτελεί μια τεχνική επικοινωνίας μεταξύ διεργασιών που χρησιμοποιείται για εφαρμογές που βασίζονται στο μοντέλο [πελάτη - διακομιστή](#).

Ένας πελάτης έχει ένα μήνυμα αίτησης, το οποίο το παραλαμβάνει η εκάστοτε βιβλιοθήκη RPC, το μεταφράζει και το στέλνει στο διακομιστή. Αυτό το αίτημα μπορεί να είναι μια διαδικασία ή μια κλήση διαδικασίας σε έναν απομακρυσμένο διακομιστή (υπολογιστή). Μόλις ο διακομιστής λάβει το αίτημα, στέλνει απάντηση πίσω στον πελάτη. Στην γενική περίπτωση, ο πελάτης συνεχίζει την εκτέλεσή του μόνο μετά την ολοκλήρωση του διακομιστή [9].



**Εικόνα 2:** Κλήση Απομακρυσμένης Διαδικασίας

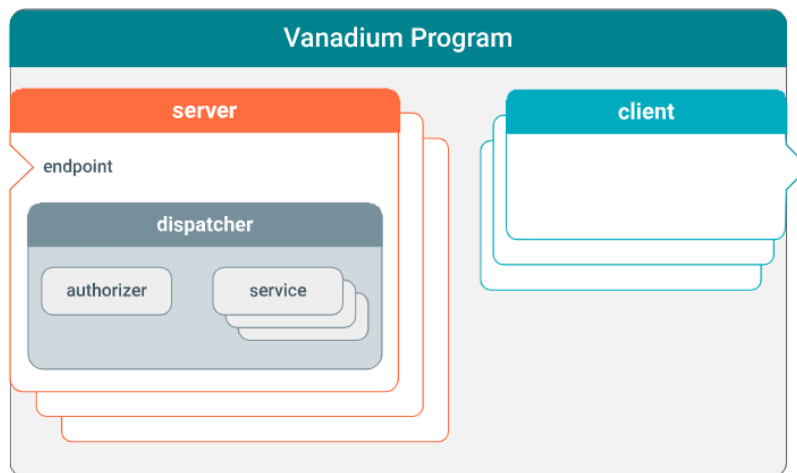
Πηγή: <http://web.mit.edu/6.033/1997/reports/dp1-danlief.html>

Η εικόνα 2 απεικονίζει τη ροή μιας κλήσης απομακρυσμένης διαδικασίας. Αρχικά, η εφαρμογή πελάτη πραγματοποιεί μια κλήση διαδικασίας, η οποία δε διαφέρει σε κάτι από μια συμβατική κλήση διαδικασίας (τοπική). Η διαδικασία που καλεί ο πελάτης ονομάζεται στέλεχος (stub). Το στέλεχος δημιουργείται από την εκάστοτε βιβλιοθήκη *RPC*. μιας συμβατικής (τοπικής) κλήσης διαδικασίας. Σειριοποιεί τα ορίσματα που του διαβιβάζονται, τα «συσκευάζει» με πρόσθετες πληροφορίες που απαιτούνται από τον διακομιστή και μεταδίδει ολόκληρο το πακέτο (που αποτελείται από τα σεριοποιημένα ορίσματα και τις επιπρόσθετες πληροφορίες) στη διεπαφή του δικτύου πελάτη (Client Network Interface) [9].

## 3.1 Μοντέλο Επικοινωνίας

### 3.1.1 Μοντέλο Πελάτη – Διακομιστή στο Vanadium

Στο *Vanadium*, ένας διακομιστής αποτελείται από ένα τελικό σημείο (endpoint), μια διεύθυνση *Vanadium* που περιλαμβάνει ένα όνομα κεντρικού υπολογιστή, μια θύρα και έναν διαβιβαστή (dispatcher). Ο διαβιβαστής δρομολογεί κατάλληλα ένα εισερχόμενο αίτημα σε έναν εξουσιοδοτούντα (authorizer) και μια υπηρεσία. Το αίτημα φτάνει στην υπηρεσία **μόνο εάν το εγκρίνει ο εξουσιοδοτών**. Ένας διαβιβαστής έχει στην κατοχή του έναν εξουσιοδοτούντα και μία ή περισσότερες υπηρεσίες. Όλες οι υπηρεσίες περιέχουν **μόνο** τον κώδικα που υλοποιεί την λογική τους, δεν έχουν γνώση της ταυτότητας του εξουσιοδοτούντος που τις προστατεύει [10].



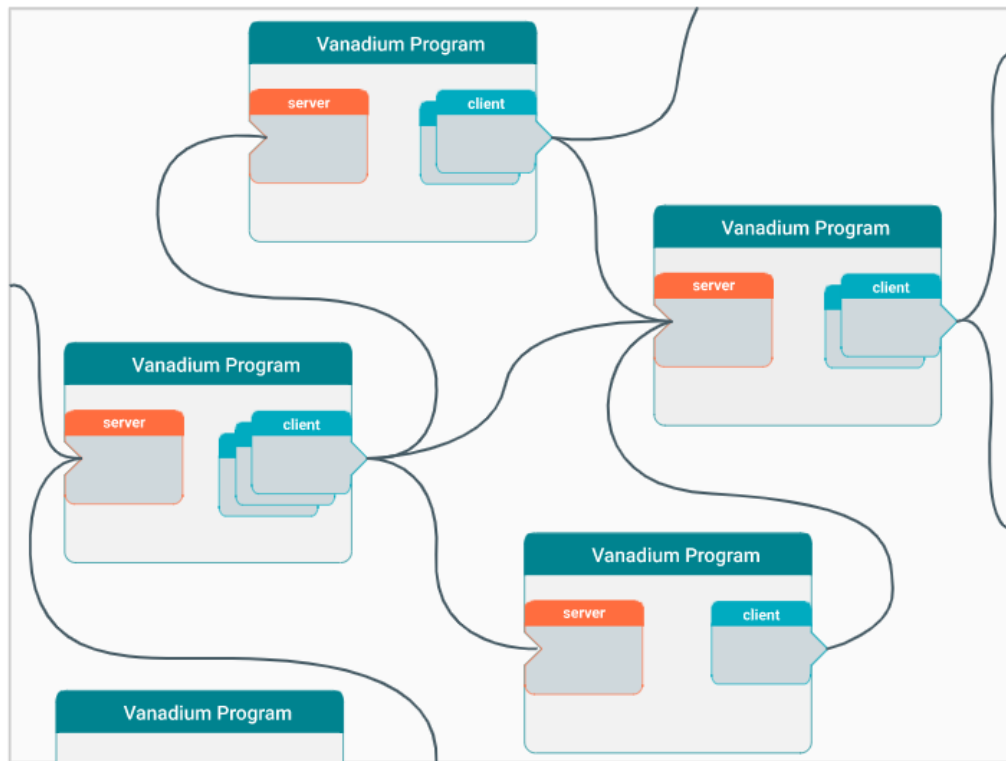
Εικόνα 3: Βασική Δομή του Μοντέλου

Πηγή: <https://vanadium.github.io/>

Ένας πελάτης είναι απλώς ένας κώδικας στελέχους (stub code), ο οποίος έχει την δυνατότητα πρόσβασης σε ένα ή περισσότερα τελικά σημεία στο δίκτυο (και κατ' επέκταση υπηρεσίες) με βάση το όνομα που του έχει δώσει ο διακομιστής (άρα πρέπει να έχει προηγηθεί ονοματοδοσία).

Όπως είδαμε στο [κεφάλαιο 4.1.1](#), η βασική εκδοχή του μοντέλου πελάτη – διακομιστή συμπεριλαμβάνει έναν διακομιστή και πολλαπλούς πελάτες. Ωστόσο, **μια εφαρμογή *Vanadium***

έχει την δυνατότητα να φιλοξενήσει οσοσδήποτε διακομιστές και πελάτες, δημιουργώντας, έτσι, μία αρκετά πιο πολύπλοκη κατανομημένη εφαρμογή. Η ιδέα είναι ότι ένα πρόγραμμα θα προσφέρει ορισμένες υπηρεσίες και θα επικοινωνεί με άλλα προγράμματα σε μια προσπάθεια παροχής αυτών των υπηρεσιών.



Εικόνα 4: Η περίπτωση δικτύου εφαρμογών

Πηγή: <https://vanadium.github.io/>

### 3.1.2 Κλήση Απομακρυσμένης Διαδικασίας στο Vanadium

Όπως έχουμε ήδη αναφέρει, το *Vanadium* κάνει χρήση του μοντέλου πελάτη – διακομιστή, συνδυάζοντάς το με την τεχνική της Κλήσης Απομακρυσμένης Διαδικασίας [10]. Σαν δομή, το RPC του ολοκληρωμένου πλαισίου *Vanadium* δε διαφέρει σε κάτι από εκείνη που είδαμε στο κεφάλαιο 4.1.2, αφού αποτελεί μια υλοποίηση αυτής με πρόσθετη κρυπτογραφική υποστήριξη.

**Στο *Vanadium RPC*, υφίσταται συμμετρικός έλεγχος ταυτότητας και κρυπτογραφημένη, αμφίδρομη ανταλλαγή μηνυμάτων.** Η χρήση της υποδομής αυτής είναι κατάλληλη για εφαρμογές κέντρου δεδομένων μεγάλης κλίμακας (data centers, clusters), καθώς και για εφαρμογές μικρότερης κλίμακας. [3]

Ιδιαίτερο ενδιαφέρον έχουν δύο επιμέρους συστατικά του *Vanadium RPC*, το ***Vanadium Object Marshalling*** (VOM) και το ***Vanadium Definition Language*** (VDL).

### **3.1.2.1 *Vanadium Definition Language***

**Το *Vanadium Definition Language* (VDL) αποτελεί μια γλώσσα προδιαγραφών που χρησιμοποιείται για την περιγραφή της διεπαφής προγραμματισμού εφαρμογών ενός λογισμικού.** Αποτελεί μια ειδική υλοποίηση της Γλώσσα περιγραφής διεπαφής (*Interface Definition Language, IDL*) και **επιτρέπει τη διαλειτουργικότητα μεταξύ στοιχείων λογισμικού που εκτελούνται σε διαφορετικά περιβάλλοντα.** Για παράδειγμα, μια εφαρμογή γραμμένη σε Java μπορεί να επιθυμεί να επικοινωνήσει με ένα backend γραμμένο στο Go και βρίσκεται στο cloud. Η χρήση του *VDL*, αν και **δεν είναι απαραίτητη**, συστήνεται. (Taly & Shankar, 2016)

Η πιο συνήθης χρήση του *VDL*, είναι για τον ορισμό διεπαφών. **Μια διεπαφή (interface) αντιπροσωπεύει ένα σύνολο υπογραφών μεθόδων, χωρίς όμως να συμπεριλαμβάνεται η υλοποίησή τους (το σώμα).** Οι διεπαφές μπορούν να ενσωματώσουν και άλλες διεπαφές αναφέροντας μόνο το όνομα της προς ενσωμάτωση διεπαφής. Αυτό προσθέτει όλες τις μεθόδους της ενσωματωμένης διεπαφής στο σύνολο των μεθόδων.

Για να αντιληφθούμε καλύτερα την έννοια του *VDL*, ας δούμε το παράδειγμα ορισμού της διεπαφής μιας αριθμομηχανής (calculator):

```

/* Κάθε διεπαφή μπορεί να ορίζει τις υπογραφές των μεθόδων */
type Calculator interface {

    Add(x, y bignum.Int) (bignum.Int | error)

    Sub(x, y bignum.Int) (bignum.Int | error)

    Multi(x, y bignum.Int) (bignum.Int | error)

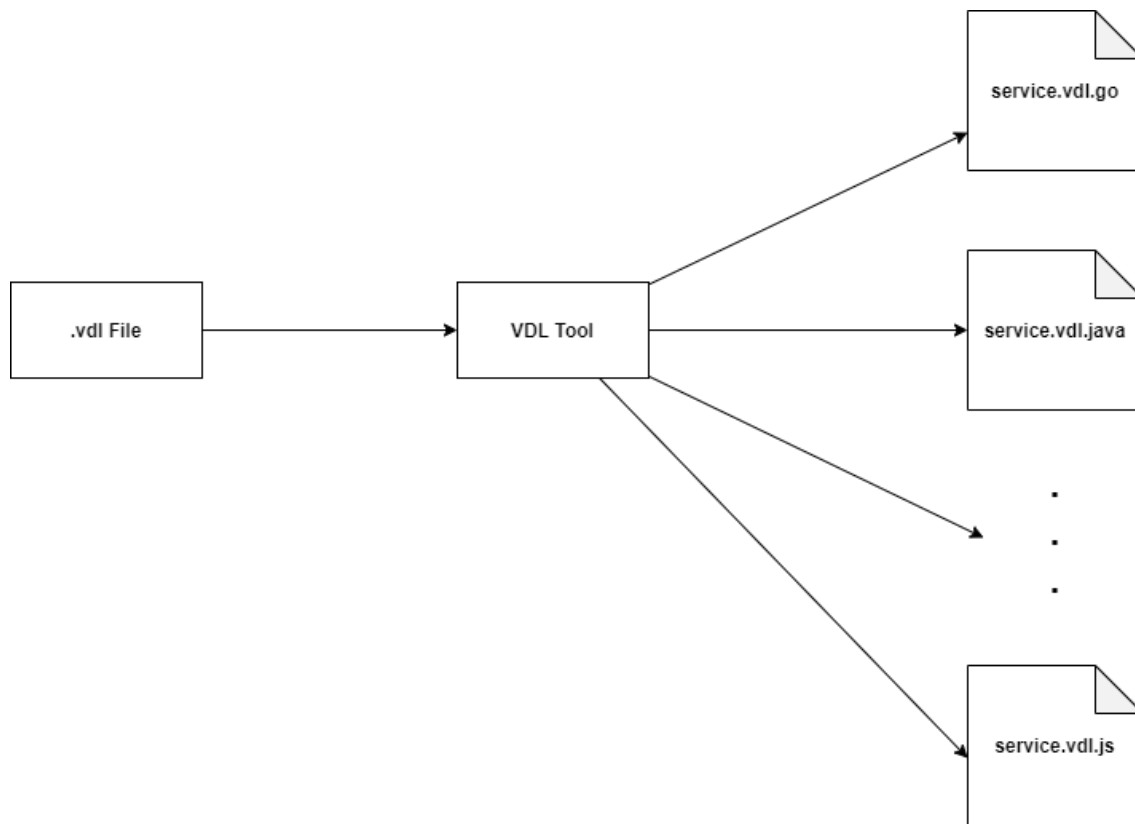
    Div(x, y bignum.Int) (bignum.Int | error)

}

```

Για να δημιουργήσουμε ένα πρόγραμμα, το οποίο θα κάνει αριθμητικούς υπολογισμούς (στην περίπτωσή μας μόνο για ακραίους), πρέπει να προσδιορίσουμε τις 4 βασικές αριθμητικές πράξεις ως μεθόδους. Κάθε μέθοδος θα λαμβάνει 2 αριθμούς (ορίσματα μεθόδου) και θα επιστρέφει έναν νέο ακέραιο (`bignum.Int`, αντίστοιχος του `long`). Όπως προαναφέραμε, η υλοποίησή τους δεν περιλαμβάνεται.

Τέλος, αφού ολοκληρώσαμε το αρχείο *vdل* που περιέχει την διεπαφή, θα χρησιμοποιήσουμε το εργαλείο *VDL (VDL Tool)* για να παράξουμε τον κώδικα στελέχους του πελάτη και του διακομιστή (stub code), στην γλώσσα της επιλογής μας. Ουσιαστικά, το εργαλείο *VDL*, πρόκειται για έναν μεταγλωττιστή πρωτοκόλλου (*protocol compiler*), αντίστοιχο με τον *thrift* στο ολοκληρωμένο πλαίσιο *Apache Thrift*, ή τον *proto* στο ολοκληρωμένο πλαίσιο *GRPC*.



Εικόνα 5: Λειτουργία Εργαλείου VDL

### 3.1.2.2 Vanadium Object Marshalling

Το *Vanadium Object Marshalling* (VOM) αποτελεί μηχανισμό κωδικοποίησης / σειριοποίησης των κλήσεων του *Vanadium RPC*. Πρόκειται για μια ειδική υλοποίηση του μηχανισμού που αναφέραμε και στην [γενικότερη περιγραφή της ιδέας της Κλήσης Απομακρυσμένης Διαδικασίας](#). Το VOM σειριοποιεί, ουσιαστικά, κάθε τιμή που υφίσταται σε αρχεία [VDL](#).

Το VOM είναι μια δυαδική μορφή βασισμένη σε byte, η οποία αποτελείται από μια ακολουθία μηνυμάτων. Κάθε μήνυμα είναι είτε ορισμός τύπου (type), είτε ορισμός τιμής (value). Προφανώς, οι τύποι καθορίζονται προτού τεκμηριωθούν από τιμές.

Προκειμένου να γίνει περισσότερο κατανοητός ο τρόπος λειτουργίας και αναπαράστασης του VOM, μπορούμε να ρίξουμε μια ματιά στο συντακτικό του:



```

/* Το πρώτο byte πάντα είναι ο αριθμός της έκδοσης */
VOM:      version Message*

/* Κάθε μήνυμα θα είναι ορισμός τύπου (TypeMsg) ή ορισμός τιμής (ValueMsg) */
Message:  TypeMsg | ValueMsg

/* Κάθε μήνυμα ξεκινά με την τιμή του typeid, η οποία μπορεί να είναι θετική
(+typeid) ή αρνητική (-typeid). Όσα μηνύματα ξεκινούν με θετικό typeid, αποτελούν
ορισμό τύπων, ενώ όσα μηνύματα ξεκινούν με θετικό typeid, αποτελούν ορισμό τιμής,
σε ήδη ορισμένο τύπο */
TypeMsg:  -typeid MainValue(string)

ValueMsg:  +typeid MainValue

```

### 3.1.2.2.1 Κωδικοποίηση

Η βάση για την κωδικοποίηση VOM είναι το var128 (Variable-length quantity), ένας μη-προσημασμένος ακέραιος αριθμός μεταβλητού μήκους με μέγιστο μέγεθος 128 bit (δηλ. 16 bytes). Αυτή είναι μια κωδικοποίηση βασισμένη σε byte. Οι τιμές στο εύρος [0, 127] κωδικοποιούνται κατά λέξη σε ένα byte. Μεγαλύτερες τιμές χρησιμοποιούν το πρώτο byte για να κωδικοποιήσουν το μήκος byte της τιμής, με τα επόμενα byte να κωδικοποιούν την τιμή στην ελάχιστη big-endian αναπαράσταση της (το σημαντικότερο byte αποθηκεύεται στην "μικρότερη" θέση μνήμης) [10].

Το πρώτο byte της κωδικοποίησης αποτελεί τον πυρήνα (crux) της κωδικοποίησης. Από τις 256 πιθανές καταστάσεις για το πρώτο byte, 128 καταστάσεις χρησιμοποιούνται για την αναπαράσταση των τιμών [0, 127], και άλλες 16 καταστάσεις χρησιμοποιούνται για να αντιπροσωπεύουν τα μήκη πολλαπλών byte [1, 16]. Αυτό αφήνει 112 καταστάσεις, οι οποίες προορίζονται για καταχωρήσεις ελέγχου [10].

var128 πρώτου byte	7	6	5	4	3	2	1	0
0x00..0x7F Τιμή ενός byte	0	Τιμή εύρους [0, 127]						

var128 πρώτου byte	7	6	5	4	3	2	1	0
<b>0x80..0xBF</b> Control1 (64 καταχωρήσεις)	1	0	-	-	-	-	-	-
<b>0xC0..0xDF</b> Control2 (32 καταχωρήσεις)	1	1	0	-	-	-	-	-
<b>0xE0..0xEF</b> Control3 (16 καταχωρήσεις)	1	1	1	0	-	-	-	-
<b>0xF0..0xFF</b> Μήκος Πολλών byte (FF=1, FE=2, ...)	1	1	1	1	Μήκος εύρους [1, 16]			

**Πίνακας 1:** Κωδικοποίηση var-128

Αξίζει να σημειωθεί ότι ενώ έχουν δεσμευτεί 112 καταστάσεις για τις καταχωρήσεις ελέγχου, μόλις 2 χρησιμοποιούνται:

<b>0xE0</b>	NIL	Αναπαριστά μη-υπάρχουσα τιμή (αντίστοιχο του NULL)
<b>0xE1</b>	END	Αναπαριστά το τέλος μιας σειράς (αντίστοιχο του EOF)

**Πίνακας 2:** Δεσμευμένες Καταστάσεις Ελέγχου στο VOM

**Το VOM υποστηρίζει τόσο την σειριοποίηση βασικών τύπων (int, float, bool κλπ.), όσο και τύπων ορισμένους από τον χρήστη.** Προφανώς, για τους τύπους ορισμένους από τον χρήστη, υπάρχει η προϋπόθεση ότι και οι δύο πλευρές (πελάτης και διακομιστής) γνωρίζουν τους τύπους αυτούς.

## 3.2 Μοντέλο Ασφάλειας

### 3.2.1 Γενικά

Στο δίκτυο, οι οντότητες που υπάρχουν είναι εν'δυνάμει και πελάτες και διακομιστές, επομένως μπορούν να επικοινωνούν και να προσφέρουν υπηρεσίες η μία στην άλλη. Για την κατανάλωση μιας υπηρεσίας, μια οντότητα πρέπει να πρώτα να λάβει την κατάλληλη αδειοδότηση από τον πάροχό της. Όπως αναφέραμε, το πλαίσιο χρησιμοποιεί το μοντέλο SPKI/SDSI. Επομένως, για την αυθεντικοποίηση και την αδειοδότηση των οντοτήτων, το Vanadium στηρίζεται στην ασύμμετρη μορφή κρυπτογραφίας, με την χρήση ζεύγους δημόσιου και ιδιωτικού κλειδιού. Το ζεύγος κλειδιών, μαζί με το όνομα της οντότητας, λειτουργούν ως τα διαπιστευτήριά της. Παρακάτω θα δούμε περισσότερες λεπτομέρειες για τα δομικά συστατικά και τις λειτουργίες ενός δικτύου Vanadium. Αυτά είναι οι οντότητες του δικτύου και ο τρόπος με τον οποίο αλληλεπιδρούν. Επίσης θα δούμε την λογική της αδειοδότησης και τα συστατικά από τα οποία αποτελείται. Έπειτα θα δούμε τις επιπρόσθετες λειτουργίες που δρουν επικουρικά στην διαδικασία της αδειοδότησης και τέλος θα εξετάσουμε τις λειτουργίες της επαλήθευσης και της εξουσιοδότησης.

### 3.2.2 Principals & Blessings

Στο δίκτυο του Vanadium τα *Principals* αποτελούν τις οντότητες του δικτύου που αλληλεπιδρούν μεταξύ τους. Τα *Blessings* είναι οι ιδιότητες κάποιας οντότητας, βάση των οποίων κρίνετε η δυνατότητα επικοινωνίας της με κάποια άλλη. Αν θεωρήσουμε σαν οντότητα έναν πάροχο υπηρεσίας, θα μπορούσε ο ίδιος να αδειοδοτήσει έναν τρίτο. Ο σκοπός της αδειοδότησης είναι να δώσει στον τρίτο, την δυνατότητα να καταναλώσει την υπηρεσία του. Πέραν της άμεσης σύνδεσης ενός *Blessing* ως άδεια επικοινωνίας μεταξύ *Principal*, χρησιμοποιούνται για την ευκολότερη κατανόησή των αδειών από ανθρώπους. Στην ουσία το *Blessing* είναι η αντιστοίχιση μιας άδειας πρόσβασης με κείμενο. Το κείμενο, είναι μορφής ονόματος ή ιδιότητας ώστε να παρομοιάζει την ανθρώπινη καθημερινότητα. Συμπεράνουμε λοιπόν, πως για κάθε επικοινωνία μεταξύ οντοτήτων μέσα στο δίκτυο Vanadium, είναι προηγουμένως απαραίτητη η αυθεντικοποίηση.

Για τον σκοπό της αυθεντικοποίησης, κάθε οντότητα έχει ένα ζευγάρι δημόσιου και ιδιωτικού κλειδιού. Το ιδιωτικό κλειδί του ζεύγους, ποτέ δεν μεταφέρετε ή διαμοιράζετε μέσω του δικτύου. Κύριο ρόλο στην διαδικασία έχουν τα δημόσια κλειδιά σε συνδυασμό με τις Ιδιότητες (Blessing). Μία οντότητα μπορεί να έχει μία ή περισσότερες Ιδιότητες, βάση των οποίων μπορεί να αποκτά δικαίωμα επικοινωνίας με άλλες οντότητες. Κάθε αίτημα πρόσβασης πρέπει να επιβεβαιωθεί και να εγκριθεί. Κατά την διαδικασία έγκρισης, δεν είναι αναγκαία η παροχή του δημόσιου κλειδιού, εφόσον ο αιτών μπορεί να αποδείξει την κατοχή κατάλληλης ιδιότητας (Blessing). Μία ιδιότητα αποτελείται από ένα σύνολο διαπιστευτηρίων.

$n ::= \text{ordinary names not containing}$	$/$	names
$C ::= C\langle n, pk, clist, sig \rangle$		certificates
$B ::= C$		blessings
$  B : C$		
$clist ::= \text{empty}$		list of caveats

Εικόνα 6: Διαπιστευτήρια (Certificates), Ιδιότητες (Blessings) και Περιορισμοί (Caveats) [3]

Στην εικόνα 6 βλέπουμε από τι αποτελούνται τα Διαπιστευτήρια (C), οι Ιδιότητες (B) και οι Περιορισμοί (clist). Για τους Περιορισμούς θα γίνει αναφορά αργότερα, οπότε δεν θα εμβαθύνουμε ακόμα. Τα Διαπιστευτήρια αποτελούνται από τέσσερα πεδία, το όνομα (n), το δημόσιο κλειδί (pk), την λίστα των περιορισμών (clist) και την ψηφιακή υπογραφή (sig).

Οι ιδιότητες είναι μια λίστα Διαπιστευτηρίων, όπου κάθε διαπιστευτήριο χρησιμοποιείται για την επαλήθευση ενός κόμβου της Ιδιότητας. Για να γίνει πιο κατανοητό, μία Ιδιότητα είναι της μορφής Πάροχος:Ομάδα1:Ομάδα2:Κατηγορία1... Διαχωρίζοντας την Ιδιότητα στις άνω και κάτω τελείες καταλήγουμε στους κόμβους από τους οποίους αποτελείται. Κάθε κόμβος λοιπόν αντιστοιχίζεται με κάποιο Διαπιστευτήριο, ώστε να μπορεί να επιβεβαιωθεί. Εξαίρεση αποτελεί ο πρώτος κόμβος του Παρόχου, καθώς μπορεί να επιβεβαιωθεί από τα Διαπιστευτήρια του αμέσως επόμενου (στο παράδειγμα μας από τον κόμβο Ομάδα1).

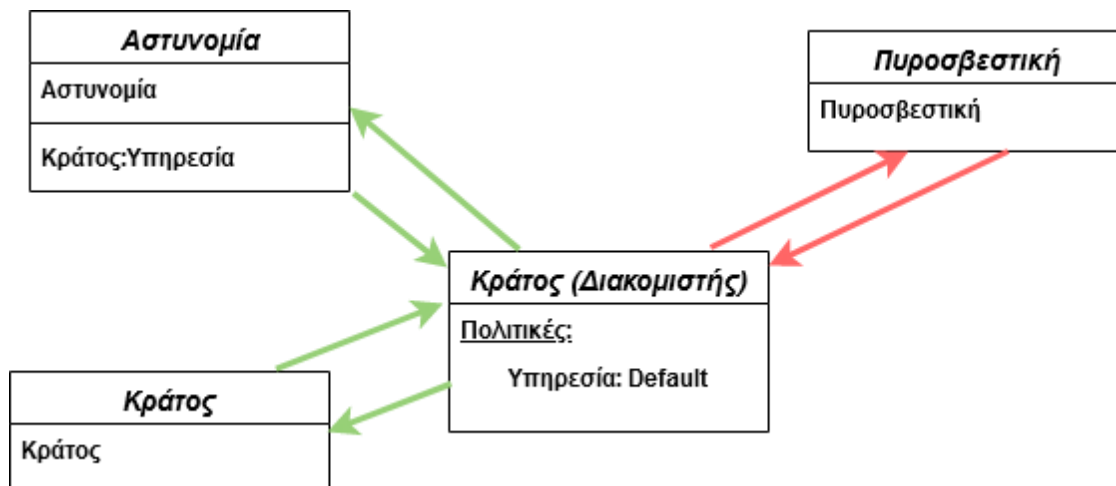
Για την ευκολότερη κατανόηση της χρήσης και αυθεντικοποίησης των Οντοτήτων και Ιδιοτήτων θα υλοποιήσουμε ένα απλό παράδειγμα. Ας υποθέσουμε ότι έχουμε ένα δίκτυο

Vanadium με τρεις τρεις οντότητες (*Principals*). Αυτές οι οντότητες είναι το **Κράτος**, η **Αστυνομία** και η **Πυροσβεστική**.

<b>Κράτος</b>	<b>Αστυνομία</b>	<b>Πυροσβεστική</b>
Κράτος	Αστυνομία	Πυροσβεστική
	Κράτος:Υπηρεσία	

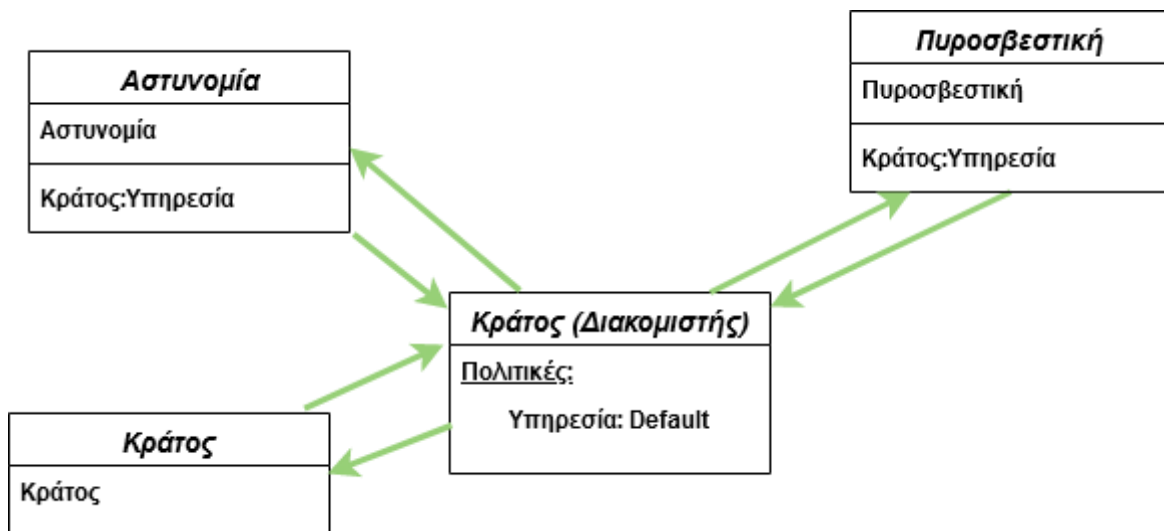
**Εικόνα 7:** Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος

Όπως αναφέραμε, στο δίκτυο του Vanadium, κάθε χρήστης είναι εν 'δυνάμει πελάτης και διακομιστής. Στο παράδειγμα μας, η οντότητα Κράτος έχει τον ρόλο του διακομιστή, οπότε οι άλλες οντότητες θα προσπαθήσουν να επικοινωνήσουν με αυτήν. Παρατηρούμε πως κάθε οντότητα έχει κάποιες ιδιότητες (*Blessings*). Η οντότητα Αστυνομία έχει τις ιδιότητες Αστυνομία και Κράτος:Υπηρεσία. Αντίστοιχα, η Πυροσβεστική έχει την ιδιότητα Πυροσβεστική. Μία οντότητα μπορεί να έχει διάφορες ιδιότητες, ανεξάρτητες μεταξύ τους, ώστε να επικοινωνεί ή αποκτά πρόσβαση σε διαφορετικά *Principals*. Για παράδειγμα η Αστυνομία και η Πυροσβεστική θα μπορούσαν να έχουν και την ιδιότητα Ένστολοι:Άμεση\_Δράση, η οποία θα ήταν ανεξάρτητη με την οντότητα του Κράτους. Βάση των προαναφερθέντων ιδιοτήτων, αποφασίζεται η πρόσβαση σε έναν διακομιστή. Ο διακομιστής αποφασίζει ποιος επιτρέπεται να έχει πρόσβαση, βάση των πολιτικών (*policies*) πρόσβασής του.



**Εικόνα 8:** Άρνηση Πρόσβασης ανάλογα με την Ιδιότητα (Blessing)

Στην προσομοίωση της εικόνας 8, παρατηρούμε πως ο διακομιστής επιτρέπει την επικοινωνία με τις οντότητες του Κράτους και της Αστυνομίας και αρνείται πρόσβαση στην οντότητα της Πυροσβεστικής. Το Κράτος έχει πρόσβαση μιας και πρόκειται για τον εαυτό του, όμως η Αστυνομία αποκτά πρόσβαση διαφορετικά. Βλέπουμε πως η Αστυνομία, αντίθετα με την Πυροσβεστική, έχει την ιδιότητα Κράτος:Υπηρεσία. Επίσης παρατηρούμε πως το Κράτος ως διακομιστής, έχει την πολιτική αυθεντικοποίησης Υπηρεσία. Ως αποτέλεσμα ο διακομιστής επιτρέπει σε οντότητες με ιδιότητα (Blessing) Κράτος:Υπηρεσία να επικοινωνούν μαζί του. Αν το Κράτος προσδώσει την ιδιότητα Κράτος:Υπηρεσία και στην οντότητα της Πυροσβεστικής, τότε θα αποκτήσει και αυτή πρόσβαση.

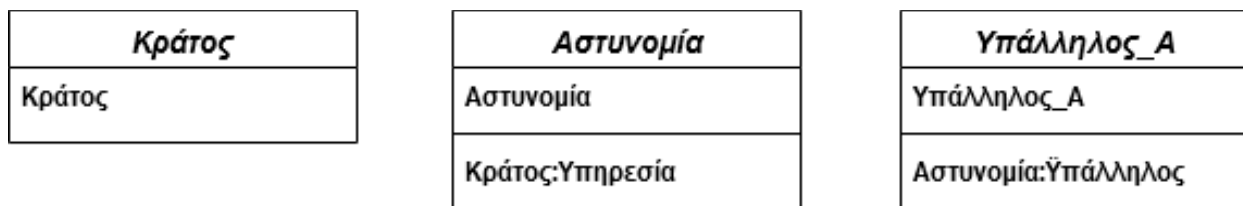


Εικόνα 9: Απόδοση Ιδιότητας και Άρση της Απαγόρευσης Πρόσβασης

### 3.2.3 Αναθέσεις

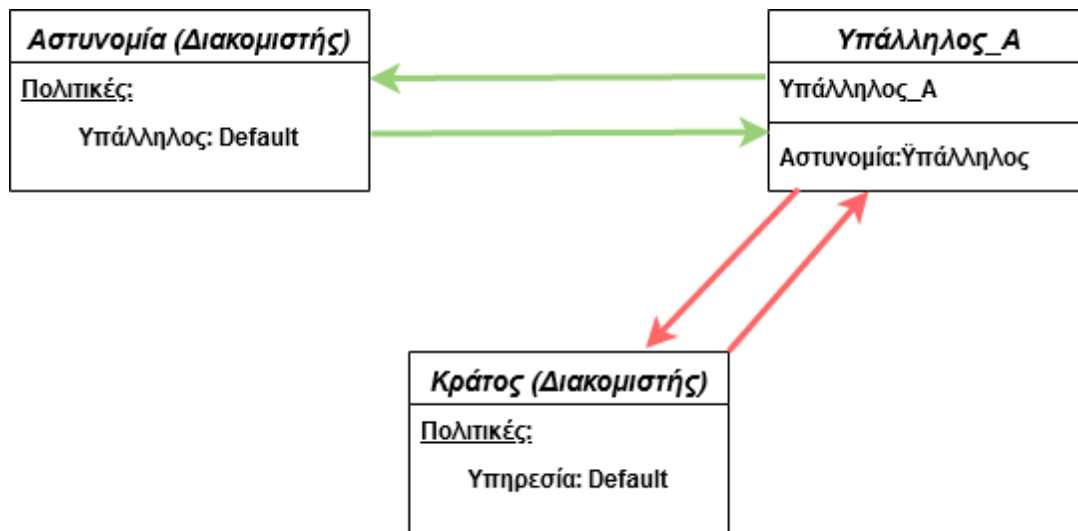
Στα πλαίσια του Vanadium, μία οντότητα μπορεί να μεταβιβάσει κάποια ιδιότητα της σε μία άλλη, με την διαδικασία της ανάθεσης. Η οντότητα που μεταβιβάζει την ιδιότητα, χρησιμοποιεί ένα νέο όνομα και το προσκολλά στην ήδη υπάρχουσα ιδιότητα.

Για παράδειγμα, ας συνεχίσουμε την δομή που είχαμε προηγουμένως με το Κράτος και την Αστυνομία. Αυτή την φορά όμως θα προσθέσουμε και έναν υπάλληλο που εργάζεται στην υπηρεσία της Αστυνομίας.



Εικόνα 10: Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος

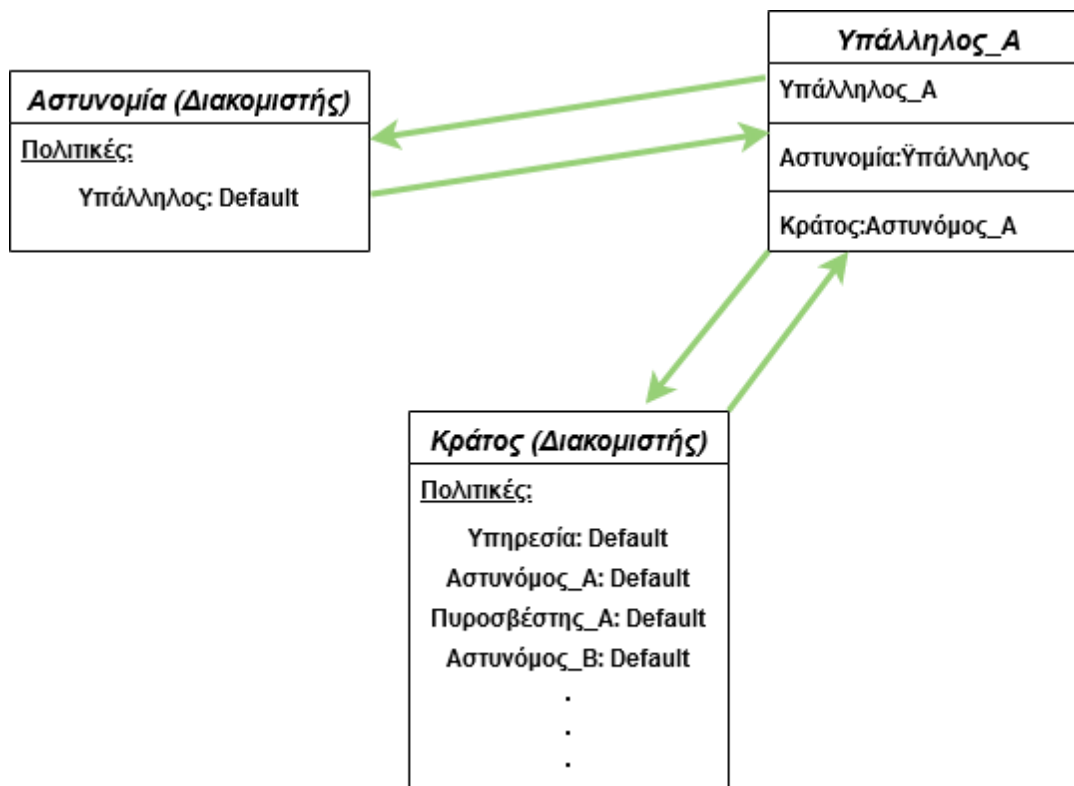
Ο υπάλληλος έχει δύο ιδιότητες, την Υπάλληλος\_A και την Αστυνομία:Υπάλληλος, καθώς εργάζεται στην οντότητα της Αστυνομίας. Στην περίπτωση που η Αστυνομία λειτουργήσει ως διακομιστής, θα περιμένουμε να έχει και την κατάλληλη πολιτική αυθεντικοποίησης ώστε να επιτρέπει στους υπαλλήλους της να έχουν πρόσβαση σε αυτή. Αν θεωρήσουμε ως πελάτη τον Υπάλληλο και ως διακομιστές το Κράτος και την Αστυνομία, έχουμε το παρακάτω αποτέλεσμα (εικόνα 11).



**Εικόνα 11:** Άρνηση Πρόσβασης ανάλογα με την Ιδιότητα (Blessing)

Δεδομένου ότι κάποιος υπάλληλος της Αστυνομίας, πιθανώς να χρειάζεται υπηρεσίες που προσφέρει το κράτος, θα πρέπει να αποκτήσει ιδιότητα (*Blessing*) ώστε να μπορεί να έχει πρόσβαση. Ο Υπάλληλος\_A θα μπορούσε να παραλάβει την ιδιότητα κατευθείαν από το Κράτος. Σε αυτήν την περίπτωση το Κράτος θα έπρεπε να δημιουργήσει μια νέα πολιτική αυθεντικοποίησης για τους Υπαλλήλους της Αστυνομίας. Χρησιμοποιώντας αυτή την λογική, η οντότητα Κράτος θα έπρεπε να δίνει *Blessing* σε κάθε νέο Υπάλληλο. Επίσης το Κράτος θα έπρεπε να δημιουργεί μια νέα πολιτική για κάθε τύπο Υπαλλήλου κάθε Υπηρεσίας.





**Εικόνα 12:** Απόδοση Ιδιότητας και Άρση της Απαγόρευσης Πρόσβασης

Δεδομένου ότι μία Υπηρεσία μπορεί να έχει διαφορετικούς τύπους Υπαλλήλων, βλέπουμε πως η πολυπλοκότητα και η εξαρτήσεις από μια οντότητα σε μία άλλη αυξάνονται. Επιπρόσθετα οι εξαρτήσεις δεν είναι σωστές, καθώς κάποιος Υπάλληλος εξαρτάτε από το Κράτος και όχι από την Υπηρεσία στην οποία εργάζεται.

Η διαδικασία της ανάθεσης, έρχεται να λύσει αυτό το πρόβλημα. Αντί ο Υπάλληλος να παίρνει ιδιότητα από το Κράτος, του μεταβιβάζεται μία από την υπηρεσία που δουλεύει. (στην προκειμένη περίπτωση η Αστυνομία). Η Αστυνομία επομένως, αναθέτει στον Υπάλληλο την ιδιότητα Κράτος:Υπηρεσία:Αστυνομία:Τύπος\_A.

Κράτος	Αστυνομία	Υπάλληλος_A
Κράτος	Αστυνομία	Υπάλληλος_A
	Κράτος:Υπηρεσία	Αστυνομία:Υπάλληλος
		Κράτος:Υπηρεσία:Αστυνομία:Τύπος_A

Εικόνα 13: Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος

Ο υπάλληλος, έχοντας πλέον την νέα του ιδιότητα, μπορεί να επικοινωνήσει με το Κράτος. Δεδομένου ότι το Κράτος έχει φροντίσει να έχει κατάλληλες πολιτικές αυθεντικοποίησης για την οντότητα της Αστυνομίας, θα επιτρέψει στον Υπάλληλο να έχει πρόσβαση.

### 3.2.4 Περιορισμοί

Με την παροχή της δυνατότητας ανάθεσης δικαιωμάτων σε τρίτους μέσω της μεταφοράς Ιδιότητας, προκύπτει θέμα ασφάλειας. Θα μπορούσε ο Υπάλληλος να έχει πρόσβαση σε όλες της δυνατότητες που δίνει το Κράτος στην Αστυνομία, και θα έπρεπε αυτό να συμβαίνει; Αυτόν τον προβληματισμό έρχονται να επιλύσουν οι Περιορισμοί (*Caveats*). Οι Περιορισμοί δίνονται από ένα *Principal* σε ένα άλλο κατά την διαδικασία της ανάθεσης. Η λειτουργικότητα που προσφέρουν είναι συγκεκριμένη και ο κύριος σκοπός τους είναι η διασφάλιση της σωστή λειτουργίας ενός *Principal* και των εξαρτήσεών του. Οι περιορισμοί μειώνουν τα δικαιώματα της οντότητας στην οποία έχει γίνει η ανάθεση.

$clist ::= \text{empty}$   $clist : c$	list of caveats
$c ::= fc$   $tc$	caveats
$fc ::= \text{timeCaveat}$   $\text{targetCaveat}$   $\dots$	first-party caveats
$tc ::= T\langle nonce, pk, fc, loc \rangle$	third-party caveats
$d ::= D\langle clist, sig \rangle$	discharges

Εικόνα 14: Ιεράρχηση Περιορισμών (Caveats)

Όπως παρατηρούμε στην *εικόνα 14*, οι περιορισμοί είναι μέρος μίας λίστας. Σε αντίθεση με τα πεδία τιμών που είχαμε δει προηγουμένως, η λίστα των περιορισμών μπορεί να είναι κενή. Σε αυτή την περίπτωση, η οντότητα που κάνει ανάθεση της Ιδιότητας της, μεταφέρει επίσης και όλα τα δικαιώματα που τις έχουν δοθεί.

Κράτος	Αστυνομία	Υπάλληλος_A
Κράτος	Αστυνομία	Υπάλληλος_A
	Κράτος:Υπηρεσία	Αστυνομία:Υπάλληλος
		Κράτος:Υπηρεσία:Αστυνομία:Τύπος_A

Εικόνα 15: Οι Οντότητες (με τις αντίστοιχες νέες ιδιότητες) Παραδείγματος

Συνεχίζοντας το προηγούμενο παράδειγμα, θεωρούμε πως το *Principal* Αστυνομία, θέλει να περιορίσει τον Υπάλληλο\_A (*εικόνα 15*). Μπορεί να θέσει σαν Περιορισμό (Caveat) ένα ωράριο εργασίας, οπού μετά το πέρας κάποιας ώρας, η ιδιότητα Κράτος:Υπηρεσία:Αστυνομία:Τύπος\_A που έχει αναθέσει στον Υπάλληλο, παύει να ισχύει. Επίσης μπορεί να περιορίσει την πρόσβαση ενός κατόχου της ιδιότητας, σε συγκεκριμένες

λειτουργικότητες του Principal, όπου όταν μία αίτηση στο Κράτος δεν είναι στις επιτρεπτές, να απορρίπτεται.

Η λειτουργία των περιορισμών είναι ιδιαίτερα χρήσιμη, καθώς μπορεί να χρησιμοποιεί και τεχνικές απομακρυσμένης αδειοδότησης. Σε αυτή την περίπτωση η αρμοδιότητα ελέγχου και επιβεβαίωσης του περιορισμού, δίνεται σε κάποιον τρίτο. Για παράδειγμα η ιδιότητα του Υπάλληλου\_A θα μπορούσε να είναι ενεργή μόνο όταν ο ίδιος βρίσκεται μέσα στο κτήριο της Αστυνομίας. Για να επιβεβαιωθεί αυτό, θα έπρεπε να υπάρχει ένα σύστημα, με τον ρόλο του τρίτου, που θα παρακολουθεί την τοποθεσία του Υπάλληλου σε σχέση με το κτήριο. Όταν λοιπόν ο Υπάλληλος κάνει μία αίτηση προς το Κράτος, θα πρέπει να πάρει μία επιβεβαίωση (*discharge*) ότι είναι εντός των επιτρεπτών ορίων. Αυτή η επιβεβαίωση μπορεί να έχει περιορισμούς, για να αποφευχθεί η λανθασμένη χρήση της. Για παράδειγμα θα μπορεί να έχει χρονικό περιθώριο εγκυρότητας χρήσης, ή περιορισμένο αριθμό χρήσεων.

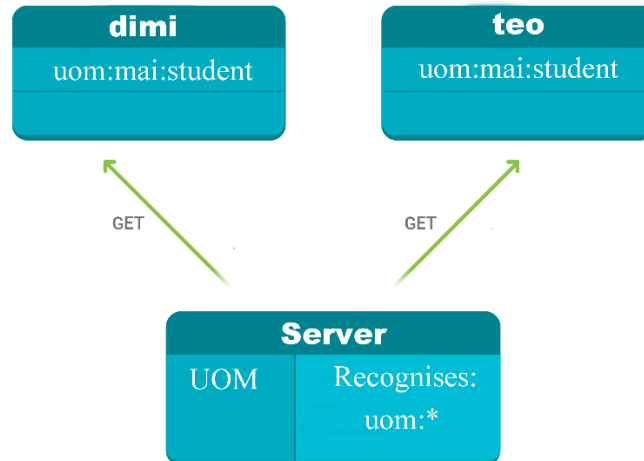
### 3.2.5 Επαλήθευση των Blessings

**Ένα *blessing* θεωρείται έγκυρο στο πλαίσιο ενός RPC εάν ισχύουν όλες οι παρακάτω προϋποθέσεις:**

- Το *blessing* είναι κρυπτογραφικά έγκυρο, δηλαδή, κάθε πιστοποιητικό στην αλυσίδα πιστοποιητικών του συγκεκριμένου *blessing* έχει έγκυρη υπογραφή
- Όλοι οι περιορισμοί (*caveats*) που σχετίζονται με το *blessing* είναι έγκυροι στο πλαίσιο του RPC
- Το *blessing* αναγνωρίζεται από τον διακομιστή

Ένα *blessing* αναγνωρίζεται από μια εφαρμογή *Vanadium* εάν και μόνο εάν η εφαρμογή θεωρεί τη ρίζα του *blessing* (*blessing root*) ως αξιόπιστη για το όνομα του *blessing*. (Ρίζα του *blessing* αποτελεί το δημόσιο κλειδί του πρώτου πιστοποιητικού στην αλυσίδα πιστοποιητικών του *blessing*)

Για παράδειγμα, ένας διακομιστής μπορεί να θεωρήσει τη ρίζα  $P_{UOM}$  ως έγκυρη σε όλα τα ονόματα των *blessings* που ξεκινούν με το  $UOM$ . Μια τέτοια εφαρμογή θα αναγνώριζε τότε το *blessing*  $UOM:mai:student$  αν έχει τις ρίζες του στο  $P_{UOM}$ , όπως φαίνεται στην εικόνα 4-2-5-1.



Εικόνα 16: Παράδειγμα Αναγνώρισης Blessing από τον Διακομιστή

Σε αυτή την περίπτωση, οι 2 πελάτες (*dimi* και *teo*) πραγματοποιούν αίτημα τύπου *GET*. Λόγω του ότι ο διακομιστής (server) αναγνωρίζει όλα τα ονόματα των *blessings* που ξεκινούν με  $UOM$ , το αίτημά και των δύο πελατών γίνεται δεκτό. Στην περίπτωση του παραδείγματος που παραθέσαμε, ο διακομιστής αναγνωρίζει μόνο αιτήματα τύπου  $uom:*$ , κάτι το οποίο δεν είναι σε καμία περίπτωση δεσμευτικό. Αντίστοιχα, θα μπορούσε να αναγνωρίζει *blessings* και ξεκινούν με άλλο αλφαριθμητικό, π.χ. *auth:\**.

### 3.2.6 Αυθεντικοποίηση

Η επικοινωνία μεταξύ δύο οντοτήτων πραγματοποιείται αφού δημιουργηθεί μια έμπιστη, επικυρωμένη σύνδεση. Η κρυπτογράφηση πραγματοποιείται με κλειδιά που προέρχονται από ανταλλαγή κλειδιών με χρήση ελλειπτικών καμπυλών *Diffie-Hellman (ECDH)*. Η διαδικασία αυτή χρησιμοποιείται για την παροχή εμπιστευτικότητας και ακεραιότητας μηνυμάτων. Ο στόχος του πρωτοκόλλου ελέγχου ταυτότητας είναι η ανταλλαγή των *blessings* με τρόπο που παρέχει τις ακόλουθες ιδιότητες:

- **Εγκαθίδρυση συνεδρίας:** Το ιδιωτικό μέρος  $S$  του δημόσιου κλειδιού  $P$ , στο οποίο υφίστανται τα *blessings*, πρέπει να κατέχεται από την ίδια διεργασία της εφαρμογής *Vanadium*, με την οποία έγινε η εγκαθίδρυση κλειδιών συνεδρίας (με την διαδικασία που περιγράψαμε παραπάνω).
- **Απόρρητο πελάτη:** Ένας ενεργός ή παθητικός εισβολέας δικτύου που ακούει κάθε επικοινωνία μεταξύ των δύο διαδικασιών δεν μπορεί να καθορίσει το σύνολο των *blessings* που παρουσίασε ο η μία διεργασία στην άλλη.

#### Λεπτομέρειες του Πρωτοκόλλου:

- **Ο διακομιστής στέλνει πρώτος τα *blessings*:** Εάν ο πελάτης έστειλε τις ευλογίες του πριν από την επικύρωση και την έγκριση των *blessings* του διακομιστή, τότε ένας επιτιθέμενος θα μπορούσε να μάθει τα *blessings* του πελάτη και να θέσει σε κίνδυνο το απόρρητό του.
- **Όλα τα *blessings* κρυπτογραφούνται με το κλειδί συνεδρίας:** Με τον τρόπο αυτό και οι δύο πλευρές (πελάτης, διακομιστής) αποδεικνύουν ότι υπάρχει αμφίπλευρη γνώση του κλειδιού.

### 3.2.6 Εξουσιοδότηση

Σε μια κλήση απομακρυσμένης διαδικασίας (RPC call), πρέπει να ληφθούν δύο αποφάσεις, μία από κάθε πλευρά, πριν γίνει εξουσιοδότηση:

- **Ο πελάτης εμπιστεύεται τον διακομιστή αρκετά για να καλέσει μία απομακρυσμένη διαδικασία;**
- **Επιτρέπει ο διακομιστής τον πελάτη να καλέσει μια μέθοδο σε ένα αντικείμενο με τα παρεχόμενα ορίσματα;**

Στο πλαίσιο του *Vanadium*, η εξουσιοδότηση βασίζεται σε επικυρωμένα ονόματα *blessing* (blessing names) και είναι **αμφίδρομη**.

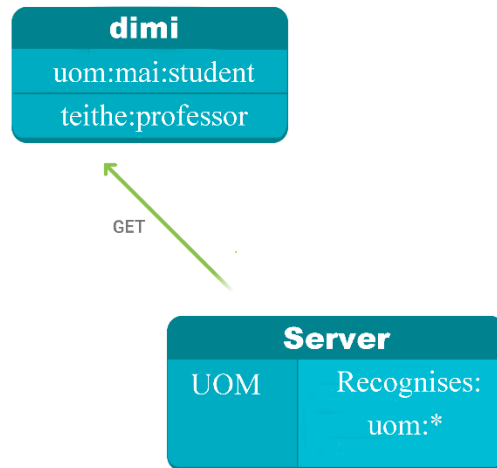
Αυτό πρακτικά σημαίνει ότι ένας πελάτης θέλει να καλεί μια συγκεκριμένη (απομακρυσμένη) μέθοδο μιας υπηρεσίας, μόνο εάν ο διακομιστής κατέχει ένα *blessing*, το οποίο

είναι συγκεκριμένης μορφής (π.χ. *UOM:mai:crypto*). Ομοίως, μια υπηρεσία μπορεί να επιτρέπει σε έναν πελάτη να καλεί μια (απομακρυσμένη) μέθοδο, μόνο εάν ο πελάτης κατέχει ένα *blessing*, το οποίο είναι συγκεκριμένης μορφής (π.χ. *UOM:mai:student*).

**Τα δημόσια κλειδιά των *principal* του πελάτη και του διακομιστή δε θα ληφθούν υπόψη από την στιγμή που υφίσταται *blessing* με ένα έγκυρο όνομα που αντιστοιχεί στην πολιτική εξουσιοδότησης του άλλου άκρου. Κάθε άκρο επιβεβαιώνει το έγκυρο όνομα του *blessing* του άλλου άκρου, επικυρώνοντας όλους τους περιορισμούς που σχετίζονται με το *blessing* και επαληθεύοντας ότι το συγκεκριμένο *blessing* όντως αναγνωρίζεται.**

**Ένα *principal* μπορεί να έχει συλλέξει πολλά *blessings*. Αντί να παρουσιάζει το σύνολο των *blessings* (όλο τον κατάλογο) κάθε φορά που γίνεται αυθεντικοποίηση, με κόστος διαρροής ευαίσθητων πληροφοριών, γίνεται επιλεκτική διαμοίραση των *blessings* που βρίσκονται στον κατάλογο, αναλόγως την ταυτότητα του άλλου άκρου.**

Ας δούμε το παράδειγμα της εικόνας 4-2-6-1, στο οποίο ο πελάτης *dimi*, έχει, πλέον, και το *blessing teithe:professor* στον κατάλογο των *blessings* του, μαζί με το *blessing uom:mai:student*, το οποίο ήδη κατείχε. Με βάση την δομή του *Vanadium*, ο διακομιστής που διαχειρίζεται το Πανεπιστήμιο Μακεδονίας (*UOM*) θα είναι σε θέση να διαβάσει μόνο το *blessing uom:mai:student* (και όχι το *teithe:professor*), όταν ο *dimi* του στείλει κάποιο αίτημα. Οποιοσδήποτε άλλος διακομιστής που επικοινωνεί με τον *dimi* δεν θα γνωρίζει ότι έχει αυτό το *blessing* από τον διακομιστή *UOM*. Κατ' αντιστοιχία, κανένας άλλος διακομιστής δε θα είναι σε θέση να γνωρίζει ότι ο *dimi* κατέχει *blessing* από τον διακομιστή *teithe*, εκτός από εκείνους που πρέπει να αυθεντικοποιηθεί με το *blessing teithe:professor*.



**Εικόνα 17:** Παράδειγμα Επιλεκτικής Διαμοίρασης των Blessings

(Ο διακομιστής βλέπει μόνο το *uom:mai:student*)

## 4 Υλοποίηση

Στο παρόν κεφάλαιο, για λόγους επίδειξης του πλαισίου *Vanadium*, θα παρουσιαστεί μία εφαρμογή γραμμής εντολών ανταλλαγής άμεσων μηνυμάτων (chat app). Με τον τρόπο αυτό, θα γίνει αντιληπτό πώς μπορεί κανείς να χρησιμοποιήσει βασικές λειτουργίες που παρουσιάστηκαν στα προηγούμενα κεφάλαια. Η υλοποίηση είναι διαθέσιμη σε αποθετήριο στο GitHub:

[https://github.com/teomaik/MSc\\_Crypto\\_ergasia](https://github.com/teomaik/MSc_Crypto_ergasia)

### 4.1 Εφαρμογή Ανταλλαγής Άμεσων Μηνυμάτων

Στην συγκεκριμένη εφαρμογή, για λόγους απλότητας, θεωρούμε ότι τόσο ο διακομιστής, όσο και ο πελάτης στέλνουν ένα μήνυμα τη φορά, δηλαδή η επικοινωνία τους είναι blocking.

#### 4.1.1 Κώδικας

##### 4.1.1.1 VDL



Προτού αρχίσουμε να παρουσιάζουμε κάποιο κώδικα, θα κάνουμε χρήση του *Vanadium Definition Language*, προκειμένου να δημιουργήσουμε τον «σκελετό» της υπηρεσίας, την οποία πρόκειται να δημιουργήσουμε. Προφανώς, πρακτικά δεν θα είχε νόημα η χρήση του για ένα τόσο μικρό παράδειγμα. Η δημιουργία της διεπαφής της υπηρεσίας γίνεται ως εξής (αρχείο *chat.vdl*):

```
/* Το Chat είναι η διεπαφή για την υπηρεσία άμεσων
   μηνυμάτων */
type Chat interface {

    /* Η SendMessage() στέλνει ένα μήνυμα στον διακομιστή
       (όρισμα msg τύπου string) και επιστρέφει μία απάντηση
       τύπου string (reply) */
    SendMessage(msg string) (reply string | error)

}
```

Η παραπάνω πληροφορία, αφορά τόσο τον πελάτη, όσο και τον διακομιστή, αφού δίνοντάς την ως είσοδο στο εργαλείο VDL (VDL Tool), παράγει κώδικα (ανάλογα με την γλώσσα της επιλογής μας, επιλέξαμε *Go*), ο οποίος αντιστοιχεί στον κώδικα στελέχους (stub code) του πελάτη και του διακομιστή (αρχείο *chat.vdl.go*, παραλείπεται).

#### 4.1.1.2 Διακομιστής

##### Main

Καθώς πρόκειται για απλουστευμένη εφαρμογή, για την σύνταξη του κώδικα του διακομιστή θα χρειαστούμε 2 συστατικά (με βάση την περιγραφή που κάναμε στο [κεφάλαιο 4.1](#)):

- Έναν εξουσιοδοτούντα (authorizer)
- Μια υπηρεσία (service) που θα περιέχει τις υλοποιήσεις

```

ctx, shutdown := v23.Init()
defer shutdown()

authorizer := security.DefaultAuthorizer() /* Παίρνουμε τον απλό
εξουσιοδοτούντα */
impl := internal.NewImpl() /* Το αντικείμενο impl περιέχει τις υλοποιήσεις
*/
service := chat.ChatServer(impl) /* Περνάμε ως παράμετρο στο service τις
υλοποιήσεις */
ctx, server, err := v23.WithNewServer(ctx, *name, service, authorizer) /*
Δημιουργία του server με τα 2 απαραίτητα συστατικά */
if err != nil {
    log.Panic("Failure creating server: ", err)
}
log.Printf("Listening at: %v\n\n\n", server.Status().Endpoints[0])

<-signals.ShutdownOnSignals(ctx)

```

## Υλοποίηση

Η υλοποίηση του μοναδικού service της εφαρμογής (ChatService), βρίσκεται στον υποκατάλογο του διακομιστή, internal. Αυτό αποτελεί ένα standard της γλώσσας Go, δηλαδή όταν επιθυμούμε να «κρύψουμε» τις υλοποιήσεις, τις τοποθετούμε σε ένα εσωτερικό πακέτο (internal package).

```

type impl struct {
}

func NewImpl() chat.ChatServerMethods {
    return &impl{}
}

/* Η μοναδική διαθέσιμη μέθοδος, αυτή θα καλέσει ο εκάστοτε πελάτης */
func (f *impl) SendMessage(_ *context.T, _ rpc.ServerCall, msg string)
(string, error) {
    log.Printf("%v", msg) /* Εμφάνισε το μήνυμα στο παράθυρο το
διακομιστή */
    reply := getInput() /* Μπλόκαρε μέχρι να δεχθείς είσοδο από το
πληκτρολόγιο */
    return reply, nil /* Στείλε απάντηση */
}

```

Όπως παρατηρούμε, το *impl* που περάσαμε ως όρισμα στο service (στην [Main](#)) αποτελεί μία κενή δομή. Αυτό συμβαίνει, καθώς στην υλοποίηση που κάναμε δεν έχουμε κάποιου είδους

δεδομένα (π.χ. πίνακας με το ιστορικό συνομιλίας). Η υλοποίηση περιέχει μόνο μία μέθοδο: την *SendMessage()* (αγνοούμε την υλοποίηση της βοηθητικής συνάρτησης *get\_input()*).

Σε ό,τι αφορά την μέθοδο αυτή, δέχεται ως μοναδικό όρισμα από τον χρήστη ένα μήνυμα τύπου *string* (τα υπόλοιπα ορίσματα αναλαμβάνει να τα συμπληρώσει η βιβλιοθήκη του *Vanadium RPC* κατά την απομακρυσμένη κλήση).

#### 4.1.1.3 Πελάτης

Ο κώδικας του πελάτη αποτελείται από μόνο ένα αρχείο (το *main.go*) και είναι παρόμοιος με τον κύριο κώδικα του διακομιστή:

```
var (
    serverName = flag.String("serverName", "/127.0.0.1:2507", "Name of
the server to connect to") /* Κάθε φορά προσδιορίζουμε την διεύθυνση IP και
την πόρτα του διακομιστή */
)

func main() {

    ctx, shutdown := v23.Init()
    defer shutdown()

    client := chat.ChatClient(*serverName) /* Δημιουργία Client */

    ctx, cancel := context.WithTimeout(ctx, time.Minute)
    defer cancel()

    for true { /* Μέχρι ο πελάτης να αποσυνδεθεί (CTRL+C), δέξου είσοδο
*/
        var msg string = getInput()
        value, _ := client.SendMessage(ctx, msg) /* Η απομακρυσμένη
κλήση μεθόδου, μπλοκάρει μέχρι να δεχθεί πίσω απάντηση (value) */
        log.Printf("%v", value) /* Εμφάνιση απάντησης */
    }
}
```

Η λογική της ροής του κώδικα είναι αρκετά απλή:

- Δημιουργούμε έναν πελάτη τύπου *ChatClient*, συνδεόμαστε σε έναν διακομιστή, του

οποίου τα στοιχεία τα έχουμε προκαθορίσει (πριν την εκτέλεση)

- Εφόσον βρεθεί ο διακομιστής και γίνει επιτυχής σύνδεση, μπορούμε να στέλνουμε μηνύματα στον διακομιστή, περιμένοντας να δεχθούμε σε κάθε μήνυμα μια απάντηση από τον διακομιστή (αφού κάναμε την παραδοχή της blocking επικοινωνίας). (\*)

(\*): Προκειμένου να στέλνουμε μηνύματα στον διακομιστή, δεν αρκεί μόνο να γίνει επιτυχής σύνδεση, αλλά και να μας αναγνωρίσει ο εξουσιοδοτών (*authorizer*) του διακομιστή, την οποία επιθυμούμε να χρησιμοποιήσουμε (το *chat service*). Επειδή ο διακομιστής, όπως υλοποιήθηκε, χρησιμοποιεί τον προεπιλεγμένο εξουσιοδοτούντα (*Default Authorizer*), χρειάζεται να διαθέτουμε *blessing* από τον διακομιστή, προκειμένου να είναι σε θέση να μας αναγνωρίσει ο εξουσιοδοτών.

## 4.1.2 Εκτέλεση

Για να εκτελέσουμε χωρίς προβλήματα τις παρακάτω εντολές, θα πρέπει να βρισκόμαστε στον υποκατάλογο *chat*, που προηγείται τόσο αυτόν του διακομιστή, όσο και αυτόν του πελάτη. Επίσης, η κλωνοποίηση του αποθετηρίου πρέπει να γίνει στο κεντρικό κατάλογο του χρήστη (*/home/<user>*).

### 4.1.2.1 Διακομιστής

Καταρχάς, χρειάζεται να δημιουργήσουμε ένα *principal* για τον διακομιστή (άπαξ το κάνουμε αυτό), με την ακόλουθη εντολή:

```
go run v.io/x/ref/cmd/principal create --with-passphrase=false --  
overwrite ~/MSc_Crypto_ergasia/vanadium-demo/chat/server/cred/dimi dimi
```

Από την στιγμή που τρέξουμε αυτή την εντολή, έχουμε δημιουργήσει, ουσιαστικά, ένα ζεύγος κλειδιών (δημόσιου, ιδιωτικού), το οποίο είναι άρρηκτα συνδεδεμένο με το όνομα (χρήστη) *dimi*. Πάνω σε αυτό το *principal*, μπορούμε να συνδέσουμε και διάφορα *blessings*. Όλα τα αρχεία που αφορούν *principals* και *blessings* αποθηκεύονται στον υποκατάλογο *cred*.

Εμείς θα δημιουργήσουμε ένα μόνο *blessing* της μορφής *friend:<name>*, το οποίο θα αφορά το όνομα/χρήστη *teo*. Αυτό μπορούμε να το πετύχουμε με την ακόλουθη εντολή:

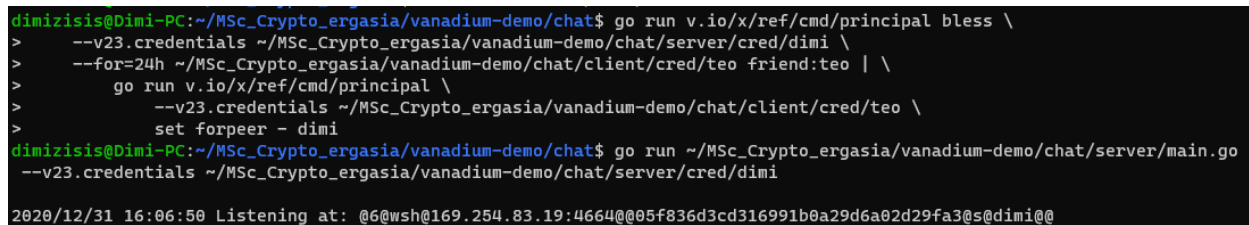
```
go run v.io/x/ref/cmd/principal bless \
  --v23.credentials ~/MSc_Crypto_ergasia/vanadium-
demo/chat/server/cred/dimi \
  --for=24h ~/MSc_Crypto_ergasia/vanadium-demo/chat/client/cred/teo
friend:teo
```

Επομένως, έχουμε δημιουργήσει ένα *blessing* για τον χρήστη *teo*, το οποίο θα είναι **σε ισχύ για 24 ώρες**.

Πλέον, είμαστε σε θέση να τρέξουμε τον διακομιστή μας, ώστε να ακούει σε μια συγκεκριμένη διεύθυνση IP και θύρα:

```
go run ~/MSc_Crypto_ergasia/vanadium-demo/chat/server/main.go --
v23.credentials ~/MSc_Crypto_ergasia/vanadium-demo/chat/server/cred/dimi
```

Μετά την εκτέλεση των εντολών αυτών, θα πρέπει ο διακομιστής να είναι σε θέση να δεχτεί αιτήματα:



```
dimizisis@Dimi-PC:~/MSc_Crypto_ergasia/vanadium-demo/chat$ go run v.io/x/ref/cmd/principal bless \
> --v23.credentials ~/MSc_Crypto_ergasia/vanadium-demo/chat/server/cred/dimi \
> --for=24h ~/MSc_Crypto_ergasia/vanadium-demo/chat/client/cred/teo friend:teo | \
> go run v.io/x/ref/cmd/principal \
> --v23.credentials ~/MSc_Crypto_ergasia/vanadium-demo/chat/client/cred/teo \
> set forpeer - dimi
dimizisis@Dimi-PC:~/MSc_Crypto_ergasia/vanadium-demo/chat$ go run ~/MSc_Crypto_ergasia/vanadium-demo/chat/server/main.go
--v23.credentials ~/MSc_Crypto_ergasia/vanadium-demo/chat/server/cred/dimi
2020/12/31 16:06:50 Listening at: @6@wsh@169.254.83.19:4664@@@05f836d3cd316991b0a29d6a02d29fa3@s@dimi@@
```

Εικόνα 18: Στιγμιότυπο Οθόνης Γραμμής Εντολών Διακομιστή

#### 4.1.2.2 Πελάτης

Για τον πελάτη η διαδικασία είναι ακόμα απλούστερη, αφού χρειάζεται μόνο να δημιουργήσουμε το *principal* (ζεύγους δημόσιου-ιδιωτικού κλειδιού), το οποίο θα είναι «δεμένο» με το όνομα *teo*. Αυτό το πετυχαίνουμε με την ακόλουθη εντολή:

```
go run v.io/x/ref/cmd/principal \
  --v23.credentials ~/MSc_Crypto_ergasia/vanadium-
demo/chat/client/cred/teo
```

Τέλος, για να εκκινήσουμε τον πελάτη, τρέχουμε την εντολή:

```
go run ~/MSc_Crypto_ergasia/vanadium-demo/chat/client/main.go --  
v23.credentials ~/MSc_Crypto_ergasia/vanadium-demo/chat/client/cred/teo
```

Και η γραμμή εντολών του πελάτη θα πρέπει να εμφανίσει το μήνυμα επιτυχής σύνδεσης:

```
dimizisis@Dimi-PC:~/MSc_Crypto_ergasia/vanadium-demo/chat$ go run ~/MSc_Crypto_ergasia/vanadium-demo/chat/client/main.go  
--v23.credentials ~/MSc_Crypto_ergasia/vanadium-demo/chat/client/cred/teo  
2020/12/31 16:31:34 Connected!
```

Εικόνα 19: Στιγμιότυπο Οθόνης Γραμμής Εντολών Πελάτη

Πλέον, μπορούμε να στέλνουμε (blocking) μηνύματα ανάμεσα σε πελάτη και διακομιστή.

## 5 Συμπεράσματα

Στην συγκεκριμένη εργασία, παρουσιάστηκε το *Vanadium*, ένα ολοκληρωμένο πλαίσιο κατασκευής ασφαλών κατανεμημένων εφαρμογών. Αρχικά, παρουσιάσαμε τις τεχνολογίες και τα πρωτόκολλα που χρησιμοποιεί το *Vanadium*, σε ένα γενικότερο πλαίσιο. Περιγράψαμε, στη συνέχεια, ορισμένα βασικά σενάρια επικοινωνίας εφαρμογών *Vanadium*, παρουσιάζοντας και αντίστοιχα αντιπροσωπευτικά παραδείγματα. Τέλος, υλοποιήσαμε μία μικρή εφαρμογή επίδειξης, με στόχο να δείξουμε την απλότητα χρήσης του συγκεκριμένου ολοκληρωμένου πλαισίου, αλλά και ένα από τα βασικά σενάρια αυθεντικοποίησης (με principal, blessings). Μπορούμε, συνεπώς, να ισχυριστούμε ότι μέσω της παρούσας εργασίας γνωρίσαμε σε σημαντικό βαθμό τον τρόπο με τον οποίο λειτουργεί το συγκεκριμένο ολοκληρωμένο πλαίσιο, το οποίο αποτελεί μία τεχνολογία που σίγουρα θα άξιζε κανείς να ασχοληθεί.

# Βιβλιογραφία

- [1] “The OAuth 2.0 Authorization Framework.” <https://tools.ietf.org/id/draft-ietf-oauth-v2-31.html> (accessed Jan. 23, 2021).
- [2] “Using OAuth 2.0 to Access Google APIs | Google Identity.” <https://developers.google.com/identity/protocols/oauth2> (accessed Jan. 23, 2021).
- [3] A. Erbsen, A. Shankar, and A. Taly, “Distributed Authorization in Vanadium,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9808 LNCS, pp. 139–162, Jul. 2016, Accessed: Jan. 17, 2021. [Online]. Available: <http://arxiv.org/abs/1607.02192>.
- [4] “The Official PGP User’s Guide: Zimmermann, Philip R.: 9780262740173: Amazon.com: Books.” <https://www.amazon.com/Official-PGP-Users-Guide/dp/0262740176> (accessed Jan. 23, 2021).
- [5] “RFC 4880 - OpenPGP Message Format.” <https://tools.ietf.org/html/rfc4880> (accessed Jan. 23, 2021).
- [6] C. M. Ellison, B. Frantz, B. Thomas, T. Ylonen, R. Rivest, and B. Lampson, “SPKI Certificate Theory.”
- [7] R. Canetti and H. Krawczyk, “Security analysis of IKE’s signature-based key-exchange protocol,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2002, vol. 2442, pp. 143–161, doi: 10.1007/3-540-45708-9\_10.
- [8] H. Zhang, “Architecture of Network and Client-Server model,” Jul. 2013, Accessed: Jan. 17, 2021. [Online]. Available: <http://arxiv.org/abs/1307.6665>.
- [9] D. L. Rosenband, “A Remote Procedure Call Library,” 1997, [Online]. Available: <http://web.mit.edu/6.033/1997/reports/dp1-danlief.html>.
- [10] “Vanadium.github.io.” <https://vanadium.github.io/core.html>.