Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Ανάλυση Πρωτοκόλλων Ηλεκτρονικής Ψηφοφορίας με την ProVerif

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΙΟΥ ΧΡΥΣΟΣΤΟΜΟΣ

Επιβλέπων: Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2020

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Ανάλυση Πρωτοκόλλων Ηλεκτρονικής Ψηφοφορίας με την ProVerif

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΙΟΥ ΧΡΥΣΟΣΤΟΜΟΣ

Επιβλέπων: Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Φεβρουαρίου 2020.

.....	.....	.....
Νικόλαος Σ. Παπασπύρου	Αριστείδης Θ. Παγουρτζής	Πέτρος Στεφανέας
Καθηγητής Ε.Μ.Π.	Αναπληρωτής Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2020

.....
Χίου Χρυσόστομος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της παρούσας εργασίας είναι η μελέτη ιδιοτήτων ασφαλείας κρυπτογραφικών πρωτοκόλλων μέσω τυπικών μεθόδων και εργαλείων αυτόματης ανάλυσης. Πιο συγκεκριμένα, θα εξετάσουμε τον Εφαρμοσμένο π-Λογισμό και το εργαλείο αυτόματης ανάλυσης ProVerif που βασίζεται σε αυτόν, προκειμένου να μελετήσουμε ιδιότητες ασφαλείας των πρωτοκόλλων ηλεκτρονικής ψηφοφορίας. Επιλέξαμε ένα νέο πρωτόκολλο ηλεκτρονικής ψηφοφορίας που προτάθηκε πρόσφατα στη βιβλιογραφία και επιτυγχάνει ισχυρές ιδιότητες ασφαλείας. Κατασκευάσαμε σενάρια στην ProVerif που μοντελοποιούν το πρωτόκολλο, προκειμένου να αποδείξουμε αυτόματα τις ιδιότητες της ορθότητας και της μυστικότητας της ψήφου. Ελπίζουμε να καταδείξουμε τη χρησιμότητα τέτοιων εργαλείων στην ανάλυση πρωτοκόλλων καθώς και να σκιαγραφήσουμε τους τομείς που θα μπορούσαν κατά τη γνώμη μας να αποτελέσουν αντικείμενο περαιτέρω μελέτης.

Λέξεις κλειδιά

κρυπτογραφία, ηλεκτρονική ψηφοφορία, εφαρμοσμένος π-λογισμός, ProVerif, αυτόματη ανάλυση, τυπικές μέθοδοι, επαλήθευση πρωτοκόλλων

Abstract

In this thesis we used formal methods and automated reasoning tools in order to examine security properties of cryptographic protocols, specifically electronic voting (e-voting) protocols. We present the formal language of Applied pi-Calculus and the automated protocol verifier ProVerif, that uses the former as a high-level specification language. We chose to examine a new e-voting protocol suggested in recent literature. We constructed scripts that model the protocol and had ProVerif automatically prove the properties of soundness and vote-privacy. We hope to showcase the importance of these tools and discern areas of future work and improvement.

Keywords

cryptography, electronic voting, e-voting, Applied pi-Calculus, ProVerif, automated reasoning, formal methods, protocol verification

Ευχαριστίες

Ευχαριστώ τον Ομότιμο Καθηγητή κ. Γεώργιο Κολέτσο για τη συμμετοχή του στην επίβλεψη της εργασίας και τον Καθηγητή Πέτρο Στεφανέα για την υποστήριξη και την ευκαιρία να παρουσιάσω μέρος της εργασίας αυτής στην ημερίδα Λογικής που διοργάνωσε. Ευχαριστώ επίσης τους Καθηγητές κ. Παπασπύρου και κ. Παγουρτζή για τη συνεργασία τους και τη συμμετοχή τους στην εξεταστική επιτροπή.

Περιεχόμενα

Κατάλογος Πινάκων

Κεφάλαιο 1

1.1 Εισαγωγή

Στη σημερινή εποχή, η χρήση της κρυπτογραφίας είναι καθολική και καθημερινή από όλους μας, ακόμη και αν πολλές φορές κάτι τέτοιο δεν είναι άμεσα αντιληπτό. Για παράδειγμα, οι εφαρμογές άμεσης επικοινωνίας (instang messaging ή IM) όπως το WhatsApp ή το Viber, προσφέρουν δυνατότητες κρυπτογράφησης από άκρο σε άκρο (end-to-end encryption) χρησιμοποιώντας εξελιγμένα πρωτόκολλα κρυπτογραφίας [1]. Φυσικά, τραπεζικές εφαρμογές (web banking) και ιστοσελίδες ηλεκτρονικού εμπορίου (e-commerce) χρησιμοποιούν κατά κόρον κρυπτογραφία εδώ και πολλά χρόνια. Ακόμα και ιστοσελίδες που δεν περιέχουν εμπιστευτικά δεδομένα, προσφέρουν κρυπτογραφημένες συνδέσεις για τις υπηρεσίες τους (πχ πρακτορείο ειδήσεων CNN).
Μια ακόμα σημαντική εφαρμογή της κρυπτογραφίας, που εξετάζεται στην παρούσα εργασία, βρίσκεται στα πρωτόκολλα ηλεκτρονικής ψηφοφορίας (e-voting). Τα πρωτόκολλα αυτά επιτρέπουν τη διενέργεια ψηφοφοριών απομακρυσμένα, συνήθως μέσω internet και με τη χρήση οποιουδήποτε κοινού ηλεκτρονικού υπολογιστή ή έξυπνης συσκευής με σύνδεση στο διαδίκτυο. Ο ρόλος που μπορούν να παίξουν τέτοια πρωτόκολλα στη λήψη αποφάσεων, στη διαχείριση οργανισμών και γενικότερα στην ενίσχυση της δημοκρατίας είναι σαφής. Τέτοια πρωτόκολλα έχουν χρησιμοποιηθεί ήδη σε διάφορες περιπτώσεις [2], όμως η χρήση τους σε εθνικές εκλογές θα καθυστερήσει ακόμα, αν και σίγουρα θα αυξηθεί. Ο λόγος που υπάρχει ακόμα δισταγμός και αμφιβολία για τη χρήση τους σε τόσο ευρείες εκλογικές διαδικασίες, οφείλεται στα ζητήματα ασφαλείας που μπορεί να παρουσιαστούν και στην εμπιστοσύνη που ενδεχομένως θα δείξουν οι ψηφοφόροι στην εγκυρότητα του αποτελέσματος [3]. Πράγματι, τα πρωτόκολλα e-voting καλούνται να παρέχουν πολλές και ιδιαίτερα εξελιγμένες ιδιότητες ασφαλείας όπως δυνατότητα ελέγχου της εγκυρότητας του αποτελέσματος (verifiability), αντίσταση στον εξαναγκασμό (coercion resistance) και πολλές άλλες [4] (βλ. και κεφάλαιο 4). Για αυτό το λόγο η χρήση κρυπτογραφικών τεχνικών στα πρωτόκολλα αυτά είναι ευρεία αλλά το ίδιο αυξημένη είναι και η ανάγκη να παρέχονται κρυπτογραφικές αποδείξεις για τις ιδιότητες ασφαλείας που προαναφέραμε.
Προκειμένου η χρήση κρυπτογραφίας να οδηγεί πραγματικά στις ιδιότητες ασφαλείας που οφείλει να ικανοποιεί το εκάστοτε πρωτόκολλο, πρέπει τα πρωτόκολλα αυτά να έχουν ελεγχθεί τόσο σε θεωρητικό επίπεδο όσο και σε επίπεδο υλοποίησης. Όσον αφορά τα κρυπτογραφικά θεμελιακά στοιχεία (cryptographic primitives) όπως το κρυπτοσύστημα RSA, αυτό έχει γίνει σε θεωρητικό επίπεδο μέσω μαθηματικών αποδείξεων αλλά και σε επίπεδο υλοποίησης, μέσω βιβλιοθηκών ανοιχτού κώδικα οι οποίες έχουν ελεγχθεί ενδελεχώς από την ερευνητική και όχι μόνο κοινότητα. Δυστυχώς, η χρήση ορθών και ασφαλών primitives δεν οδηγεί απαραίτητα σε ορθά και ασφαλή ολοκληρωμένα πρωτόκολλα. Τέτοια παραδείγματα υπάρχουν πάρα πολλά (βλ. [5], [6]). Κατά συνέπεια,

οφείλει κανείς να ελέγξει εκ νέου τα πρωτόκολλα που σχεδιάζει και υλοποιεί.
Κάτι τέτοιο θα μπορούσε να γίνει με τον ίδιο τρόπο που γίνεται και για τα primitives. Το ζήτημα όμως είναι ότι τέτοιες μαθηματικές αποδείξεις θα ήταν πολύ πιο δύσκολες, μακροσκελείς και επιρρεπείς σε λάθη [7] καθώς ο κώδικας (ή ψευδοκώδικας) που θα έπρεπε να ελεγχθεί θα ήταν πιθανώς αρκετές τάξεις μεγέθους μεγαλύτερος από τον κώδικα ενός primitive. Κατά συνέπεια εμφανίζονται κάποιες δυσκολίες στον έλεγχο των πρωτοκόλλων. Κάποιοι ερευνητές, εδώ και αρκετές δεκαετίες εξετάζουν τη χρήση τυπικών μεθόδων προκειμένου να μπορούν να γίνουν τέτοιοι έλεγχοι αυτόματα ή με τρόπο καθοδηγούμενο από το χρήστη. Σε αυτό το πλαίσιο εντάσσεται η τυπική γλώσσα του εφαρμοσμένου π-λογισμού (applied pi calculus) [8] και το εργαλείο αυτόματης ανάλυσης ProVerif [9], τα οποία εξετάζονται στην παρούσα εργασία.
Η ιστορία των τυπικών μεθόδων στην κρυπτογραφία είναι μεγάλη. Η λογική BAN [10] είναι ένα από τα πρώτα τυπικά συστήματα που εμφανίστηκαν στη βιβλιογραφία για την διατύπωση και ανάλυση κρυπτογραφικών πρωτοκόλλων. Το σύστημα FDR χρησιμοποιήθηκε στο [11] για την εύρεση της κλασικής επίθεσης man-in-the-middle στο πρωτόκολλο Needham-Schroeder [12] και για την επιδιόρθωση του. Αργότερα γράφτηκε ένας μεταγλωττιστής (compiler) για το FDR από μια γλώσσα μοντελοποίησης υψηλού επιπέδου. Συνολικά το σύστημα είναι γνωστό ως Casper/FDR και παρουσιάζεται στο [13]. Το εργαλείο AVISPA [14], περιλαμβάνει μια συλλογή διαφορετικών back-ends για την εφαρμογή διαφορετικών τεχνικών κατά την ανάλυση πρωτοκόλλων. Το εργαλείο Scyther [15] είναι ένα σύγχρονο εργαλείο που έχει χρησιμοποιηθεί επιτυχώς για την ανάλυση πρωτοκόλλων. Για μια σύγκριση των χαρακτηριστικών και της απόδοσης των παραπάνω εργαλείων παραπέμπουμε στο [16].
Οι τυπικές μέθοδοι στην κρυπτογραφία έχουν καταφέρει αρκετά κυρίως μέσω της αναγνώρισης σφαλμάτων σε πρωτόκολλα που θεωρούνταν ασφαλή (με χαρακτηριστικότερο παράδειγμα το [11]). Ένα ακόμη παράδειγμα είναι το [17], όπου με χρήση της τυπικής γλώσσας του εφαρμοσμένου π-λογισμού, που παρουσιάζουμε στο κεφάλαιο 2, βρίσκονται και έπειτα διορθώνονται σφάλματα στο δημοφιλές πρωτόκολλο/λογισμικό e-voting Helios [18]. Χρήση της ProVerif, που παρουσιάζεται στο κεφάλαιο 3, έχει γίνει στο [19] για να αποδειχθεί η ασφάλεια κάποιων τμημάτων του TLS 1.3.
Στην παρούσα εργασία, θα παρουσιάσουμε την τυπική γλώσσα του εφαρμοσμένου π-λογισμού ως το φορμαλιστικό πλαίσιο στο οποίο θα δίνονται οι ορισμοί των ιδιοτήτων ασφαλείας και οι περιγραφές των πρωτοκόλλων. Θα πρέπει έπειτα να αποδειχθεί ότι κάποιο πρωτόκολλο ικανοποιεί μια συγκεκριμένη ιδιότητα ασφαλείας. Θα χρησιμοποιήσουμε την ProVerif προκειμένου να αποδείξουμε αυτόματα τέτοιες προτάσεις. Πεδίο εφαρμογής μας θα είναι το πρωτόκολλο e-voting που παρουσιάστηκε στο [20], το οποίο είναι ένα σύγχρονο πρωτόκολλο που ικανοποιεί πολλές ιδιότητες ασφαλείας. Αφού παρουσιάσουμε το πρωτόκολλο και το μοντελοποιήσουμε σε εφαρμοσμένο π-λογισμό, θα εξετάσουμε δύο ιδιότητες, τη μυστικότητα της ψήφου και την ορθότητα (βλ. κεφάλαιο 4).

1.2 Το συμβολικό μοντέλο στην κρυπτογραφία

Η σύγχρονη κρυπτογραφία δεν στηρίζεται πλέον στην αδυναμία του σχεδιαστή/ερευνητή να βρει κάποιο σφάλμα στο πρωτόκολλο που ο ίδιος σχεδίασε. Αν και τελικά ένα πρωτόκολλο μπορεί να θεωρηθεί "ασφαλές" μόνο μετά από χρόνια μελέτης και χρήσης κατά τα οποία δεν βρέθηκαν ή διορθώθηκαν τυχόν προβλήματα, σε πρώτο στάδιο κάθε προτεινόμενο πρωτόκολλο οφείλει να συνοδεύεται σήμερα από μαθηματικές αποδείξεις για την ασφάλειά του. Οι αποδείξεις αυτές αφορούν προτάσεις της εξής μορφής:

Το κρυπτογραφικό σχήμα C είναι "ασφαλές" αν για κάθε πιθανοτικό πολυωνυμικού-χρόνου αντίπαλο A ο οποίος ¨επιτίθεται" στο C, η πιθανότητα να "επιτύχει" ο A είναι αμελητέα. [21]

Φυσικά η παραπάνω πρόταση απέχει αρκετά από το να μπορεί να θεωρηθεί σαφής: θα πρέπει να ορίσουμε τυπικά τι εννοούμε λέγοντας "ασφαλές","επιτίθεται" και "επιτύχει" (όροι όπως "πιθανοτικός πολυωνυμικού-χρόνου" και "αμελητέα" έχουν αυστηρούς ορισμούς στη θεωρητική επιστήμη των υπολογιστών). Ο αυστηρός ορισμός των εννοιών αυτών εξαρτάται από το εκάστοτε πρωτόκολλο και μάλιστα μπορούν να εξεταστούν διάφοροι ορισμοί για κάθε ένα πρωτόκολλο, οι οποίοι θα αντιστοιχούν σε διαφορετικές έννοιες ασφάλειας, παρέχοντας διαφορετικές εγγυήσεις για το πρωτόκολλο.
Τα προηγούμενα αφορούν τον τρόπο με τον οποίο δίνονται σήμερα οι αποδείξεις ασφαλείας των διαφόρων κρυπτογραφικών primitives (πχ κρυπτοσύστημα RSA ή ανταλλαγή κλειδιών Diffie-Hellman) και αποκαλούνται συνολικά το υπολογιστικό μοντέλο στην κρυπτογραφία ή υπολογιστική ασφάλεια (computational security).
Μετατοπίζοντας το ενδιαφέρον μας από τα θεμελιακά στοιχεία στα κρυπτογραφικά πρωτόκολλα (πχ πρωτόκολλο αυθεντικοποίησης Needham-Shroeder [12]) διαπιστώνουμε ότι δεν είναι εύκολο, ακόμη και αν έχουμε στη διάθεσή μας (αποδεδειγμένα) ασφαλή primitives να σχεδιάσουμε ένα ασφαλές πρωτόκολλο. Συγκεκριμένα για το προαναφερθέν πρωτόκολλο βρέθηκε επίθεση τύπου man-in-the-middle σχεδόν 20 χρόνια μετά την αρχική δημοσίευσή του [5]. Δημιουργείται έτσι η ανάγκη να αποδείξουμε με σαφήνεια, σε επίπεδο πρωτοκόλλου πλέον, την ασφάλειά του. Η χρήση του υπολογιστικού μοντέλου στο επίπεδο αυτό είναι δύσκολη, επιρρεπής σε λάθη και οδηγεί σε μακροσκελείς αποδείξεις οι οποίες συνήθως δεν ελέγχονται ξανά [22]*. Οι παρατηρήσεις αυτές οδήγησαν τους Dolev και Yao να ορίσουν στο [7] ένα τυπικό μοντέλο υψηλότερου αφαιρετικού επιπέδου για τις αποδείξεις ασφαλείας των κρυπτογραφικών πρωτοκόλλων. Στο μοντέλο αυτό, ένα σχήμα κρυπτογράφησης δημοσίου κλειδιού (public key encryption scheme) θεωρείται ως ένα ζεύγος συναρτήσεων (E_X , D_X) με τις ιδιότητες:

* Από το [22], καταλήγοντας σχετικά με τα προηγούμενα: "In consequence, there is almost no real cryptographic proof of a larger protocol, and several times supposedly proven, relatively small systems were later broken".

- $E_X\, D_X = \ D_X\, E_X = 1$ (η σταθερή συνάρτηση)
- γνώση του $E_X(M)$ δεν αποκαλύπτει τίποτα για το M
- μόνο η οντότητα X έχει στη διάθεσή της τη συνάρτηση D_X

Ανάλογα μπορούν να θεωρηθούν και σχήματα ψηφιακών υπογραφών καθώς και άλλα primitives που χρησιμοποιούνται στα κρυπτογραφικά πρωτόκολλα, πολλά από τα οποία θα παρουσιαστούν παρακάτω στη γλώσσα του εφαρμοσμένου π-λογισμού και της ProVerif (βλ. 2.1, 3.2 και 4.3).
Όσον αφορά τις δυνατότητες του εχθρού, θεωρείται ότι έχει στη διάθεσή του το υποκείμενο δίκτυο και συνεπώς μπορεί να βλέπει όλα τα μηνύματα που στέλνονται σε αυτό και να στείλει μηνύματα σε όποιον θέλει, κάτι που περιλαμβάνει προφανώς να ξαναστείλει μηνύματα που έχει δει ήδη στο δίκτυο (replay attacks) αλλά και να στείλει εξ ολοκλήρου νέα μηνύματα. Ένας τέτοιος εχθρός συχνά αποκαλείται στη βιβλιογραφία "εχθρός Dolev-Yao".
Αυτή η αλγεβρική, ουσιαστικά, θεώρηση των κρυπτογραφικών primitives ακολουθείται στη βιβλιογραφία των τυπικών μεθόδων στην κρυπτογραφία. Αν και κάθε τυπικό σύστημα μπορεί να παρουσιάζει μικροδιαφορές και αποκλίσεις από άλλα τυπικά συστήματα ή από την αρχική διατύπωση στο [7], η θεώρηση αυτή απαρτίζει το λεγόμενο "συμβολικό μοντέλο" στην κρυπτογραφία.
Ένα εύλογο ερώτημα που προκύπτει στο σημείο αυτό αφορά τη σχέση μεταξύ των δύο μοντέλων. Αν βρεθεί κάποια επίθεση στο συμβολικό μοντέλο τότε είναι δυνατόν να ελεγχθεί εύκολα αν η ίδια επίθεση ισχύει και στο υπολογιστικό μοντέλο. Η αντίθετη κατεύθυνση παρουσιάζει μεγαλύτερο ενδιαφέρον. Αν αποδείξουμε την ασφάλεια ενός συστήματος στο συμβολικό μοντέλο, έχουμε κάποια εγγύηση ασφάλειας στο υπολογιστικό μοντέλο; Στη βιβλιογραφία υπάρχουν διαφορά θετικά αποτελέσματα (βλ. [23],[22]) αλλά και κάποια αρνητικά [24].
Στο [22] ορίζεται τυπικά ένα κατανεμημένο υπολογιστικό σύστημα με δυνατότητα ανταλλαγής μηνυμάτων, στο οποίο υπάρχουν τα (συμβολικά) primitives της κρυπτογράφησης και των ψηφιακών υπογραφών. Αυτό το ιδανικό (ideal) σύστημα τυποποιεί το συμβολικό μοντέλο και τον εχθρό Dolev-Yao. Ορίζεται έπειτα ένα κατανεμημένο υπολογιστικό σύστημα, όπως το προηγούμενο, αλλά με κρυπτογραφικά primitives τα οποία παρέχουν εγγυήσεις υπολογιστικής ασφάλειας, πχ IND-CCA2 [25] για το σχήμα κρυπτογράφησης. Αυτό το "πραγματικό" (real) σύστημα τυποποιεί τη συνηθισμένη προγραμματιστική χρήση της κρυπτογραφίας ως κλήση συναρτήσεων (function calls) σε κάποιο κρυπτογραφικό API/βιβλιοθήκη (library). Επίσης εδώ ο εχθρός είναι πιθανοτικός πολυωνυμικού χρόνου, όπως συνηθίζεται στο υπολογιστικό μοντέλο. Μπορεί τότε να αποδειχθεί ότι το πραγματικό σύστημα είναι τόσο ασφαλές όσο και το ιδανικό. Αυτό σημαίνει ότι αν για ένα πρωτόκολλο που έχει μοντελοποιηθεί στο ιδανικό σύστημα δεν έχουν βρεθεί επιθέσεις, τότε δεν θα υπάρχουν επιθέσεις και στο πραγματικό σύστημα. Αυτή η έννοια ορθότητας, που αποκαλείται Black Box Reactive Simulatability (BRSIM), είναι η πιο ισχυρή ιδιότητα ορθότητας που μπορούμε να ορίσουμε για το συμβολικό μοντέλο.

Τα θετικά αποτελέσματα που εμφανίζονται στο [22], αφορούν την κρυπτογραφία δημοσίου κλειδιού και τις ψηφιακές υπογραφές, δύο από τα σημαντικότερα primitives που χρησιμοποιούνται στα σύγχρονα πρωτόκολλα που προσφέρουν κάποιου είδους ασφάλεια. Αν και τα αποτελέσματα αυτά είναι δυνατόν να επεκταθούν και σε περισσότερα primitives, όπως για παράδειγμα στην συμμετρική κρυπτογραφία [26], προβλήματα έχουν παρουσιαστεί σε διάφορα άλλα. Ίσως το σημαντικότερο εξ αυτών είναι οι συναρτήσεις κατακερματισμού (hash functions), όπου φαίνεται ότι η μοντελοποίηση τους προσφέρει μεγαλύτερη ασφάλεια από όση μπορεί να επιτευχθεί στην πράξη. Πράγματι, στο συμβολικό μοντέλο είναι συνηθισμένη η θεώρηση ότι η κατασκευή hash(M) δεν αποκαλύπτει απολύτως τίποτα για το M, κάτι που δεν ισχύει στην πράξη και άρα δεν μπορούμε να δώσουμε μια πραγματοποίηση (realization) μιας συνάρτησης κατακερματισμού έτσι ώστε να ισχύει η BRSIM ιδιότητα μεταξύ του ιδανικού και του πραγματικού συστήματος [24].
Στην παρούσα εργασία θα χρησιμοποιηθεί μια πληθώρα από κρυπτογραφικά primitives: κρυπτογραφία δημοσίου κλειδιού, ψηφιακές υπογραφές, τυφλές υπογραφές (blind signatures-μάλιστα μια νέα παραλλαγή τους που αναπτύχθηκε για το συγκεκριμένο πρωτόκολλο, βλ. κεφάλαιο 4) και αποδείξεις μηδενικής γνώσης (zero-knowledge proofs). Κάτι τέτοιο είναι σύνηθες στη βιβλιογραφία του εφαρμοσμένου π-λογισμού και της ProVerif, αν και πρέπει να σημειώσουμε ότι, όπως φαίνεται και από τις προηγούμενες παραγράφους, δεν είναι πάντοτε σαφές το επίπεδο ασφάλειας που ισχύει κατά την πραγματοποίηση των ιδεατών πρωτοκόλλων που αναλύουμε με τα πραγματικά κρυπτογραφικά primitives, ειδικά στην περίπτωση που κάποια από αυτά τα primitives είναι σχετικά νέα και με περιορισμένη χρήση (όπως η παραλλαγή των blind signatures που προαναφέραμε). Σημειώνουμε πάντως ότι δεν χρησιμοποιούμε συναρτήσεις κατακερματισμού.

Κεφάλαιο 2

2.1 Εφαρμοσμένος π-Λογισμός

Ο Εφαρμοσμένος π-Λογισμός είναι μια επέκταση του π-Λογισμού [27] με συναρτήσεις και ισότητα. Κύριο πεδίο εφαρμογής του είναι η τυπική διατύπωση και ανάλυση κρυπτογραφικών πρωτοκόλλων στο συμβολικό μοντέλο. Μία διάλεκτος του εφαρμοσμένου π-Λογισμού είναι η γλώσσα μοντελοποίησης (specification language) που χρησιμοποιεί το λογισμικό ProVerif ως είσοδο υψηλού επιπέδου (high level input) για την αυτόματη ανάλυση κρυπτογραφικών πρωτοκόλλων. Στην ενότητα αυτή θα μελετήσουμε τον Εφαρμοσμένο π-λογισμό προκειμένου να διατυπώσουμε αργότερα τις ιδιότητες που θα αποδείξουμε για το πρωτόκολλο e-voting που μελετάμε. Επίσης θα προσπαθήσουμε να διασαφηνίσουμε τη σχέση μεταξύ ProVerif και εφαρμοσμένου π-λογισμού. Βασική αναφορά αποτελεί το [8].
Θεωρούμε μια υπογραφή (signature) Σ που περιέχει πεπερασμένο πλήθος συναρτησιακών συμβόλων (function symbols), το καθένα συσχετισμένο με κάποιο πλήθος ορισμάτων (arity-πχ *f/1,g/2,enc/3,hash/1,pk/1,sign/2*). Μπορούμε να θεωρήσουμε ως σύμβολα σταθερών τα σύμβολα που έχουν arity 0. Επίσης θεωρούμε ότι διαθέτουμε ένα (αριθμήσιμα) άπειρο σύνολο με ονόματα (names) (πχ *a,b,n,skey,nonce,rand*). Τέλος θεωρούμε ότι έχουμε στη διάθεσή μας ένα (αριθμήσιμα) άπειρο σύνολο μεταβλητών (πχ *x,y,z*). Μπορούμε να συμβολίσουμε το σύνολο των μεταβλητών με *Var* και το σύνολο των ονομάτων με *Names*. Κατασκευάζουμε το σύνολο των όρων της γλώσσας ως εξής:

Ορισμός 2.1. Έστω υπογραφή Σ όπως παραπάνω. Το σύνολο των όρων T_Σ που παράγονται από το Σ είναι το μικρότερο σύνολο για το οποίο ισχύει:

- Αν $x \in Var$ τότε $x \in T_\Sigma$.
- Αν $n \in Names$ τότε $n \in T_\Sigma$.
- Αν $M_1, M_2, ..., M_k \in T_\Sigma$ και f σύμβολο συνάρτησης k-θέσεων του Σ τότε $f(M_1, M_2, ..., M_k) \in T_\Sigma$.

Για παράδειγμα, μπορούμε να έχουμε τους εξής όρους: *enc(pk(skey),m,rand), sign(m,skey),hash(x)*. Εδώ οι πρώτοι δύο όροι είναι *κλειστοί* αφού δεν περιέχουν μεταβλητές. Θα χρησιμοποιούμε τα *a,b,c,n* για να αναφερθούμε σε ονόματα και τα *x,y,z* για να αναφερθούμε σε μεταβλητές. Μπορούμε να χρησιμοποιήσουμε τα *u,v,w* για να αναφερθούμε είτε σε ονόματα είτε σε μεταβλητές.
Στον εφαρμοσμένο π-λογισμό, μοντελοποιούμε τα πρωτόκολλα που θέλουμε να μελετήσουμε ως διεργασίες (processes). Οι διεργασίες αυτές μπορούν να κάνουν είσοδο/έξοδο (input/output) από/σε όρους-κανάλια (channels). Επίσης μπορούν να τρέχουν παράλληλα με άλλες διεργασίες, να χρησιμοποιούν νέα ονόματα (με τοπική εμβέλεια) και να εκτελούνται υπό συνθήκη (conditionals). Το τελευταίο αυτό σημείο μας

αναγκάζει να μιλήσουμε για θεωρίες ισότητας (equational theories), προκειμένου να ελέγξουμε αν η συνθήκη που εξετάζεται είναι αληθής ή ψευδής. Συνεπώς πριν δώσουμε την γραμματική των διεργασιών, δίνουμε στα επόμενα τον ορισμό της σχέσης $=_E$, ακολουθώντας το [28]. Ξεκινάμε από την έννοια της αντικατάστασης:

Ορισμός 2.2. Μια αντικατάσταση (substitution) είναι μια συνάρτηση σ:Var→T_Σ. Συμβολίζουμε την αντικατάσταση σ για την οποία σ(x)=M και σ(y)=y για όλες τις άλλες μεταβλητές y, με {M/x}.

Για τις αντικαταστάσεις συνηθίζεται να γράφουμε *xσ* αντί για *σ(x)*, σύμβαση την οποία ακολουθούμε. Μπορούμε εύκολα να επεκτείνουμε μια αντικατάσταση σ στους όρους του T_Σ επαγωγικά, ως εξής:

$$x\sigma = \sigma(x)$$
$$n\sigma = n$$
$$f(M_1, M_2, ..., M_k)\sigma = f(M_1\sigma, M_2\sigma, ..., M_k\sigma)$$

Αργότερα θα επεκτείνουμε την αντικατάσταση και στις διεργασίες.

Ορισμός 2.3. Έστω E ένα πεπερασμένο υποσύνολο του $T_\Sigma{}^2$. Η $=_E$ είναι η μικρότερη σχέση ισοδυναμίας στο $T_\Sigma{}^2$ που περιλαμβάνει την E και για την οποία ισχύουν τα εξής:

i. Αν $M_1 =_E N_1$, $M_2 =_E N_2$, ..., $M_k =_E N_k$ και f σύμβολο συνάρτησης k-θέσεων του Σ τότε $f(M_1, M_2, ..., M_k) =_E f(N_1, N_2, ..., N_k)$.
ii. Αν $M =_E N$ και σ μια αντικατάσταση, τότε $M\sigma =_E N\sigma$.
iii. Αν $M =_E N$ και τ μια "μετονομασία" των ονομάτων, δηλαδή τ:Names → Names με τ 1-1 και επί, τότε $M\tau =_E N\tau$.

Η $=_E$ λέγεται θεωρία ισότητας που παράγεται από το E.

Αν $M =_E N$ συνήθως θα γράφουμε απλά $M = N$, παρ'όλα αυτά θα πρέπει να έχουμε υπόψιν ότι δεν πρόκειται απλά για συντακτική ισότητα και ότι η ισότητα δύο όρων εξαρτάται πάντα από το σύνολο E που χρησιμοποιούμε. Επίσης, αν *(M,N)* ∈ *E*, μπορούμε να γράφουμε $M = N$.
Ένα απλό παράδειγμα για το σύνολο E είναι το εξής:

$$E = \{dec(enc(pk(skey),m,rand),skey)=m,$$
$$ver(sign(m,skey),pk(skey))=true,$$
$$getmsg(sign(m,skey))=m\}$$

Το σύνολο αυτό περιλαμβάνει τα (μοντέλα για τα) primitives της κρυπτογράφησης δημοσίου κλειδιού και της ψηφιακής υπογραφής.
Μπορούμε τώρα να ορίσουμε το σύνολο των απλών διεργασιών.

Ορισμός 2.4. Έστω υπογραφή Σ και σύνολο όρων T_Σ. Το σύνολο των απλών διεργασιών $P(T_\Sigma)$ είναι το μικρότερο σύνολο για το οποίο ισχύουν τα εξής:

i. Η διεργασία 0 ανήκει στο $P(T_\Sigma)$
ii. Αν $P,Q \in P(T_\Sigma)$, n όνομα, x μεταβλητή και $M,N \in T_\Sigma$ τότε:
   - $P \mid Q \in P(T_\Sigma)$
   - $!P \in P(T_\Sigma)$
   - $vn.P \in P(T_\Sigma)$
   - if $M = N$ then P else $Q \in P(T_\Sigma)$
   - $N(x).P \in P(T_\Sigma)$
   - $N\langle M\rangle.P \in P(T_\Sigma)$

Μπορούμε κάπως άτυπα να δώσουμε την εξής σημασία στα παραπάνω (βλ. 2.2 για τυπική σημασιολογία): με 0 συμβολίζουμε την διεργασία που δεν κάνει τίποτα, *!P* είναι η αντιγραφή (replication) της διεργασίας P και δηλώνει ότι τρέχει ένα ακαθόριστο πλήθος αντιγράφων της P παράλληλα, το σύμβολο | δηλώνει ότι δύο διεργασίες τρέχουν παράλληλα και *vn.P* σημαίνει ότι το n είναι ένα νέο δεσμευμένο όνομα στην P (name restriction). Αν $M =_E N$ τότε εκτελείται η P, διαφορετικά η Q στη διεργασία *if M = N then P else Q,* ενώ η *N(x).P* δηλώνει ότι θα γίνει input στο κανάλι N και έπειτα τρέχει η διεργασία P με τον όρο που έγινε input να αντικαθιστά τις ελεύθερες εμφανίσεις της μεταβλητής x στην διεργασία P. Τέλος η διεργασία $N\langle M\rangle.P$ κάνει output τον όρο M στο κανάλι N και έπειτα εκτελεί τη διεργασία P. Αν και τυπικά τα κανάλια είναι γενικοί όροι, στην πράξη είναι απλώς ονόματα. Ένα παράδειγμα διεργασίας είναι:

P:= vrand. c(x). c(s). if ver(s,x)=true then c<enc(x,mess,rand)> else 0

Η P αφού δημιουργήσει μια τυχαιότητα *rand*, κάνει input στο κανάλι *c* στις μεταβλητές *x* και *s*. Αναμένει ένα δημόσιο κλειδί στη θέση του *x* με το οποίο θα επαληθεύσει μια υπογραφή που έχει λάβει στη μεταβλητή *s*. Αν η υπογραφή επαληθεύεται, θα στείλει στο κανάλι *c* το μήνυμα *mess* κρυπτογραφημένο με το κλειδί *x*, με χρήση της τυχαιότητας *rand*. Διαφορετικά δεν θα κάνει τίποτα (ο κλάδος else 0 συνήθως παραλείπεται). Σημειώνουμε ότι το κανάλι *c* και το μήνυμα *mess* είναι ελεύθερα ονόματα, εν αντιθέσει με το *rand* που είναι δεσμευμένο (βλ. και παρακάτω).
Τα παραπάνω αποτελούν τις λεγόμενες *απλές διεργασίες (plain processes).* Η διάκριση αυτή έχει κάποια σημασία για λόγους που θα φανούν παρακάτω. Παρ'όλα αυτά επεκτείνουμε τον προηγούμενο ορισμό για να συμπεριλάβουμε και μερικές ακόμη κατασκευές διεργασιών:

Ορισμός 2.5. Έστω υπογραφή Σ και σύνολο όρων T_Σ. Το σύνολο των επεκτεταμένων διεργασιών (extended processes) $EP(T_\Sigma)$ είναι το μικρότερο σύνολο το οποίο περιέχει το $P(T_\Sigma)$ (ορισμός 2.4) και για το οποίο ισχύουν τα εξής:
Αν $A,B \in EP(T_\Sigma)$, n όνομα, x μεταβλητή και $M \in T_\Sigma$ τότε:

- $A \mid B \in EP(T_\Sigma)$
- $vn.A \in EP(T_\Sigma)$
- $vx.A \in EP(T_\Sigma)$
- $\{M/x\} \in EP(T_\Sigma)$

Αυτό που επιτρέπει ο παραπάνω ορισμός είναι να θεωρηθούν οι αντικαταστάσεις ως διεργασίες, τις οποίες ονομάζουμε ενεργές αντικαταστάσεις (active substitutions). Σημειώνουμε ότι επιτρέπουμε το πολύ μία ενεργή αντικατάσταση για κάθε μεταβλητή για προφανείς λόγους σαφήνειας. Μπορούμε έτσι να επιτρέψουμε την προγραμματιστική κατασκευή *let x=M in P* (δέσμευση της μεταβλητής x και αντικατάσταση των ελεύθερων εμφανίσεων της στη διεργασία P με τον όρο M). Η κατασκευή αυτή εμφανίζεται αυτολεξεί στην ProVerif. Στον εφαρμοσμένο π-λογισμό με extended processes αυτό μπορεί πλέον να γραφτεί τυπικά ως: *vx.({M/x} | P)*. Θεωρούμε τη διεργασία *let x=M in P* ως "syntactic sugar" της προηγούμενης διεργασίας. Μπορούμε έτσι να απλοποιήσουμε τη διατύπωση ορισμένων διεργασιών. Για παράδειγμα η διεργασία P που δώσαμε παραπάνω μπορεί να γραφτεί πλέον ως:

P:=vrand. c(x). let z=enc(x,mess,rand) in c(s). if ver(s,x) then c<z>

Όπως συνηθίζεται στη λογική, μπορούμε να ορίσουμε για μια επεκτεταμένη διεργασία *A* το σύνολο των ελεύθερων ονομάτων της A, *fn(A)*, και το σύνολο των ελεύθερων μεταβλητών της, *fv(A)* (πιο αυστηρά: ονόματα και μεταβλητές που έχουν ελεύθερες εμφανίσεις στην A). Σημειώνουμε εδώ τις μη τετριμμένες περιπτώσεις (για τον πλήρη ορισμό βλ. [8]):

- $fn(vn.A) = fn(A) \setminus \{n\}$
- $fv(vx.A) = fv(A) \setminus \{x\}$
- $fv(N(x).A) = fv(N) \cup (fv(A) \setminus \{x\})$
- $fv(\{M/x\}) = fv(M) \cup \{x\}$

2.2 Σημασιολογία

Στην ενότητα αυτή θα δώσουμε με τυπικό τρόπο την ερμηνεία των διεργασιών. Ακολουθούμε μια προσέγγιση λειτουργικής σημασιολογίας (operational semantics) που βασίζεται σε δύο σχέσεις: την σχέση της εσωτερικής αναγωγής $\rightarrow$ (internal reduction) και τη σχέση της δομικής ισοδυναμίας $\equiv$ (structural equivalence). Ξεκινάμε με τη σημαντική έννοια του περιβάλλοντος αποτίμησης (evaluation context), το οποίο μπορεί να περιγραφεί ως μία επεκτεταμένη διεργασία με κάποιο κενό (hole). Ένα περιβάλλον αποτίμησης C (συμβολ. C[_]) παίρνει ως είσοδο μια επεκτεταμένη διεργασία A η οποία αντικαθίσταται στα κενά του C (συμβολ. C[A]), δίνοντας έτσι μια νέα επεκτεταμένη διεργασία. Η διεργασία A στην C[A] θα πρέπει να μην βρίσκεται υπό αντιγραφή, να μην βρίσκεται σε κάποιο κλάδο μιας διεργασίας *if ... then ... else* και να μην προηγείται αυτής είσοδος ή έξοδος.
Ως ένα απλό παράδειγμα, που θα χρησιμοποιήσουμε και παρακάτω, μπορούμε να έχουμε *C[_]=vx.({M/x} | _)*. Τότε για *P' = N<x>.P | N(x).Q*, έχουμε *C[P']= vx.({M/x} | (N<x>.P | N(x).Q))*.
Στους επόμενους ορισμούς θα επεκτείνουμε το πεδίο ορισμού μιας αντικατάστασης σ από το T_Σ στο $EP(T_\Sigma)$.

Ορισμός 2.6. Έστω σ: Var $\rightarrow$ T_Σ αντικατάσταση. Συμβολίζουμε με σ_x την αντικατάσταση για την οποία $\sigma_x(x) = x$ και $\sigma_x(y) = \sigma(y)$ για οποιαδήποτε άλλη μεταβλητή y .

Ορισμός 2.7. Έστω σ: Var$\rightarrow$ T_Σ αντικατάσταση. Έχουμε ήδη δει πως η σ επεκτείνεται στους όρους του T_Σ. Ορίζουμε την επέκταση της σ στις διεργασίες του $EP(T_\Sigma)$,ως εξής:

- $0\sigma = 0$
- $(A \mid B)\sigma = A\sigma \mid B\sigma$
- $(!P)\sigma = !(P\sigma)$
- $(vn.A)\sigma = vn.\ A\sigma$
- $(N(x).\ P)\sigma = (N\sigma)(x).P\sigma_x$
- $(N\langle M\rangle.\ P)\sigma = (N\sigma)\langle M\sigma\rangle.\ P\sigma$
- $(\text{if } M = N \text{ then } P \text{ else } Q)\sigma = (\text{if } M\sigma = N\sigma \text{ then } P\sigma \text{ else } Q\sigma)$
- $(vx.\ A)\sigma = vx.\ A\sigma_x$.
- $(\{M/x\})\sigma = \{M\sigma/x\}$

Για μια επεκτεταμένη διεργασία A, θα λέμε ότι μια μεταβλητή x είναι δεσμευμένη ή πιο αυστηρά, ότι έχει δεσμευμένες εμφανίσεις στη διεργασία A, όταν υπάρχουν εμφανίσεις της μεταβλητής που βρίσκονται στην εμβέλεια της δέσμευσης (binder) *vx* ή της εισόδου *N(x)*. Ομοίως, θα λέμε ότι το όνομα n είναι δεσμευμένο στην A όταν βρίσκεται στην

εμβέλεια της δέσμευσης *vn*. Για παράδειγμα, στη διεργασία *vrand. N(x). M<enc(x,key,rand)>* όλες οι εμφανίσεις της μεταβλητής x και όλες οι εμφανίσεις του ονόματος *rand* είναι δεσμευμένες. Αντιθέτως η μεταβλητή *key* είναι ελεύθερη.
Διαισθητικά, δύο διεργασίες που διαφέρουν μόνο στις δεσμευμένες μεταβλητές και στα δεσμευμένα ονόματα δεν διαφέρουν και πολύ. Για παράδειγμα, η διεργασία *vrandom. N(y). M<enc(y,key,random)>,* δεν διαφέρει ουσιαστικά με αυτή που είδαμε στην προηγούμενη παράγραφο. Αν αντικαταστήσουμε όλες τις δεσμευμένες εμφανίσεις του ονόματος *random* με το όνομα *rand* και ομοίως τις δεσμευμένες εμφανίσεις της μεταβλητής y με τη μεταβλητή x, προκύπτει ακριβώς η ίδια διεργασία. Γενικότερα, αν τ:Names → Names και ρ:Var → Var με τ, ρ 1-1 και επί, δηλαδή τ, ρ μετονομασίες των ονομάτων και των μεταβλητών αντίστοιχα, η διεργασία A' που προκύπτει από την A με εφαρμογή των τ και ρ στις δεσμευμένες εμφανίσεις ονομάτων και μεταβλητών της A, θα λέγεται α-μετατροπή (α-conversion) της A*.
Δίνουμε με βάση τα προηγούμενα τον ορισμό της α-ισοδυναμίας (α-equivalence) στον εφαρμοσμένο π-λογισμό:

Ορισμός 2.8. Έστω (επεκτεταμένες) διεργασίες A, B ∈ $EP(T_\Sigma)$. Αν υπάρχει α-μετατροπή A' της διεργασίας A ώστε A' = B (συντακτική ισότητα-ταυτότητα) τότε οι A και B θα λέγονται α-ισοδύναμες διεργασίες.

Μπορούμε τώρα να δώσουμε τον ορισμό της δομικής ισοδυναμίας:

Ορισμός 2.9. Έστω $=_E$ θεωρία ισότητας (ορισμός 2.3). Η δομική ισοδυναμία ≡ είναι η μικρότερη σχέση ισοδυναμίας στο σύνολο $EP(T_\Sigma)$ για την οποία ισχύον τα εξής:

i. Αν P ≡ Q και C κάποιο περιβάλλον αποτίμησης τότε C[P] ≡ C[Q].
ii. Αν P και Q είναι α-ισοδύναμες τότε P ≡ Q.
iii. A ≡ A | 0
iv. A | (B | C) ≡ (A | B) | C και A | B ≡ B | A
v. !P ≡ P | !P
vi. vn.0 ≡ 0 και vu.vw. A ≡ vw.vu. A
vii. A | vu.B ≡ vu.(A|B) όταν u δεν ανήκει στο (fv(A) ⋃ fn(A)).
viii. vx.{M/x} ≡ 0
ix. {M/x}| A ≡ {M/x}| A{M/x}
x. Αν M $=_E$ N τότε {M/x} ≡ {N/x}.

* Μπορούμε να θεωρήσουμε ότι δεν έχουμε στις διεργασίες μας επαναχρησιμοποίηση των δεσμευμένων μεταβλητών, δηλαδή δεν έχουμε διεργασίες της μορφής *N(x). M<hash(x)>.vx. M<enc(x, key)>*. Σε μια γενικότερη περίπτωση α-μετατροπής, η προηγούμενη διεργασία θα μπορούσε να μετατραπεί στην *N(y). M<hash(y)>.vz. M<enc(z, key)>,* ενώ ο ορισμός μας δεν επιτρέπει κάτι τέτοιο.

Στον παραπάνω ορισμό *P,Q,A,B,C* είναι διεργασίες, *n* όνομα,*x* μεταβλητή, *M, N* όροι του T_Σ και υπενθυμίζουμε ότι με *w,u* συμβολίζουμε είτε κάποια μεταβλητή είτε κάποιο όνομα. Επίσης υποθέτουμε ότι υπάρχει το πολύ μια ενεργή αντικατάσταση για κάθε μεταβλητή και ότι για κάθε δεσμευμένη μεταβλητή x υπάρχει ακριβώς μια ενεργή αντικατάσταση της μορφής {M/x}. Αυτοί οι περιορισμοί είναι απαραίτητοι προκειμένου να έχει νόημα η ερμηνεία που δόθηκε παραπάνω και να αποφεύγονται καταστάσεις όπως *{M/x} | {N/x} | A*. Με βάση τον παραπάνω ορισμό έχουμε *A{M/x} ≡ vx.({M/x}| A)* διότι:

$$\mathrm{A\{M/x\}} \equiv \mathrm{A\{M/x\}} \mid 0 \equiv \mathrm{A\{M/x\}} \mid \mathrm{vx.\{M/x\}} \equiv \mathrm{vx.(A\{M/x\}} \mid \mathrm{\{M/x\})} \equiv \mathrm{vx.}$$
$$\mathrm{(\{M/x\}} \mid \ \mid \mathrm{A\{M/x\})} \equiv \mathrm{vx.(\{M/x\}|A)}$$

Η προηγούμενη σχέση ισχύει με την προϋπόθεση ότι $x \notin fv(M)$ (λόγω του (vii) από τον ορισμό 2.9).
Η βασική σχέση που δίνει τη σημασιολογία των διεργασιών είναι η εσωτερική αναγωγή. Με αυτή τη σχέση καθορίζεται ο υπολογισμός (evaluation) των διεργασιών.

Ορισμός 2.10. Έστω $=_E$ θεωρία ισότητας (ορισμός 2.3). Η εσωτερική αναγωγή → είναι η μικρότερη μεταβατική σχέση που περιέχει τη δομική ισοδυναμία (ορισμός 2.9) και επιπλέον ικανοποιεί τα εξής:

i. Αν P → Q και C κάποιο περιβάλλον αποτίμησης τότε C[P] → C[Q].
ii. N<x>.P | N(x).Q → P | Q , όπου x μεταβλητή
iii. if M = N then P else Q → P , αν M $=_E$ N.
iv. if M = N then P else Q → Q , αν M $\neq_E$ N.

Με βάση τα παραπάνω μπορούμε για παράδειγμα να δείξουμε ότι *N<M>. P | N(x).Q → P | Q{M/x},* όπως αναμένουμε διαισθητικά:

$$\mathrm{N<M>.P} \mid \mathrm{N(x).Q} \equiv \mathrm{N<M>.P} \mid \mathrm{N(x).Q} \mid 0 \equiv$$
$$\equiv \mathrm{N<M>.P} \mid \mathrm{N(x).Q} \mid \ \mathrm{vx.\{M/x\}} \equiv \mathrm{vx.(\{M/x\}} \mid \mathrm{N<M>.P} \mid \mathrm{N(x).Q)} \equiv$$
$$\equiv \mathrm{vx.(\{M/x\}} \mid \mathrm{N<x>.P\ \{M/x\}} \mid \mathrm{N(x).Q)} \equiv \mathrm{vx.(\{M/x\}} \mid \mathrm{N<x>.P} \mid \mathrm{N(x).Q)} \rightarrow \mathrm{vx.}$$
$$\mathrm{(\{M/x\}} \mid \mathrm{P} \mid \mathrm{Q)} \equiv \mathrm{vx.(\{M/x\}|P\{M/x\}|Q\{M/x\})} \equiv \mathrm{vx.(\{M/x\}} \mid \mathrm{P} \mid \mathrm{Q\{M/x\})}$$
$$\equiv \mathrm{(vx.\{M/x\})} \mid \mathrm{P} \mid \mathrm{Q\{M/x\}} \equiv \mathrm{P} \mid \mathrm{Q\{M/x\}}$$

με την προϋπόθεση ότι x∉(*fv(P)* ∪ *fv(N)* ∪ *fv(M)).* Παρατηρούμε ότι στο βήμα της εσωτερικής αναγωγής, κάναμε χρήση του (i) από τον ορισμό 2.10 με *C[_] = vx.({M/x} | _)* όπως στο παράδειγμα στην αρχή της ενότητας αυτής.
Προτού προχωρήσουμε στην έννοια της παρατηρήσιμης ισοδυναμίας (observational equivalence), θα πρέπει να σημειώσουμε ότι στην παραπάνω παρουσίαση αγνοήσαμε τη δυνατότητα να έχουμε ένα σύστημα με sorts για τον εφαρμοσμένο π-λογισμό. Θα πρέπει

τότε για παράδειγμα, να επιτρέπουμε το input/output μόνο σε όρους με sort Channel. Για την παρουσίασή μας ένα τέτοιο σύστημα δεν παίζει σημαντικό ρόλο γι΄αυτό και αποφασίσαμε να το αγνοήσουμε. Οι αλλαγές στους ορισμούς είναι ελάχιστες αλλά ο ενδιαφερόμενος αναγνώστης μπορεί να απευθυνθεί στο [8].

2.3 Παρατηρήσιμη Ισοδυναμία

Με την έννοια της παρατηρήσιμης ισοδυναμίας προσπαθούμε να εκφράσουμε την κατάσταση εκείνη κατά την οποία ένας εχθρός δεν μπορεί να διακρίνει μεταξύ δύο διεργασιών. Ξεκινάμε με κάποιους ορισμούς που θα μας χρειαστούν στη συνέχεια. Από το (ix) του ορισμού 2.9, προκύπτει ότι μια ενεργή αντικατάσταση {M/x} "εξαπλώνεται" προς όλες τις διεργασίες με τις οποίες τρέχει παράλληλα και αντικαθιστά τις ελεύθερες εμφανίσεις της μεταβλητής x σε αυτές. Κατά κάποιον τρόπο η προηγούμενη ενεργή αντικατάσταση εκθέτει (exports) τον όρο M στο περιβάλλον μέσω της μεταβλητής-handle x. Ενδιαφερόμαστε για όλες αυτές τις μεταβλητές-handles μιας επεκτεταμένης διεργασίας A και τις συλλέγουμε στο σύνολο *dom(A)*:

Ορισμός 2.11. Έστω επεκτεταμένη διεργασία A. Ορίζουμε το σύνολο dom(A) ως εξής:

i. $\mathrm{dom}(\mathrm{P}) = \varnothing$, όπου $\mathrm{P} \in \mathrm{P}(\mathrm{T}_\Sigma)$ (απλή διεργασία)
ii. $\mathrm{dom}(\mathrm{A} \mid \mathrm{B}) = \mathrm{dom}(\mathrm{A}) \cup \mathrm{dom}(\mathrm{B})$
iii. $\mathrm{dom}(\nu n.\mathrm{A}) = \mathrm{dom}(\mathrm{A})$
iv. $\mathrm{dom}(\nu x.\mathrm{A}) = \mathrm{dom}(\mathrm{A}) \backslash \{\mathrm{x}\}$
v. $\mathrm{dom}(\{\mathrm{M/x}\}) = \{\mathrm{x}\}$

Παρατηρούμε ότι στο (iv) του προηγούμενου ορισμού, η μεταβλητή x εξαιρείται από το σύνολο διότι ακόμα και αν υπάρχει αντικατάσταση *{M/x}* στην A, αυτή δεν γίνεται export στο περιβάλλον. Σε αυτή την περίπτωση έχουμε ουσιαστικά μία *let* $x = M$ *in* κατασκευή, όπως είδαμε στην ενότητα 2.1.

Ορισμός 2.12. Μια επεκτεταμένη διεργασία A λέγεται κλειστή όταν $fv(\mathrm{A}) = \mathrm{dom}(\mathrm{A})$. Ένα περιβάλλον αποτίμησης C κλείνει την επεκτεταμένη διεργασία A όταν C[A] είναι κλειστή διεργασία.

Για παράδειγμα το *C[_]:= {enc(m,pkey,rand)/x} | _* κλείνει την *A:= Ch<sign(x,skey)>*, αφού *C[A] = {enc(m,pkey,rand)/x} | Ch<sign(x,skey)>* ≡ *{enc(m,pkey,rand)/x} | Ch<sign(enc(m,pkey,rand),skey)>* . Παρατηρούμε ότι *fv(A) = fv(C[A]) = {x}, fn(A) = {Ch, skey}* και *dom(C[A]) = {x},* επομένως ικανοποιείται ο ορισμός 2.12.

Ένας εχθρός θα μπορούσε εύκολα να διακρίνει μεταξύ δύο διεργασιών, αν η μία έκανε output σε ένα κανάλι c (όπου c ελεύθερο όνομα στις διεργασιές) ενώ η άλλη όχι. Είναι συνεπώς σημαντικό να θεωρήσουμε τα κανάλια στα οποία κάνει output μια διεργασία: γράφουμε $\mathrm{A}{\downarrow}\mathrm{c}$ όταν η A μπορεί να στείλει κάποιον όρο M στο κανάλι c.

Ο ορισμός της παρατηρήσιμης ισοδυναμίας επίσης εξετάζει τη δυναμική συμπεριφορά των διεργασιών, δηλαδή πώς μπορεί να εξελιχθεί μια διεργασία μέσω της εσωτερικής αναγωγής. Θέλουμε τότε, προκειμένου να δείξουμε για δύο διεργασίες ότι είναι ισοδύναμες, κάθε αναγωγή της μίας να αντιστοιχεί σε μία αναγωγή της άλλης και οι προκύπτουσες διεργασίες να είναι ισοδύναμες.
Τέλος, στα παραπάνω έχουμε αγνοήσει την ενεργή συμπεριφορά του εχθρού, τον οποίο θεωρήσαμε ως παθητικό παρατηρητή. Θα μπορούσε να υπάρχει κάποιος αλγόριθμος μέσω του οποίου να ξεχωρίζουν οι δύο διεργασίες που εξετάζονται. Δηλαδή με όρους του εφαρμοσμένου π-λογισμού, θα μπορούσε να υπάρχει κάποιο περιβάλλον αποτίμησης C, το οποίο να οδηγεί τις διεργασίες σε κάποιο υπολογισμό/output ο οποίος, με βάση τα παραπάνω, να τις κάνει να ξεχωρίζουν. Συνεπώς πρέπει να αποτραπεί και αυτή η περίπτωση.
Τα παραπάνω δικαιολογούν τον επόμενο ορισμό για την παρατηρήσιμη ισοδυναμία:

Ορισμός 2.13. Παρατηρήσιμη ισοδυναμία (observational equivalence) $\approx$ είναι η μεγαλύτερη συμμετρική σχέση στο $\mathrm{EP}(\mathrm{T}_\Sigma)^2$ για την οποία ισχύουν τα εξής για κάθε A, $\mathrm{B} \in \mathrm{EP}(\mathrm{T}_\Sigma)$ με $\mathrm{A} \approx \mathrm{B}$:

i. A και B είναι κλειστές με $\mathrm{dom(A)} = \mathrm{dom(B)}$.
ii. Αν $\mathrm{A}\downarrow c$ τότε $\mathrm{B}\downarrow c$.
iii. Αν $\mathrm{A} \to \mathrm{A}'$ με A' κλειστή τότε υπάρχει B' για την οποία $\mathrm{B} \to \mathrm{B}'$ με $\mathrm{A}' \approx \mathrm{B}'$.
iv. $\mathrm{C[A]} \approx \mathrm{C[B]}$ για κάθε περιβάλλον αποτίμησης C που κλείνει τις διεργασίες A και B.

Ως ένα απλό παράδειγμα παρατηρήσιμα ισοδύναμων διεργασιών έχουμε τις: *vm. c<m>* $\approx$ *vm. c<hash(m)>*, όπου *hash* σύμβολο συνάρτησης με ένα όρισμα, χωρίς εξισώσεις. Οι συνθήκες (i), (ii) και (iii) του ορισμού 2.13 είναι πολύ εύκολο να ελεχθούν: οι A, B είναι κλειστές με $dom(A) = \varnothing = dom(B)$, ισχύει ότι $\mathrm{A}\downarrow c$ και $\mathrm{B}\downarrow c$ και τέλος οι A και B δεν μπορούν αναγχθούν με βάση τη σχέση της εσωτερικής αναγωγής άρα η (3) αληθεύει εν κενώ. Η συνθήκη (4) από την άλλη πλευρά είναι πολύ δύσκολο να ελεγχθεί τόσο σε αποδείξεις με το χέρι όσο και αυτόματα. Γι' αυτό μπορεί να οριστεί μια νέα έννοια ισοδυναμίας διεργασιών που ονομάζεται labelled bisimilarity ($\approx_l$) [8]. Μπορεί να αποδειχθεί ότι αυτές οι δύο σχέσεις είναι ίσες ($\approx = \approx_l$). Η σχέση $\approx_l$ είναι όμως πιο εύκολο να ελεγχθεί, συνεπώς χρησιμοποιείται στις αποδείξεις.
Θα χρησιμοποιήσουμε την έννοια της παρατηρήσιμης ισοδυναμίας παρακάτω για να εκφράσουμε κάποιες ιδιότητες που θέλουμε να ικανοποιεί ένα πρωτόκολλο e-voting. Η ProVerif μπορεί να δώσει κάποιες αποδείξεις παρατηρήσιμης ισοδυναμίας και θα δούμε εν συντομία παρακάτω πως το κάνει. Στη γενική περίπτωση, το πρόβλημα της παρατηρήσιμης ισοδυναμίας είναι μη αποκρίσιμο (undecidable).

2.4 Γεγονότα και ιδιότητες αντιστοιχίας

Θα δώσουμε παρακάτω μια επέκταση του εφαρμοσμένου π-λογισμού την οποία θα χρησιμοποιήσουμε για να ορίσουμε κάποιες από τις ιδιότητες ασφαλείας που θέλουμε να ικανοποιεί το πρωτόκολλο που εξετάζουμε. Ουσιαστικά, επιτρέπουμε την ύπαρξη γεγονότων (events) στις διεργασίες, τα οποία μπορεί να περιλαμβάνουν όρους του T_Σ και τα οποία είναι διαφανή (transparent) για τον εχθρό. Θα διατυπώσουμε έπειτα κάποιες ιδιότητες που συνδέουν διαφορετικά γεγονότα και τους όρους που αυτά πιθανώς να περιέχουν. Οι ιδιότητες αυτές ονομάζονται ιδιότητες αντιστοιχίας (correspondence assertions) ή αντιστοιχίες (correspondences). Η ProVerif περιλαμβάνει τόσο γεγονότα στις διεργασίες της όσο και τη δυνατότητα διατύπωσης ερωτημάτων σχετικά με ιδιότητες αντιστοιχίας (βλ. κεφάλαιο 3). Για τους παρακάτω ορισμούς βλ. [28].
Αρχικά θεωρούμε στην υπογραφή Σ ένα αριθμήσιμα άπειρο σύνολο συμβόλων γεγονότων (πχ e_1, e_2, ΜΥEVENT κλπ). Τα σύμβολα αυτά δεν συμμετέχουν στο σύνολο των όρων αλλά το σύνολο των διεργασιών $P(T_\Sigma)$ επεκτείνεται ώστε να περιλαμβάνει την εξής διεργασία: $e_1(M).Q \in P(T_\Sigma)$, όπου $Q \in P(T_\Sigma)$ και $M \in T_\Sigma$. Η σημασιολογία της προηγούμενης διεργασίας είναι εξαιρετικά απλή: το γεγονός εκτελείται "σιωπηλά" και έπειτα εκτελείται η διεργασία Q. Θα πρέπει δηλαδή ο ορισμός της σχέσης της εσωτερικής αναγωγής → (ορισμός 2.10) να επεκταθεί με την εξής ιδιότητα: $e_1(M).Q \rightarrow Q$. Δίνουμε παρακάτω μερικούς ακόμη ορισμούς:

Ορισμός 2.14. Έστω $P \in EP(T_\Sigma)$ διεργασία. Μια ακολουθία διεργασιών P_1, P_2, .., P_n τέτοια ώστε $P \rightarrow P_1 \rightarrow P_2 \rightarrow ... \rightarrow P_n$ θα ονομάζεται ίχνος (trace) της P.

Ορισμός 2.15. Έστω $P \in EP(T_\Sigma)$ και t κάποιο ίχνος της. Θα λέμε ότι στο ίχνος t εκτελείται το γεγονός $e \in \Sigma$ με όρο $M \in T_\Sigma$ αν το ίχνος t περιλαμβάνει εσωτερική αναγωγή της μορφής $C[e(M).Q] \rightarrow C[Q]$ για κάποια διεργασία Q και κάποιο περιβάλλον αποτίμησης C (πιθανώς και το ταυτοτικό). Θα λέμε ότι το προηγούμενο γεγονός εκτελείται στο κ-οστό βήμα όταν η κ-οστή αναγωγή του t είναι της προηγούμενης μορφής.

Μια αντιστοιχία είναι μια παράσταση της μορφής $e_1(M) \Rightarrow e_2(N)$, όπου e_1, e_2 σύμβολα γεγονότων του Σ και M, N όροι του T_Σ. Ο επόμενος ορισμός εξηγεί τι σημαίνει ότι μια διεργασία ικανοποιεί μια αντιστοιχία:

Ορισμός 2.16. Έστω διεργασία P και $e_1(M) \Rightarrow e_2(N)$ ιδιότητα αντιστοιχίας. Θα λέμε ότι η P ικανοποιεί την προηγούμενη αντιστοιχία όταν για κάθε ίχνος t της P, αν στο ίχνος t εκτελείται το γεγονός e_1 με όρο $M' \in T_\Sigma$ στο κ-οστό βήμα και υπάρχει αντικατάσταση σ τέτοια ώστε $M\sigma = M'$, τότε στο ίχνος t εκτελείται το γεγονός e_2 με όρο $N' = N\sigma$ στο λ-οστό βήμα με $\lambda \leq \kappa$.

Στις πιο απλές περιπτώσεις αντιστοιχίας οι όροι M και N είναι μία μεταβλητή x. Για μια διεργασία που ικανοποιεί την αντιστοιχία $e_1(x) \Rightarrow e_2(x)$, έχουμε τότε ότι αν εκτελείται το γεγονός e_1 με κάποιον όρο M τότε έχει προηγηθεί εκτέλεση του e_2 με τον ίδιο όρο. Παρατηρούμε ότι είναι δυνατόν πολλαπλές εκτελέσεις του e_1 να έπονται μοναδικής εκτέλεσης του e_2 και παρ'όλα αυτά να συνεχίζει να ικανοποιείται μια αντιστοιχία της προηγούμενης μορφής. Αν θέλουμε να αποκλείσουμε μια τέτοια περίπτωση μπορούμε να χρησιμοποιήσουμε την έννοια της ένα-προς-ένα αντιστοιχίας (injective correspondence):

Ορισμός 2.17. Έστω διεργασία P και $e_1(M) \Rightarrow^{inj} e_2(N)$ ένα-προς-ένα ιδιότητα αντιστοιχίας. Θα λέμε ότι η P ικανοποιεί την προηγούμενη αντιστοιχία όταν για κάθε ίχνος t της P, υπάρχει ένα προς ένα συνάρτηση φ: $\mathbb{N} \to \mathbb{N}$ τέτοια ώστε αν στο ίχνος t εκτελείται το γεγονός e_1 με όρο $M' \in T_\Sigma$ στο κ-οστό βήμα και υπάρχει αντικατάσταση σ τέτοια ώστε $M\sigma = M'$, τότε στο ίχνος t εκτελείται το γεγονός e_2 με όρο $N' = N\sigma$ στο $\varphi(\kappa)$-οστό βήμα με $\varphi(\kappa) \leq \kappa$.

Μπορούμε ακόμη να μιλήσουμε για εμφωλευμένες αντιστοιχίες (nested correspondences) της μορφής $e_1(x) \Rightarrow (e_2(x) \Rightarrow e_3(x))$. Η σημασία της προηγούμενης αντιστοιχίας είναι, διαισθητικά, ότι μια εκτέλεση του e_1 έπεται μιας εκτέλεσης του e_2, η οποία έπεται μιας εκτέλεσης του e_3, όπου όλα τα γεγονότα εκτελούνται με τον ίδιο όρο. Παραπέμπουμε στο [9] για μια πληρέστερη κάλυψη των αντιστοιχιών που μπορεί να ελέγξει η ProVerif. Στην ενότητα 3.5 δίνουμε σε ProVerif ένα εκτενές παράδειγμα ιδιότητας αντιστοιχίας σε ένα μικρό πρωτόκολλο αυθεντικοποίησης.

Κεφάλαιο 3

3.1 Η ProVerif και η γλώσσα μοντελοποίησης

Η ProVerif είναι ένα εργαλείο αυτόματης ανάλυσης κρυπτογραφικών πρωτοκόλλων. Το πρωτόκολλο διατυπώνεται σε μια γλώσσα μοντελοποίησης υψηλού επιπέδου, η οποία είναι μια διάλεκτος του εφαρμοσμένου π-λογισμού. Έπειτα, η ProVerif μεταφράζει το μοντέλο αυτό σε προτάσεις Horn που αφορούν κάποια κατηγορήματα. Σε κάθε μοντέλο που αναλύεται, πρέπει να είναι ρητά ή άρρητα διατυπωμένο ένα ερώτημα (query) το οποίο θα μεταφραστεί κατάλληλα σε κάποια πρόταση Horn στόχο (goal clause), την οποία έπειτα θα προσπαθήσει το σύστημα να αποδείξει. Ο αλγόριθμος που χρησιμοποιείται είναι μια παραλλαγή του αλγορίθμου της ανάλυσης (resolution). Ξεκινάμε τώρα με την παρουσίαση της γλώσσας μοντελοποίησης που χρησιμοποιεί η ProVerif και στην ενότητα 3.3 θα εξετάσουμε αναλυτικότερα τη μετάφραση που πραγματοποιείται κατά τη διαδικασία της ανάλυσης ενός πρωτοκόλλου.

Όπως είπαμε ήδη, η γλώσσα μοντελοποίησης της ProVerif μοιάζει πολύ με τον εφαρμοσμένο π-λογισμό. Θα παρουσιάσουμε αναλυτικότερα ένα υποσύνολο της γλώσσας αυτής, αφού η ProVerif είναι ένα εργαλείο με αρκετά χρόνια ιστορίας και κατά συνέπεια έχει πολλές κατασκευές για να καλύψει όσο περισσότερες περιπτώσεις γίνεται στη μοντελοποίηση ενός πρωτοκόλλου, με περισσότερες δυνατότητες να προστίθενται κατά καιρούς. Επίσης, η ProVerif δέχεται input σε διαφορετικές μορφές της ίδιας ουσιαστικά γλώσσας: με τύπους και χωρίς τύπους. Εδώ θα επικεντρωθούμε στη μορφή χωρίς τύπους διότι αυτή χρησιμοποιήσαμε στα σενάρια (scripts) μοντελοποίησης που φτιάξαμε. Οι τύποι (types) υπάρχουν για τη διευκόλυνση του χρήστη (αποσφαλμάτωση (debugging) του script) και δεν παίζουν ρόλο στην ανάλυση του πρωτοκόλλου. Βασική αναφορά για τα επόμενα είναι τα [29] και [30].

Όπως και στον εφαρμοσμένο π-λογισμό, ένα script ξεκινάει με τον ορισμό συμβόλων συναρτήσεων, το καθένα συσχετισμένο με κάποιο arity, για παράδειγμα: *fun enc/3.* Επίσης μπορούμε στην αρχή του script να ορίσουμε τα ελεύθερα ονόματα που θα εμφανίζονται στις διεργασίες μας: *free channel. private free my_ secret.* Στη πρώτη περίπτωση έχουμε δηλώσει ένα ελεύθερο όνομα το οποίο είναι γνωστό στον εχθρό (ουσιαστικά θα χρησιμοποιηθεί ως ένα δημόσιο κανάλι επικοινωνίας), ενώ στη δεύτερη περίπτωση το ελεύθερο όνομα *my_ secret* δεν είναι γνωστό στον εχθρό. Τέλος στα scripts μας μπορούν να εμφανίζονται (δεσμευμένες) μεταβλητές.

Με βάση τα παραπάνω (που αντιστοιχούν στην υπογραφή Σ του εφαρμοσμένου π-λογισμού) δημιουργείται το σύνολο των όρων T όπως ακριβώς στον ορισμό 2.1. Μπορούμε εναλλακτικά να θεωρήσουμε ότι το σύνολο T των όρων παράγεται με την εξής γραμματική, όπου τα σύμβολα x, M, s κλπ αποτελούν τη σύμβαση για την αναφορά σε αντικείμενα του κάθε είδους:

M,N (όροι) ::=

x, y, z (μεταβλητές)
a, b, c, k, s (ονόματα)
$f(M_1, M_2, ..., M_k)$ (εφαρμογή συνάρτησης στους όρους M_1, M_2, ..., M_k)

όπου, κατά τα γνωστά, το σύμβολο συνάρτησης f πρέπει να έχει οριστεί ως *fun f/k*. Τα συναρτησιακά σύμβολα με arity 0 θεωρούνται σταθερές. Οι σταθερές true και false είναι built-in στην ProVerif.
Στον εφαρμοσμένο π-λογισμό, όπως είδαμε στην ενότητα 2.1, είναι δυνατό να οριστεί μια θεωρία ισότητας η οποία βασίζεται σε ένα σύνολο από αρχικές "εξισώσεις" μεταξύ όρων. Το ίδιο μπορούμε να κάνουμε και στην ProVerif. Θεωρούμε ένα σύνολο από εξισώσεις E. Οι εξισώσεις στην ProVerif γράφονται χρησιμοποιώντας τη λέξη-κλειδί *equation*. Για παράδειγμα μπορούμε να έχουμε: *equation check(sign(skey,m),pk(skey))=m*, όπου το συναρτησιακό σύμβολο *sign* έχει οριστεί ως *fun sign/2*, το σύμβολο *pk* έχει οριστεί ως *fun pk/1* και το *check* ως *fun check/2*. Τα *skey* και *m* είναι μεταβλητές. Η προηγούμενη εξίσωση μοντελοποιεί το primitive των ψηφιακών υπογραφών. Από το σύνολο των αρχικών εξισώσεων E που γράφεται στο script προκύπτει η θεωρία ισότητας $=_E$ όπως ακριβώς στον ορισμό 2.3.
Στην ProVerif είναι δυνατόν εκτός από εξισώσεις, να έχουμε και αναγωγές δηλαδή κανόνες μεταγραφής (rewrite rules). Προφανώς, οι εξισώσεις και η θεωρία ισότητας που προκύπτει από αυτές είναι πιο κοντά στη θεωρία του εφαρμοσμένου π-λογισμού, όμως οδηγούν την ProVerif σε πιο χρονοβόρες αναλύσεις. Γι' αυτό προτείνεται στο [9], να προτιμώνται οι αναγωγές όπου αυτό είναι δυνατόν. Επίσης η ProVerif δεν μπορεί να χειριστεί όλες τις θεωρίες ισότητας. Στην πραγματικότητα, όλες οι εξισώσεις μετατρέπονται εσωτερικά καταλλήλως σε κανόνες μεταγραφής. Κατά συνέπεια, οι θεωρίες ισότητας που υποστηρίζει η ProVerif περιορίζονται σε όσες μια τέτοια μετάφραση είναι δυνατόν να γίνει. Για τις συγκεκριμένες προϋποθέσεις που πρέπει να ισχύουν, παραπέμπουμε στο [29].
Προκειμένου να συμπεριλάβουμε τους κανόνες μεταγραφής ορίζουμε το σύνολο των εκφράσεων Exp. Στους κανόνες μεταγραφής συμμετέχουν όροι και κάποια σύμβολα που θυμίζουν σύμβολα συναρτήσεων αλλά λέγονται destructors. Τα σύμβολα των destructors δεν ορίζονται ρητά όπως τα σύμβολα συναρτήσεων αλλά η ProVerif καταγράφει το όνομά τους και το πλήθος των ορισμάτων τους από τους κανόνες μεταγραφής στους οποίους εμφανίζονται. Θεωρούμε συνεπώς ότι έχουμε στη διάθεσή μας ένα σύνολο από σύμβολα για destructors το καθένα συσχετισμένο με κάποιο arity.

Ορισμός 3.1. Έστω σύνολο όρων T. Το σύνολο των εκφράσεων Exp ορίζεται επαγωγικά ως εξής:

- fail ∈ Exp
- Αν M ∈ T τότε M ∈ Exp
- Αν $D_1, D_2, ..., D_n \in$ Exp και h σύμβολο destructor με arity n τότε $h(D_1,D_2, ...,D_n)\in$Exp

Το σύνολο Exp περιλαμβάνει τη σταθερά *fail*, η οποία χρησιμοποιείται στις περιπτώσεις που δεν υπάρχει κατάλληλος κανόνας μεταγραφής. Έτσι οι εκφράσεις αποτιμώνται σε όρους ή σε fail (βλ. ενότητα 3.2). Υπάρχουν επίσης, κάποιοι built-in destructors όπως αυτοί της ισότητας ('=') και της ανισότητας ('<>'). Δίνουμε ένα παράδειγμα που θα χρησιμοποιηθεί και παρακάτω. Υποθέτουμε ότι έχουμε σύμβολα συνάρτησεων *pk* και *enc* ορισμένα ως: *fun pk/1*, *fun enc/2*. Θεωρούμε τότε τον κανόνα για την αποκρυπτογράφηση δημοσίου κλειδιού ως εξής: *reduc dec(enc(pk(skey),m),skey)=m.*, όπου βλέπουμε τη λέξη-κλειδί *reduc* με την οποία ορίζουμε στην ProVerif κανόνες μεταγραφής. Από τον προηγούμενο κανόνα προκύπτει για την ProVerif ότι το σύμβολο *dec* είναι ένας destructor με δύο ορίσματα. Σημειώνουμε ότι οι συνήθεις λογικοί σύνδεσμοι (όπως *and* και *or*) μπορούν εύκολα να οριστούν ως destructors. Για παράδειγμα θα έχουμε για το *and*: *reduc and(true, true)=true.*
Δίνουμε παρακάτω τον ορισμό του συνόλου των διεργασιών της ProVerif, οι οποίες διεργασίες σαφώς θυμίζουν αυτές του εφαρμοσμένου π-λογισμού:

Ορισμός 3.2. Έστω T σύνολο όρων και Exp σύνολο εκφράσεων. Το σύνολο των διεργασιών της ProVerif Pr ορίζεται επαγωγικά ως εξής:

- 0 ∈ Pr
- out(N,M); P ∈ Pr
- in(N, x); P ∈ Pr
- P | Q ∈ Pr
- !P ∈ Pr
- new a; P ∈ Pr
- let x=D in P else Q ∈ Pr
- if M then P else Q ∈ Pr

Στα παραπάνω έχουμε: P,Q ∈ Pr, D ∈ Exp, a όνομα, x μεταβλητή και N,M ∈ T.

Εξηγούμε συνοπτικά τις παραπάνω διεργασίες (βλ. ενότητα 3.2 για τυπική σημασιολογία). Με τη διεργασία *new a; P* γράφουμε σε ProVerif τη διεργασία *va.P* του εφαρμοσμένου π-λογισμού. Στη διεργασία *let x=D in P else Q*, αν η έκφραση D μπορεί να αποτιμηθεί σε κάποιον όρο M, εκτελείται η P με τις ελεύθερες εμφανίσεις της x να έχουν αντικατασταθεί με M. Αν η D δεν μπορεί να αποτιμηθεί σε όρο (αποτιμάται δηλαδή σε fail) τότε εκτελείται η διεργασία Q. Με εξαίρεση την πιθανή αποτυχία και τον *else* κλάδο, η διεργασία αυτή αντιστοιχεί στην *vx.({M/x} | P)* του εφαρμοσμένου π-λογισμού που είδαμε στην ενότητα 2.1, υποθέτοντας ότι η έκφραση D αποτιμάται στον όρο M. Τέλος η διεργασία *if M then P else Q*, προσπαθεί να αποτιμήσει τον όρο *M* (βλ. ενότητα 3.2 - μην ξεχνάμε ότι κάθε όρος είναι πλέον έκφραση). Αν ο όρος αυτός αποτιμάται σε *true* τότε εκτελείται η P. Σε οποιαδήποτε άλλη περίπτωση εκτελείται η Q.
Όπως αναφέραμε στην εισαγωγή του κεφαλαίου αυτού, η ProVerif αναλύει τις διεργασίες που τις δίνονται με σκοπό να απαντήσει σε κάποιο ερώτημα που επίσης της δίνεται ως input. Τα ερωτήματα αυτά μπορούν να υπονοούνται (implicit), όπως στην περίπτωση της παρατηρήσιμης ισοδυναμίας που θα δούμε παρακάτω (βλ. ενότητα 3.4). Στις περισσότερες

περιπτώσεις όμως διατυπώνονται χρησιμοποιώντας τη λέξη-κλειδί *query*. Το πιο απλό είδος ερωτήματος αφορά τη "μυστικότητα" κάποιου (ελεύθερου) ονόματος και συντάσσεται ως εξής: *query attacker: free_name.* Με το ερώτημα αυτό ουσιαστικά προσπαθούμε να ελέγξουμε αν υπάρχει κάποια "επίθεση" στο πρωτόκολλο έτσι ώστε τελικά κάποιος εχθρός (attacker) να "γνωρίζει" το όνομα *free_name.* Χρησιμοποιούμε το ερώτημα αυτό με τη διαισθητική αυτή έννοια προς το παρόν στο παράδειγμα που ακολουθεί (βλ. ενότητα 3.2 για τον αυστηρό ορισμό). Στην ενότητα 3.4 θα δούμε επιπλέον ερωτήματα που σχετίζονται με γεγονότα (events) και ιδιότητες αντιστοιχίας.
Δίνουμε παρακάτω ένα απλό παράδειγμα διεργασίας. Πέρα από τα προηγούμενα, παρατηρούμε σε αυτό τη δυνατότητα να οριστεί μια διεργασία με τη βοήθεια υποδιεργασιών που ορίζονται με εκφράσεις της μορφής *let Subprocess = .* Αυτές οι κατασκευές λειτουργούν με έναν τρόπο επέκτασης μακροεντολών (macro expansion): η ProVerif απλώς αντικαθιστά τον ορισμό τους στο σώμα της κύριας διεργασίας.

```
free channel.
private free my_secret.

fun sign/2.
fun enc/2.
fun pk/1.
fun symenc/2.

reduc dec(enc(pk(skey),m),skey) = m.
reduc ver(sign(skey,m),pk(skey)) = m.
reduc symdec(symenc(key,m),key) = m.

query attacker: my_secret.

let A =
     in(channel,pkey);
     new session_key;
     out(channel,enc(pkey,sign(skeyA,session_key)))
     .

let B =
     out(channel,pk(skeyB));
     in(channel,mess);
     let dec_msg = dec(mess,skeyB) in
     let ses_key = ver(dec_msg,pk(skeyA)) in
     out(channel,symenc(ses_key,my_secret))
     .


process
     new skeyA;
     new skeyB;
     out(channel,pk(skeyA));
     out(channel,pk(skeyB));

     A | B
```

3.1 Ένα παράδειγμα πρωτοκόλλου

Στο παραπάνω παράδειγμα (πίνακας 3.1) έχουμε ένα απλό πρωτόκολλο για την ανταλλαγή κλειδιών συνεδρίας (session keys). Ο A περιμένει να του δοθεί ένα δημόσιο κλειδί στο οποίο θα στείλει κρυπτογραφημένο και υπογεγραμμένο το κλειδί για τη συνεδρία. Ο B είναι ο νόμιμος χρήστης (honest user) ο οποίος ακολουθεί το πρωτόκολλο και αναμένει το κλειδί συνεδρίας που θα αποκτήσει να είναι μυστικό: κατά συνέπεια θα πρέπει το μήνυμα my_secret να παραμένει μυστικό. Αυτό είναι και το ερώτημα που έχουμε θέσει στην ProVerif (*query attacker: my_secret.*). Τέλος παρατηρούμε τις δηλώσεις συναρτησιακών συμβόλων και τους κανόνες μεταγραφής, που μοντελοποιούν τα primitives της αποκρυπτογράφησης δημοσίου κλειδιού, της επαλήθευσης μιας ψηφιακής υπογραφής και της συμμετρικής αποκρυπτογράφησης αντίστοιχα.
Κατά την ανάλυση του προηγούμενου πρωτοκόλλου (με την εντολή *proverif -in pi script.pv*) η ProVerif καταφέρνει να βρει μια επίθεση, αποδεικνύοντας έτσι ότι το όνομα-μήνυμα *my_secret* δεν παραμένει μυστικό παρά την κρυπτογράφησή του. Πρόκειται για μια επίθεση τύπου man-in-the-middle την οποία περιγράφουμε παρακάτω. Αρχικά ο εχθρός γνωρίζει τα δημόσια κλειδιά του A και του B (*pk(skeyA)* και *pk(skeyB)* αντίστοιχα). Έπειτα δημιουργεί ένα νέο ζεύγος ιδιωτικού και δημόσιου κλειδιού, που τα ονομάζουμε *skeyC* και *pk(skeyC)* (η ProVerif χρησιμοποιεί διαφορετικά ονόματα αλλά αυτό είναι αδιάφορο). Ο εχθρός στέλνει το *pk(skeyC)* στον A ο οποίος απαντά με ένα κρυπτογραφημένο (με το δημόσιο κλειδί του εχθρού) και υπογεγραμμένο κλειδί συνόδου *session_key* (πρόκειται για τον όρο *enc(pk(skeyC), sign(skeyA, session_key))*). Ο εχθρός αποκρυπτογραφεί το κλειδί αυτό και έπειτα κρυπτογραφεί το *sign(skeyA,session_key)* με το δημόσιο κλειδί του B (στέλνει δηλαδή τον όρο *enc(pk(skeyB),sign(skeyA,session_key))*). Ο B θεωρεί ότι αυτό το μήνυμα προέρχεται από τον A και άρα το χρησιμοποιεί για να στείλει το "μυστικό" *my_secret* στον A, κρυπτογραφημένο με συμμετρική κρυπτογράφηση χρησιμοποιώντας ως συμμετρικό κλειδί το *session_key* που μόλις έλαβε. Ο εχθρός παραλαμβάνει το μήνυμα αυτό και αποκρυπτογραφεί το μήνυμα (αφού διαθέτει το *session_key*) αποκτώντας έτσι το *my_secret*.

3.2 Σημασιολογία

Η σημασιολογία της γλώσσας μοντελοποίησης ακολουθεί ουσιαστικά τη σημασιολογία του εφαρμοσμένου π-λογισμού που δόθηκε στην ενότητα 2.2, με κάποιες διαφορές και επεκτάσεις λόγω των αναγωγών. Θα παρουσιάσουμε τις βασικές περιπτώσεις. Για περισσότερες λεπτομέρειες παραπέμπουμε στο [29].
Αρχικά ορίζουμε τη σχέση D ↓ M, με τη σημασία ότι η έκφραση D αποτιμάται στον όρο M.

Ορισμός 3.3. Έστω υπογραφή Σ με τα συναρτησιακά σύμβολα ενός script. Έστω επίσης η θεωρία ισότητας $=_E$ (ορισμός 2.3) που προκύπτει από τις εξισώσεις που γράφονται στο script. Θεωρούμε ότι για κάθε συναρτησιακό σύμβολο h k-θέσεων, έχουμε ένα σύνολο με τους κανόνες μεταγραφής του rules(h), το οποίο περιλαμβάνει κανόνες της μορφής $h(M_1, M_2, ..., M_k) \rightarrow M$, όπου $M_1, M_2, ..., M_k, M$ όροι του T_Σ. Δεδομένου του ορισμού 3.1 για το σύνολο των εκφράσεων Exp, ορίζουμε τη σχέση D ↓ M επαγωγικά ως εξής:

i. M ↓ M, για κάθε όρο M.
ii. fail ↓ fail
iii. $h(D_1, D_2, ..., D_k)$ ↓ fail αν για κάποιο i είναι D_i ↓ fail. Έστω $D_1 \downarrow M_1$, $D_2 \downarrow M_2$, ..., $D_k \downarrow M_k$. Τότε αν υπάρχει κάποια αντικατάσταση σ και κάποιος κανόνας στο rules(h) της μορφής $h(M'_1, M'_2, ..., M'_k) \rightarrow M'$ ώστε να ισχύει $M'_1\sigma =_E M_1$, $M'_2\sigma =_E M_2$, ..., $M'_k\sigma =_E M_k$, έχουμε $h(D_1, D_2, ..., D_k) \downarrow M'\sigma$. Αν δεν υπάρχει τέτοιος κανόνας έχουμε $h(D_1, D_2, ..., D_k)$ ↓ fail.

Ο παραπάνω ορισμός περιλαμβάνει το ταίριασμα προτύπων που γίνεται στους κανόνες μεταγραφής. Δεν έχουμε εξετάσει την περίπτωση στην οποία έχουμε ταίριασμα με παραπάνω του ενός κανόνα. Υπάρχουν εντολές στην ProVerif με τις οποίες ορίζεται μια αρίθμηση στους κανόνες, οπότε δοκιμάζονται με τη σειρά και ενεργοποιείται ο πρώτος που μπορεί να ταιριάξει. Αν κάτι τέτοιο δεν έχει γίνει, η ProVerif αναγκάζεται να αναλύσει όλες τις περιπτώσεις. Σημειώνουμε ότι η περίπτωση που έχουμε μόνο έναν κανόνα μεταγραφής για κάθε destructor, μας καλύπτει τόσο στο παράδειγμα της ενότητας 3.1 όσο και στο script που θα αναπτύξουμε στο κεφάλαιο 4.
Ορίζουμε τώρα τη σημασιολογία των διεργασιών στην ProVerif. Ξεκινάμε με τον ορισμό των σημασιολογικών καταστάσεων (semantic configuration) και έπειτα ορίζουμε μια σχέση αναγωγής πάνω στις σημασιολογικές καταστάσεις.

Ορισμός 3.4. Έστω υπογραφή Σ με τα συναρτησιακά σύμβολα ενός script. Σημασιολογική κατάσταση με βάση το Σ είναι ένα ζεύγος $(E, P)_\Sigma$ όπου E ονομάζεται περιβάλλον (environment) και είναι $E = (N_{pub}, N_{priv})$ όπου N_{pub}, N_{priv} πεπερασμένα σύνολα ονομάτων. Το P είναι ένα πεπερασμένο πολυσύνολο (multiset) από διεργασίες,

όπως αυτές παράγονται από την παραπάνω γραμματική των διεργασιών και των εκφράσεων, με όρους από το T_Σ.

Τα σύνολα N_{pub}, N_{priv} θα διατηρούν κατά τον υπολογισμό μιας διεργασίας τα ονόματα που είναι δημοσίως γνωστά και ιδιωτικά αντίστοιχα.

Ορισμός 3.5. Έστω υπογραφή Σ με τα συναρτησιακά σύμβολα ενός script. Έστω επίσης η θεωρία ισότητας $=_E$ (ορισμός 2.3) που προκύπτει από τις εξισώσεις που γράφονται στο script. Τέλος, έστω η σχέση ↓ του ορισμού 3.3. Η σχέση αναγωγής → ορίζεται στο χώρο των σημασιολογικών καταστάσεων με βάση το Σ και είναι η μικρότερη μεταβατική διμελής σχέση για την οποία ισχύουν τα εξής:

- $(E, P \cup \{0\}) \rightarrow (E, P)$
- $(E, P \cup \{Q \mid R\}) \rightarrow (E, P \cup \{Q, R\})$
- $(E, P \cup \{!Q\}) \rightarrow (E, P \cup \{Q, !Q\})$
- $((N_{pub}, N_{priv}), P \cup \{\text{new a; } Q\}) \rightarrow ((N_{pub}, N_{priv} \cup \{a\}, P \cup \{Q\})$, όπου a δεν ανήκει στο $N_{pub} \cup N_{priv}$
- $(E, P \cup \{\text{out(N,M); } Q, \text{in(N',x); } R\}) \rightarrow (E, P \cup \{Q, R\{M/x\}\})$, όπου $N =_E N'$.
- $(E, P \cup \{\text{let x=D in Q else R}\}) \rightarrow (E, P \cup \{Q\{M/x\}\})$, όπου $D \downarrow M$.
- $(E, P \cup \{\text{let x=D in Q else R}\}) \rightarrow (E, P \cup \{R\})$, όπου $D \downarrow \text{fail}$.
- $(E, P \cup \{\text{if M then Q else R}\}) \rightarrow (E, P \cup \{Q\})$, όπου $M =_E \text{true}$.
- $(E, P \cup \{\text{if M then Q else R}\}) \rightarrow (E, P \cup \{R\})$, όπου $M \neq_E \text{true}$.

Για ένα δεδομένο script έχουμε αρχικά N_{pub} := *{τα names που έχουν δηλωθεί με free}*, N_{priv} := *{τα names που έχουν δηλωθεί με private free}* και P := *{το σώμα του process}*. Έτσι στο παραπάνω παράδειγμα της ενότητας 3.1 έχουμε αρχικά: N_{pub} = *{channel}*, N_{priv} = *{my_secret}* και P = *{new skeyA; new skeyB; ....}*. Μετά το πρώτο βήμα έχουμε: N_{pub} = *{ch}*, N_{priv} = *{my_secret, skeyA}* και P = *{new skeyB; ....}*.
Στην ProVerif θεωρούμε ότι τα πρωτόκολλα που εξετάζουμε τρέχουν παράλληλα με κάποιον εχθρό Dolev-Yao (βλ. ενότητα 1.2). Ο εχθρός αυτός δεν είναι παρά άλλη μία διεργασία με πρόσβαση όμως στα ελεύθερα ονόματα N_{pub} της διεργασίας που μοντελοποιεί το εκάστοτε πρωτόκολλο. Μια τέτοια κλειστή διεργασία Q, για την οποία ισχύει ότι $N_{pub} \subseteq fn(Q)$, ονομάζεται N_{pub}-εχθρός (N_{pub}-adversary).
Μπορούμε τώρα να ορίσουμε τυπικά την έννοια της μυστικότητας (secrecy) ενός όρου M, στην οποία αναφερθήκαμε στο παράδειγμα της ενότητας 3.1. Κατ' αρχάς, θα λέγαμε κάπως άτυπα, ότι μια διεργασία P διατηρεί τη μυστικότητα του M όταν, ενώ τρέχει παράλληλα με τυχαίο N_{pub}-εχθρό, ο όρος M δεν γίνεται output σε δημόσιο κανάλι. Στον επόμενο ορισμό εξηγούμε με σαφήνεια τι σημαίνει για έναν όρο M να γίνει output δημοσίως. Αντίστοιχα με τον ορισμό 2.14, μια ακολουθία σημασιολογικών καταστάσεων $(E_1, P_1)_\Sigma$, $(E_2, P_2)_\Sigma$, ..., $(E_n, P_n)_\Sigma$ για τις οποίες ισχύει ότι $(E, \{P\})_\Sigma \rightarrow (E_1, P_1)_\Sigma \rightarrow (E_2, P_2)_\Sigma \rightarrow ... \rightarrow (E_n, P_n)_\Sigma$, θα λέγεται ίχνος της P.

Ορισμός 3.6. Έστω διεργασία P∈Pr και όρος M∈T. Θα λέμε ότι στο ίχνος Tr της P, με αρχική σημασιολογική κατάσταση $((N_{pub,}\ N_{priv})$, {P}), ο όρος M γίνεται output δημοσίως όταν το Tr περιλαμβάνει αναγωγή (E, $P \cup$ {out(N, M); Q, in(N', x); R}) → (E, $P \cup$ {R{M/x}}), για κάποιο περιβάλλον E, πολυσύνολο διεργασιών P, διεργασίες Q, R και μεταβλητή x, με τον όρο N ∈ N_{pub}.

Ουσιαστικά, όπως προαναφέραμε, το N_{pub} διατηρεί μεταξύ άλλων τα δημόσια κανάλια επικοινωνίας, τα οποία μπορεί να παρακολουθεί ο εχθρός. Κατά συνέπεια, για να γίνει ένας όρος δημόσια γνωστός αρκεί να εμφανιστεί σε ένα τέτοιο κανάλι κατά τον υπολογισμό μιας διεργασίας.
Η έννοια της μυστικότητας ενός όρου M ορίζεται τυπικά παρακάτω:

Ορισμός 3.7. Η κλειστή διεργασία P_0 με $fn(P_0) \subseteq N_{pub}$ διατηρεί τη μυστικότητα του όρου M ως προς το N_{pub}, όταν για κάθε N_{pub}-εχθρό Q και για κάθε ίχνος Tr της P με αρχική σημασιολογική κατάσταση $((N_{pub,}\ N_{priv})$, {P_0, Q}), ο όρος M δεν γίνεται output δημοσίως στο Tr.

Με το ερώτημα *query attacker: my_ secret* που είδαμε στην ενότητα 3.1, η ProVerif προσπαθεί να αποδείξει ότι η διεργασία P που αντιστοιχεί στο παραπάνω πρωτόκολλο διατηρεί τη μυστικότητα του όρου *my_ secret* ως προς το $N_{pub} := \{channel\}$.

3.3 Μετάφραση και αλγόριθμος ανάλυσης

Κατά τη διαδικασία της ανάλυσης ενός πρωτοκόλλου, η είσοδος που δέχεται η ProVerif στην γλώσσα υψηλού επιπέδου που παρουσιάσαμε στην ενότητα 3.1 μεταφράζεται σε προτάσεις Horn [31]. Θα παρουσιάσουμε παρακάτω τη διαδικασία της μετάφρασης και τον τροποποιημένο αλγόριθμο της ανάλυσης (resolution) που εκτελείται στις προτάσεις αυτές, όπως αυτά παρουσιάστηκαν για πρώτη φορά στο [32]. Μια πιο λεπτομερής και επεκτεταμένη διαδικασία για τη μετάφραση υπάρχει στο [29].
Ξεκινάμε με τη διαδικασία της μετάφρασης. Σκοπός της διαδικασίας είναι η παραγωγή προτάσεων Horn που αφορούν κάποια κατηγορήματα και περιγράφουν το πρωτόκολλο ορθά, όσον αφορά τη σημασιολογία που δόθηκε στην ενότητα 3.2. Οι προτάσεις αυτές είναι της μορφής:

$$F_1 \wedge F_2 \wedge ... \wedge F_n \rightarrow F$$

όπου $\{F_i\}$ και F είναι κατηγορήματα της μορφής $p(M_1, M_2, ..., M_k)$ με $\{M_i\}$ όρους. Σημειώνουμε ότι είναι δυνατον $n = 0$ στην παραπάνω πρόταση οπότε τότε έχουμε το λεγόμενο γεγονός (fact) F. Χρησιμοποιούνται δύο βασικά κατηγορήματα: το κατηγόρημα *attacker(M)*, με τη σημασία ότι ο εχθρός *μπορεί* να έχει στην κατοχή του τον όρο M και το κατηγόρημα *message(M,N)* με την έννοια ότι ο όρος M *μπορεί* να εμφανιστεί στον όρο-κανάλι N. Ακόμα, είναι δυνατόν να έχουμε μια ανισότητα της μορφής $\forall x.M \neq N$, όπου x μεταβλητή και M, N όροι. Στα επόμενα θα επικεντρωθούμε στο πρώτο κατηγόρημα.
Η διαδικασία που αναφέρουμε παρακάτω προϋποθέτει ότι οι κανόνες μεταγραφής για τους destructors αποτελούνται από μόνο μία περίπτωση, δηλαδή κάθε destructor έχει το πολύ έναν κανόνα μεταγραφής. Η διαδικασία της μετάφρασης απλοποιείται έτσι αρκετά. Σημειώνουμε ότι αν και περιοριστικό, κάτι τέτοιο ισχύει και στο παράδειγμα παραπάνω αλλά και στο πρωτόκολλο που αναλύουμε στο κεφάλαιο 4. Η ProVerif παράγει προτάσεις που αφορούν τις δυνατότητες του εχθρού και προτάσεις που αφορούν το πρωτόκολλο.
Όσον αφορά τις δυνατότητες του εχθρού, παράγονται καταρχάς προτάσεις που σχετίζονται με την αρχική του γνώση: για κάθε ελεύθερο όνομα a, που δεν έχει δηλωθεί ως private, έχουμε το γεγονός attacker(a). Ακόμα, αν n όνομα που δεν εμφανίζεται πουθενά στο script τότε έχουμε το γεγονός attacker(n). Μοντελοποιούμε έτσι τη δυνατότητα του εχθρού να δημιουργεί νέα ονόματα όπως κλειδιά και nonces. Ο εχθρός μπορεί έπειτα να χρησιμοποιήσει κάθε σύμβολο συνάρτησης και κάθε destructor, σύμφωνα φυσικά με τον εκάστοτε κανόνα μεταγραφής. Αυτά μοντελοποιούνται με τις επόμενες προτάσεις: για κάθε σύμβολο συνάρτησης f n-θέσεων δημιουργείται η πρόταση $attacker(x_1) \wedge attacker(x_2) \wedge ... \wedge attacker(x_n) \rightarrow attacker(f(x_1, x_2, .., x_n))$, όπου $x_1, x_2, ..., x_n$ μεταβλητές (που δεν εμφανίζονται στο script). Επίσης, για κάθε destructor g k-θέσεων για τον οποίο υπάρχει ο κανόνας μεταγραφής $g(M_1,M_2,...,M_k) \rightarrow M$, έχουμε την πρόταση $attacker(M_1) \wedge attacker(M_2) \wedge ... \wedge attacker(M_k) \rightarrow attacker(M)$. Σημειώνουμε

ότι υπάρχουν κάποιοι built-in destructors που αφορούν τα tuples και έτσι έχουμε κατάλληλες προτάσεις της προηγούμενης μορφής και για αυτούς.
Προχωρούμε τώρα στις προτάσεις που αφορούν το πρωτόκολλο. Υποθέτουμε ότι γίνεται input και output μόνο σε δημόσια κανάλια - ελεύθερα ονόματα δηλαδή που δεν έχουν δηλωθεί με private. Η περίπτωση στην οποία χρησιμοποιούνται ιδιωτικά ονόματα (είτε δηλωμένα με *private free* είτε αυτά που δημιουργούνται με το *new*) είναι αντίστοιχη αλλά γίνεται χρήση του κατηγορήματος *message()*. Σε γενικές γραμμές, έχουμε μία πρόταση για κάθε εντολή *out(N,M)* του πρωτοκόλλου. Το συμπέρασμα μιας τέτοιας πρότασης είναι *attacker(M')*, όπου M' όρος χωρίς μεταβλητές και η υπόθεση της πρότασης είναι μια σύζευξη της μορφής $attacker(M_1) \wedge attacker(M_2) \wedge ... \wedge attacker(M_k)$, όπου k το πλήθος των εντολών *in(N,x)* που προηγούνται της παραπάνω εντολής out(N,M) (πιθανώς k=0). Το ποιοί ακριβώς είναι οι όροι M', M_1, M_2, ..., M_k καθορίζεται από τις εντολές if-then-else και let-in-else που προηγούνται της εντολής out. Προκειμένου να καθοριστούν αυτοί οι όροι, κατά το parsing του πρωτοκόλλου από τη διαδικασία της μετάφρασης, παράγονται και αποθηκεύονται κάποιες αντικαταστάσεις οι οποίες αργότερα εφαρμόζονται στις προτάσεις Horn. Ας πάρουμε για παράδειγμα τις ακόλουθες εντολές από το παράδειγμα της ενότητας 3.2:

```
in(channel,mess);
let dec_msg = dec(mess,skeyB) in
let ses_key = ver(dec_msg,pk(skeyA)) in
out(channel,symenc(ses_key,my_secret))
```

Έχουμε μία εντολή out της οποίας προηγείται μοναδική εντολή in. Με βάση τα παραπάνω η πρόταση Horn που θα δημιουργηθεί θα έχει τη μορφή $attacker(M_1) \rightarrow attacker(M')$ (παρατηρούμε ότι στο παράδειγμα, το channel είναι δημόσιο κανάλι αφού έχει δηλωθεί με *free*). Η εντολή out έπεται δύο εντολών let: οι εκφράσεις στις εντολές αυτές πρέπει να επιτύχουν προκειμένου να εκτελεστεί η εντολή εξόδου. Προκειμένου να επιτύχει η πρώτη εντολή θα πρέπει ο όρος που αντικαθίσταται στη μεταβλητή mess να είναι της μορφής *enc(pk(skeyB), y)*, όπου *y* μεταβλητή. Προκειμένου να επιτύχει η δεύτερη εντολή let, θα πρέπει η μεταβλητή *y* (σημειωτέον: $dec(enc(pk(skeyB),y),skeyB) \downarrow y$) να είναι της μορφής *sign(skeyA, x)*, όπου x μεταβλητή. Άρα τελικά η μεταβλητή *mess*, που περιέχει το μήνυμα που λαμβάνει η διεργασία ως είσοδο στην αρχική εντολή in, θα πρέπει να είναι της μορφής *enc(pk(skeyB), sign(skeyA,x))* και η μεταβλητή *ses_key* θα πρέπει να είναι η μεταβλητή x αφού $ver(dec(enc(pk(skeyB), sign(skeyA,x)),skeyB),pk(skeyA)) \downarrow x$. Άρα θα έχουμε για τους όρους M1 και M' ότι $M1 := enc(pk(skeyB), sign(skeyA,x))$ και $M := x$. Πράγματι, αυτός ο κανόνας παράγεται από την ProVerif όπως θα δούμε παρακάτω. Η διαδικασία της μετάφρασης είναι φυσικά πλήρως αυτοματοποιημένη και περιγράφεται λεπτομερώς στο [29].
Σχετικά με το παράδειγμα της ενότητας 3.2, θέσαμε το ερώτημα *attacker(my_secret)* περί της μυστικότητας του ονόματος *my_secret*. Αν η πρόταση αυτή δεν προκύπτει από τις προτάσεις που περιγράψαμε παραπάνω (δηλαδή τις προτάσεις για τον εχθρό και τις

προτάσεις για το πρωτόκολλο), τότε το όνομα *my_ secret* παραμένει μυστικό, δηλαδή ο εχθρός δεν το μαθαίνει ποτέ. Αν η πρόταση αυτή προκύπτει από τις προηγούμενες προτάσεις, τότε η ProVerif παρουσιάζει λεπτομερώς και σταδιακά την επίθεση που βρήκε. Σημειώνουμε ότι η έννοια του κατηγορήματος *attacker(M)* είναι ότι ο εχθρός *μπορεί* να έχει στην κατοχή του τον όρο M. Αυτό υποδεικνύει ότι ακόμα και αν μια τέτοια πρόταση προκύψει από τη διαδικασία ανάλυσης της ProVerif, είναι δυνατόν να συνεχίζει να ισχύει η μυστικότητα του όρου M. Πράγματι, αυτή είναι μία περίπτωση στην οποία η ProVerif επιστρέφει απάντηση στα μέτρα του "Δε γνωρίζω". Ο λόγος που συμβαίνει αυτό, είναι οι προσεγγίσεις που γίνονται κατά τη διαδικασία της μετάφρασης που περιγράψαμε παραπάνω [29]. Η βασικότερη εξ αυτών είναι η δυνατότητα μιας πρότασης να χρησιμοποιηθεί παραπάνω της μίας φοράς, κάτι που σημαίνει ότι οι επαναλήψεις ή μη των εντολών αγνοούνται. Έτσι η προηγούμενη διεργασία έχει την ίδια μετάφραση σε προτάσεις Horn με την εξής διεργασία:

```
!in(ch,mess);
let dec_msg = dec(mess,skeyB) in
let ses_key = ver(dec_msg,pk(skeyA)) in
!out(ch,symenc(ses_key,my_secret))
```

Δίνουμε παρακάτω (πίνακας 3.2) τις προτάσεις Horn στις οποίες μεταφράζεται το παράδειγμα της ενότητας 3.1. Η πρόταση 20 (clause 20) είναι η πρόταση που περιγράψαμε παραπάνω (η ProVerif ονομάζει τη μεταβλητή x που είδαμε παραπάνω, *ses_ key_ 164*). Η έξοδος αυτή παράγεται αυτόματα από την ProVerif με κατάλληλα ορίσματα. Σημειώνουμε ότι είναι δυνατόν η ProVerif να δεχθεί είσοδο σε μορφή προτάσεων Horn και μάλιστα οι αρχικές εκδόσεις της είχαν μόνο αυτή τη δυνατότητα [32].

```
-- Query not attacker:my_secret[]
Initial clauses:
Clause 0: attacker:enc(pk(skey_47),m_48) & attacker:skey_47 -> attacker:m_48
(The attacker applies function dec.)

Clause 1: attacker:symenc(key_53,m_54) & attacker:key_53 -> attacker:m_54
(The attacker applies function symdec.)

Clause 2: attacker:sign(skey_59,m_60) & attacker:pk(skey_59) -> attacker:m_60
(The attacker applies function ver.)

Clause 3: attacker:v_67 & attacker:v_68 -> attacker:enc(v_67,v_68)
(The attacker applies function enc.)

Clause 4: attacker:v_71 & attacker:v_72 -> attacker:symenc(v_71,v_72)
(The attacker applies function symenc.)

Clause 5: attacker:v_75 & attacker:v_76 -> attacker:sign(v_75,v_76)
(The attacker applies function sign.)

Clause 6: attacker:v_78 -> attacker:pk(v_78)
(The attacker applies function pk.)
```

```
Clause 7: attacker:v_81 & attacker:v_82 -> attacker:(v_81,v_82)
(The attacker applies function 2-tuple.)

Clause 8: attacker:(v_89,v_90) -> attacker:v_89
(The attacker applies function 1-proj-2-tuple.)

Clause 9: attacker:(v_92,v_93) -> attacker:v_93
(The attacker applies function 2-proj-2-tuple.)

Clause 10: mess:v_96,v_95 & attacker:v_96 -> attacker:v_95
(The attacker can listen on all channels it has.)

Clause 11: attacker:v_98 & attacker:v_97 -> mess:v_98,v_97
(The attacker can send messages it has on all channels it has.)

Clause 12: attacker:fail-any()
(Initial knowledge of the attacker.)

Clause 13: attacker:ch[]
(Initial knowledge of the attacker.)

Clause 14: equal:v_100,v_100
(Definition of equal.)

Clause 15: attacker:new-name_185
(The attacker can create new names.)
Abbreviations:
new-name_185 = new-name[!att = v_101]

Clause 16: attacker:pk(skeyA_40[])
(The message pk(skeyA_40[]) may be sent to the attacker at output {3}.)

Clause 17: attacker:pk(skeyB_41[])
(The message pk(skeyB_41[]) may be sent to the attacker at output {4}.)

Clause 18: attacker:pkey_131 ->
 attacker:enc(pkey_131,sign(skeyA_40[],session_key_186))
(If the message pkey_131 is received from the attacker at input {5},
then the message enc(pkey_131,sign(skeyA_40[],session_key_186)) may be
 sent to the attacker at output {7}.)
Abbreviations:
session_key_186 = session_key_43[pkey_42 = pkey_131]

Clause 19: attacker:pk(skeyB_41[])
(The message pk(skeyB_41[]) may be sent to the attacker at output {8}.)

Clause 20: attacker:enc(pk(skeyB_41[]),sign(skeyA_40[],ses_key_164)) ->
attacker:symenc(ses_key_164,my_secret[])
(If the message enc(pk(skeyB_41[]),sign(skeyA_40[],ses_key_164)) is
 received from the attacker at input {9},
then the message symenc(ses_key_164,my_secret[]) may be sent to the
 attacker at output {12}.)
```

3.2 Το αποτέλεσμα της μετάφρασης

Από όσα έχουμε δει μέχρι στιγμής, έχουμε ουσιαστικά μετά τη μετάφραση του script, ένα σύνολο από προτάσεις Horn και μία πρόταση στόχο (goal clause) που προσπαθούμε να βρούμε αν έπεται από τις υπόλοιπες. Αυτό ακριβώς το πρόβλημα επιλύει η Prolog. Δυστυχώς αν προσπαθούσαμε να τη χρησιμοποιήσουμε στην περίπτωση μας, θα είχαμε ένα πρόγραμμα που συχνά δεν θα τερμάτιζε χωρίς να κάνει κάποια ουσιαστική πρόοδο. Αυτό οφείλεται στον αλγόριθμο ανάλυσης που εφαρμόζει η Prolog. Ας εξετάσουμε τις προτάσεις 0 και 18 παραπάνω (αλλάζουμε τα ονόματα των μεταβλητών για λόγους αναγνωσιμότητας):

attacker(enc(pk(skey),m)) ∧ attacker(skey) → attacker(m)

attacker(pkey) → attacker(enc(pkey,sign(skeyA,session_key)))

οι οποίες γράφονται ως εξής, προκειμένου να εφαρμοστεί ο κλασικός κανόνας ανάλυσης [31]:

¬attacker(enc(pk(skey),m)) ∨ ¬attacker(skey) ∨ attacker(m)

¬attacker(pkey) ∨ attacker(enc(pkey,sign(skeyA, session_key)))

Έτσι μπορεί να προκύψει από αυτές μέσω ανάλυσης στο *attacker(pkey)* η πρόταση:

¬attacker(enc(pk(skey),pkey)) ∨ ¬attacker(skey) ∨
∨ attacker(enc(pkey,sign(skeyA,session_key)))

μέσω της ενοποίησης των μεταβλητών m και pkey, από όπου με ένα ακόμη βήμα ανάλυσης με την πρόταση 0 έχουμε:

¬attacker(enc(pk(skey),enc(pk(skey),pkey))) ∨ ¬attacker(skey)∨
∨ attacker(enc(pkey,sign(skeyA, session_key)))

μέσω της ενοποίησης *{enc(pk(skey),pkey)/m}*. Τέτοιες ενοποιήσεις και αναλύσεις θα μπορούσαν να συνεχίζονται επ' άπειρον χωρίς κάποιο ουσιαστικό αποτέλεσμα. Έτσι, στο [32] δίνεται ένας νέος αλγόριθμος που αποφεύγει αυτό το πρόβλημα στις περισσότερες περιπτώσεις. Ο αλγόριθμος βασίζεται στον εξής κανόνα για το resolution: έστω προτάσεις $F_1 \wedge F_2 \wedge \ldots \wedge F_n \rightarrow C$ και $F_1' \wedge F_2' \wedge \ldots \wedge F_m' \rightarrow C'$ όπου C και F_1' ενοποιούνται με την αντικατάσταση σ. Τότε έχουμε ως προϊόν ανάλυσης στο F_1' την πρόταση: $(F_1 \wedge F_2 \wedge \ldots \wedge F_n \wedge F_2' \wedge \ldots \wedge F_m')\sigma \rightarrow C'\sigma$. Η χρήση του κανόνα αυτού για την ανάλυση δεν είναι αρκετή προκειμένου να επιτύχουμε τερματισμό σε μια ευρεία κατηγορία πρωτοκόλλων. Έτσι, η ανάλυση καθοδηγείται από έναν αλγόριθμο επιλογής που υποδεικνύει τις προτάσεις που πρέπει να αναλυθούν. Με αναφορά στις προηγούμενες προτάσεις, έχουμε ότι το γεγονός F_1' δεν πρέπει να είναι της μορφής *attacker(x)* όπου x

οποιαδήποτε μεταβλητή. Ακόμα, τα $\{F_i\}$ πρέπει να είναι της μορφής *attacker(x_i)*, όπου $\{x_i\}$ μεταβλητές. Όταν ισχύει αυτό, τότε συνήθως το συμπέρασμα C δεν είναι της μορφής *attacker(x)* και σε συνδυασμό με τη μορφή του F_1' παίρνουμε μια μη τετριμμένη αντικατάσταση ως ενοποιητή. Με αυτές τις προϋποθέσεις παρατηρούμε ότι η προβληματική ανάλυση που έγινε στο παραπάνω παράδειγμα δεν είναι έγκυρη, αφού το *attacker(pkey)* είναι ακριβώς της μορφής *attacker(x)* με x μεταβλητή και εμφανίζεται στις υποθέσεις της πρότασης.

Ξεκινώντας από το αρχικό σύνολο προτάσεων που παρήγαγε η διαδικασία της μετάφρασης, όπως είδαμε παραπάνω, εφαρμόζουμε τον προηγούμενο κανόνα ανάλυσης (με τις προϋποθέσεις που αναφέραμε) έως ότου να μην παράγονται νέες προτάσεις. Κατά την εφαρμογή του κανόνα, γίνονται και κάποιες απλές βελτιστοποιήσεις, όπως η εξάλειψη των επαναλαμβανόμενων υποθέσεων (από δύο ή περισσότερα ίδια κατηγορήματα στην υπόθεση μιας πρότασης κρατούμε μόνο το ένα), η εξάλειψη των ταυτολογιών (όταν δηλαδή το συμπέρασμα μιας πρότασης εμφανίζεται στις υποθέσεις της) και η εξάλειψη προτάσεων που προκύπτουν από άλλες προτάσεις μέσω αντικατάστασης (η πρόταση που διατηρείται είναι γενικότερη αυτής που απαλείφθηκε). Πρόκειται για μια διαδικασία αναζήτησης προς-τα-μπροστά (forward search) για την οποία έχει αποδειχθεί (βλ. [32]) η ιδιότητα της ορθότητας και της πληρότητας. Προφανώς, είναι ακόμη δυνατόν να έχουμε μη τερματισμό. Το σύνολο των πρωτοκόλλων για τα οποία έχουμε πάντα τερματισμό έχουν χαρακτηριστεί στο [29] και ονομάζονται πρωτόκολλα με ετικέτα (tagged protocols). Για τα πρωτόκολλα που δεν είναι tagged μπορούν να εφαρμοστούν διάφορες ευριστικές μέθοδοι (heuristics), μέσω των οποίων επιτυγχάνεται τερματισμός σε περισσότερες περιπτώσεις.

3.4 Μερικές Επεκτάσεις

Θα παρουσιάσουμε στην ενότητα αυτή μερικές επεκτάσεις της ProVerif που αφορούν λειτουργίες τις οποίες χρησιμοποιήσαμε για την ανάλυση του εκλογικού πρωτοκόλλου στο κεφάλαιο 4. Πρόκειται ουσιαστικά για την ενσωμάτωση των γεγονότων και των ιδιοτήτων αντιστοιχίας που είδαμε στην ενότητα 2.4 και της παρατηρήσιμης ισοδυναμίας που είδαμε στην ενότητα 2.3 στην ProVerif.
Αρχικά, η γλώσσα μοντελοποίησης που παρουσιάσαμε στην ενότητα 3.2 επεκτείνεται με τη δυνατότητα χειρισμού γεγονότων, σε πλήρη αντιστοιχία με τον εφαρμοσμένο π-λογισμό. Το γεγονός *MYEVENT* εκτελείται με τον όρο M με την εξής διεργασία: *event MYEVENT(M); Q*. Μετά την εκτέλεση του *MYEVENT* συνεχίζεται η εκτέλεση της διεργασίας Q χωρίς καμία αλλαγή σε αυτή, όπως συμβαίνει στον εφαρμοσμένο π-λογισμό. Η ProVerif υποστηρίζει τη δυνατότητα να δοθούν ερωτήματα σχετικά με τα γεγονότα που έχουν οριστεί στη διεργασία. Με την εντολή *query ev: MYEVENT(e)* η ProVerif προσπαθεί να βρει αν εκτελείται το γεγονός *MYEVENT* στη διεργασία και αν αυτό ισχύει, επιστρέφεται και ο όρος M με τον οποίο εκτελέστηκε το γεγονός. Πιο τυπικά για την εκτέλεση ενός γεγονότος στην ProVerif έχουμε τον εξής ορισμό [29]:

Ορισμός 3.8. Έστω όρος $M \in T$ και γεγονός e. Θα λέμε ότι στο ίχνος Tr εκτελείται το γεγονός e με όρο M όταν το Tr περιλαμβάνει αναγωγή $(E, P \cup \{e(M); R\}) \rightarrow (E, P \cup \{R\})$, για κάποιο περιβάλλον E, πολυσύνολο διεργασιών P και διεργασία R.

Υποθέτοντας ότι υπάρχει και ένα γεγονός *MYEVENT2*, μπορεί κανείς να θέσει το εξής ερώτημα στην ProVerif: *query ev: MYEVENT(e) ==> MYEVENT2(e)*. Πρόκειται για μια ιδιότητα αντιστοιχίας, όπως είδαμε και στον εφαρμοσμένο π-λογισμό: συγκεκριμένα η ProVerif προσπαθεί να αποδείξει τότε ότι για κάθε εκτέλεση του *MYEVENT* με κάποιον όρο M έχει προηγηθεί εκτέλεση του *MYEVENT2* με τον ίδιο όρο M (βλ. ορισμό 2.16). Όμως στην ProVerif λαμβάνεται υπόψιν και κάποιος πιθανός εχθρός που μπορεί να τρέχει παράλληλα με το πρωτόκολλο που εξετάζουμε. Έχουμε έτσι τυπικά τον εξής ορισμό (αντιπρβλ. με τον ορισμό 2.16):

Ορισμός 3.9. Η κλειστή διεργασία P_0 με $fn(P_0) \subseteq N_{pub}$ ικανοποιεί την ιδιότητα αντιστοιχίας $e_1(M) \Rightarrow e_2(N)$ ως προς το N_{pub}, όταν για κάθε N_{pub}-εχθρό Q και για κάθε ίχνος Tr της P με αρχική σημασιολογική κατάσταση $((N_{pub}, N_{priv}), \{P_0, Q\})$, αν στο ίχνος Tr εκτελείται το γεγονός e_1 με όρο $M' \in T_\Sigma$ στο κ-οστό βήμα και υπάρχει αντικατάσταση σ τέτοια ώστε $M\sigma = M'$, τότε στο ίχνος Tr εκτελείται το γεγονός e_2 με όρο $N' = N\sigma$ στο λ-οστό βήμα με $\lambda \leq \kappa$.

Θα χρησιμοποιήσουμε τέτοια ερωτήματα προκειμένου να αποδείξουμε την ορθότητα του εκλογικού πρωτοκόλλου (βλ. κεφάλαιο 4). Μάλιστα θα ζητήσουμε από την ProVerif να αποδείξει ότι για κάθε εκτέλεση του *MYEVENT* υπάρχει μοναδική (διακριτή) εκτέλεση του *MYEVENT2*, αλλάζοντας το παραπάνω ερώτημα σε *query evinj:MYEVENT(e) ==> MYEVENT2(e)*. Πρόκειται για μια ένα-προς-ένα αντιστοιχία, όπως είδαμε στην ενότητα 2.4. Ακόμα θα χρησιμοποιήσουμε και εμφωλευμένες αντιστοιχίες. Για περισσότερες λεπτομέρειες και τους τυπικούς ορισμούς παραπέμπουμε στο [29].
Η διαδικασία της μετάφρασης που παρουσιάστηκε στην ενότητα 3.3 επεκτείνεται κατάλληλα προκειμένου να ληφθούν υπόψιν τα γεγονότα [29]. Πιο συγκεκριμένα, χρησιμοποιείται ακόμη ένα κατηγόρημα *event()*, με τη σημασία ότι κάποιο γεγονός *μπορεί* να εκτελεστεί με κάποιον όρο καθώς και το κατηγόρημα *m-event()*, με τη σημασία ότι κάποιο γεγονός *πρέπει* να εκτελεστεί. Κατά τη διαδικασία της μετάφρασης σχηματίζονται προτάσεις της μορφής $attacker(x_1) \wedge attacker(x_2) \wedge ... \wedge attacker(x_n) \rightarrow event(ev(M))$, όταν μια διεργασία εκτελεί το γεγονός *ev* με τον όρο *M*, αφού λάβει μηνύματα x_1, x_2, ..., x_n σε κάποιο (δημόσιο) κανάλι επικοινωνίας. Επιπλέον, δημιουργούνται προτάσεις της μορφής $attacker(x_1) \wedge attacker(x_2) \wedge ... \wedge attacker(x_n) \wedge m\text{-}event(ev(M)) \rightarrow attacker(p)$, όταν μια διεργασία κάνει output σε (δημόσιο) κανάλι τον όρο *p*, αφού λάβει μηνύματα x_1, x_2, ..., x_n και εκτελέσει το γεγονός *ev* με τον όρο *M*.
Θα επεκτείνουμε το προηγούμενο παράδειγμα πρωτοκόλλου με γεγονότα, προσπαθώντας να αποδείξουμε ότι αν η διεργασία B λάβει ένα κλειδί συνόδου, τότε αυτό το κλειδί συνόδου έχει δημιουργηθεί από τη διεργασία A. Αυτό δεν αποδεικνύει ότι το κλειδί είναι μυστικό (κάτι που δεν ισχύει όπως είδαμε στην ενότητα 3.2), αλλά αποκλείει την πιθανότητα να έχουμε "παραπλανηθεί" από τον εχθρό με κάποιο ψεύτικο κλειδί. Η ProVerif αποδεικνύει ότι η ιδιότητα αυτή ισχύει. Ο επεκτεταμένος κώδικας δίνεται στον πίνακα 3.3.

```
free ch.
private free my_secret.

fun sign/2.
fun enc/2.
fun pk/1.
fun symenc/2.

reduc dec(enc(pk(skey),m),skey) = m.
reduc ver(sign(skey,m),pk(skey)) = m.
reduc symdec(symenc(key,m),key) = m.

query ev: RCVD(x)==> ev: NEWKEY(x).


let A =
     in(ch,pkey);
     new session_key;
     event NEWKEY(session_key);
     out(ch,enc(pkey,sign(skeyA,session_key)))
     .
```

```
let B =
      out(ch,pk(skeyB));
      in(ch,mess);
      let dec_msg = dec(mess,skeyB) in
      let ses_key = ver(dec_msg,pk(skeyA)) in
      out(ch,symenc(ses_key,my_secret));
      event RCVD(ses_key)
      .


process
      new skeyA;
      new skeyB;
      out(ch,pk(skeyA));
      out(ch,pk(skeyB));

      A | B
```

3.3 Ο κώδικας του προηγούμενου παραδείγματος επεκτεταμένος με γεγονότα

Εκτός από το χειρισμό γεγονότων, ενδιαφερόμαστε για αποδείξεις παρατηρήσιμης ισοδυναμίας μεταξύ δύο διεργασιών. Πράγματι, κάποιες σημαντικές ιδιότητες των εκλογικών πρωτοκόλλων (πχ μυστικότητα της ψήφου) μπορούν να εκφραστούν μόνο μέσω της παρατηρήσιμης ισοδυναμίας, όπως θα δούμε αναλυτικά στο κεφάλαιο 4. Δυστυχώς, όπως προαναφέραμε το πρόβλημα απόφασης για την παρατηρήσιμη ισοδυναμία είναι μη αποκρίσιμο. Η ProVerif μπορεί να αποδείξει παρ' όλα αυτά την ισοδυναμία σε κάποιες συγκεκριμένες περιπτώσεις. Από αυτές, η γενικότερη περίπτωση και αυτή που θα χρησιμοποιήσουμε παρακάτω, είναι η περίπτωση στην οποία δύο διεργασίες διαφέρουν μόνο ως προς τους όρους που χρησιμοποιούν [29].
Η έννοια της διεργασίας επεκτείνεται σε αυτό το πλαίσιο στην έννοια της δι-διεργασίας (biprocess). Τα biprocesses ακολουθούν τη γραμματική των κανονικών διεργασιών, μπορούν όμως να περιέχουν όρους της μορφής *diff[a,b]*, το οποίο γράφεται επίσης ισοδύναμα και ως *choice[a,b]*. Αυτό που συμβαίνει εδώ είναι ότι μια δι-διεργασία P περιλαμβάνει στην πραγματικότητα δύο "κανονικές" διεργασίες: η διεργασία *fst(P)* είναι η διεργασία που προκύπτει από την δι-διεργασία P αν θέσουμε το a σε όλες τις εμφανίσεις του *diff[a,b]* (ή του *choice[a,b]*). Η διεργασία *snd(P)* προκύπτει αντίστοιχα αν αντικαταστήσουμε τις εμφανίσεις του *diff[a,b]* με b. Σημειώνουμε ότι σε ένα biprocess οι εμφανίσεις του *diff* μπορούν να περιέχουν διαφορετικούς όρους (πχ *diff[a,b]* και *diff[hash(c),hash(d)]*). Η *fst(P)* θα είναι πάντα η διεργασία στην οποία αντικαθίσταται κάθε εμφάνιση του diff με τον πρώτο από τους δύο όρους και αντίστοιχα για την *snd(P)*. Η σημασιολογία μιας δι-διεργασίας είναι η ίδια με αυτή των κανονικών διεργασιών με τις εξής διαφορές:

- μπορεί να γίνει output σε κανάλι μόνο αν επιτυγχάνει και στις δύο διεργασίες, δηλαδή με αναφορά στον ορισμό 3.5: *(E, P U {out(N,M);Q, in(N',x);R}) → (E, P U {Q, R{M/x}})* μόνο αν *fst(N) = fst(N')* και *snd(N) = snd(N')*, όπου θεωρούμε ότι οι όροι N και N' μπορούν να περιέχουν υπο-όρους της μορφής

diff[a,b] και *fst(N)* είναι ο όρος που προκύπτει αν αντικαταστήσουμε τις εμφανίσεις του *diff[a,b]* με a. Αντίστοιχα για το *snd(N)*.

- μια διεργασία της μορφής *let x=D in Q else R*, ανάγεται στην Q σε μια δι-διεργασία P μόνο αν η έκφραση D επιτυγχάνει και στην *fst(P)* και στην *snd(P)*. Δηλαδή, με αναφορά στον ορισμό 15: *(E, P U {let x=D in Q else R}) → (E, P U {Q{diff[M,M']/x}})*, όπου *fst(D) ↓ M* και *snd(D) ↓ M'*, όπου και πάλι θεωρούμε ότι μια έκφραση D μπορεί να περιλαμβάνει όρους *diff[a,b]* και *fst(D)* είναι η έκφραση που προκύπτει αν αντικαταστήσουμε τις εμφανίσεις του *diff[a,b]* με a. Ομοίως για *snd(D)*.

- μια διεργασία της μορφής *let x=D in Q else R*, ανάγεται στην R σε μια δι-διεργασία P μόνο αν η έκφραση D αποτυγχάνει και στην *fst(P)* και στην *snd(P)*. Δηλαδή, με αναφορά στον ορισμό 15: *(E, P ∪ {let x=D in Q else R}) → (E, P ∪ {R})*, όπου *fst(D) ↓ fail* και *snd(D) ↓ fail*.

Αν μια δι-διεργασία P δεν μπορεί να ακολουθήσει τις παραπάνω περιπτώσεις (πχ *fst(D) ↓ M* αλλά *snd(D) ↓ fail*) τότε λέμε ότι αποκλίνει. Όταν για μια δι-διεργασία P δεν υπάρχει σημασιολογική κατάσταση *(E',Q)* έτσι που *(E, P) → (E', Q)* και η Q αποκλίνει, λέμε ότι η P ικανοποιεί ισοδυναμία diff (diff-equivalence). Αποδεικνύεται [29] ότι αν η δι-διεργασία P ικανοποιεί diff-equivalence τότε *fst(P) ≈ snd(P)* (ορισμός 2.13), δηλαδή οι δύο κανονικές διεργασίες που εμπεριέχονται σε αυτή, είναι παρατηρήσιμα ισοδύναμες. Αυτή την ιδιότητα θα εκμεταλευτούμε στα επόμενα, γράφοντας τις διεργασίες A και B σε κατάλληλη μορφή μέσω μιας δι-διεργασίας (θα πρέπει δηλαδή *fst(P) = A* και *snd(P) = B* για κατάλληλη δι-διεργασία P), η οποία εφόσον αποδειχθεί από την ProVerif ότι ικανοποιεί diff-equivalence, θα οδηγεί στο συμπέρασμα $A \approx B$, το οποίο θα είναι και το ζητούμενο.
Όπως και στην περίπτωση των γεγονότων παραπάνω, ο αλγόριθμος μετάφρασης της ενότητας 3.3 επεκτείνεται προκειμένου να αποδειχτεί η ιδιότητα diff-equivalence για ένα biprocess *P*. Τα συνήθη κατηγορήματα *attacker(p)* και *message(M, N)* που είδαμε στην ενότητα 3.3, μετατρέπονται στα *attacker(p_1, p_2)* και *message(M_1, N_1, M_2, N_2)* αντίστοιχα [29]. Το κατηγόρημα *attacker(p_1, p_2)* έχει την έννοια ότι ο εχθρός μπορεί να έχει στην κατοχή του τον όρο p_1 από τη διεργασία *fst(P)* και τον όρο p_2 από τη διεργασία *snd(P)*. Παρόμοια, για το κατηγόρημα *message(M_1, N_1, M_2, N_2)* έχουμε ότι το μήνυμα N_1 μπορεί να εμφανιστεί στο κανάλι M_1 από τη διεργασία *fst(P)* και το μήνυμα N_2 μπορεί να εμφανιστεί στο κανάλι M_2 από τη διεργασία *snd(P)*. Τα κατηγορήματα αυτά χρησιμοποιούνται στη θέση των "απλών" *attacker(p)* και *message(M, N)* από τον αλγόριθμο μετάφρασης.

Η τελευταία επέκταση που θα δούμε, είναι η χρήση των εντολών συγχρονισμού *sync*. Η ProVerif υποστηρίζει τη διεργασία *sync n; P*, όπου *P* κάποια διεργασία, *n* φυσικός αριθμός και η εντολή *sync n;* δεν βρίσκεται υπο αντιγραφή. Πρόκειται για μια εντολή

συγχρονισμού εν είδει φράγματος (barrier), η οποία έχει την εξής σημασία: η εκτέλεση μπλοκάρει στη διεργασία P μέχρις ότου εκτελεστούν όλες οι εντολές *sync k* για $k \leq n$ σε όλες τις διεργασίες που τρέχουν παράλληλα με τη διεργασία *sync n; P.* Τυπικά πρέπει να επεκτείνουμε τους ορισμούς 3.4 και 3.5. Η νέα σημασιολογική κατάσταση θα είναι τώρα μια τριάδα που εκτός από τα E και P που είδαμε στον ορισμό 3.4 θα περιέχει και ένα πολυσύνολο B. Το B περιλαμβάνει τους φυσικούς αριθμούς που εμφανίζονται μετά την εντολή *sync* για κάθε τέτοια εντολή σε μια διεργασία και κατ' επέκταση σε ένα πολυσύνολο διεργασιών όπως το P του ορισμού 3.4. Πιο συγκεκριμένα έχουμε:

Ορισμός 3.10. Έστω P=$\{P_1, P_2, ..., P_n\}$ πεπερασμένο πολυσύνολο διεργασιών όπως στον ορισμό 3.4. Ορίζουμε το πολυσύνολο B ως B:=barriers(P_1) ∪ barriers(P_2) ∪...∪ barriers(P_n), όπου οι συναρτήσεις barriers ορίζονται επαγωγικά στο σύνολο των διεργασιών και επιστρέφουν ένα πολυσύνολο φυσικών αριθμών ως εξής:

- barriers(0)=∅
- barriers(out(N,M); P) = barriers(P)
- barriers(in(N,x); P) = barriers(P)
- barriers(P | Q) = barriers(P) ∪ barriers(Q)
- barriers(!P)=∅ (δεν μπορούν να υπάρχουν εντολές sync υπο αντιγραφή)
- barriers(new a; P)=barriers(P)
- barriers(let x=D in P else Q)=barriers(P) ∪ barriers(Q)
- barriers(if D then P else Q)=barriers(P) ∪ barriers(Q)
- barriers(event MYEVENT(M); Q)=barriers(Q)
- barriers(sync t; P)={t} ∪ barriers(P)

Ο ορισμός 3.5 μπορεί τώρα να επεκταθεί με την εξής ιδιότητα που πρέπει να ισχύει για την σχέση αναγωγής →:

(B, E, P ∪ {sync t; P_1, sync t; P_2, ..., sync t; P_n}) → (B\{t^n}, E, P ∪ {P_1, P_2, ..., P_n}), όπου B είναι το πολυσύνολο που ορίστηκε στον ορισμό 3.10, n≥1, t^n δηλώνει την επανάληψη του t n φορές και για κάθε t'≤t έχουμε ότι t'∉B\{t^n}.

Οι υπόλοιπες ιδιότητες παραμένουν ίδιες, όπως παρουσιάστηκαν στον ορισμό 3.5, με τη διαφορά ότι αναφέρονται πάντα στην τριάδα (B, E, P) (το σύνολο B παραμένει όμως ίδιο στις άλλες ιδιότητες). Τέλος, στην αρχική σημασιολογική κατάσταση για μια διεργασία R έχουμε *B := barriers(R).*

Η χρήση των εντολών συγχρονισμού μπορεί να γίνει, επειδή έτσι επιβάλλεται από την περιγραφή ενός πρωτοκόλλου. Ανεξάρτητα από αυτό, μπορούν να χρησιμοποιηθούν για να αποδειχθεί μια παρατηρήσιμη ισοδυναμία. Στο [33], η ProVerif επεκτείνεται ώστε να έχει τη δυνατότητα να κάνει εναλλαγές δεδομένων (swapping) στα σημεία συγχρονισμού. Αυτές οι εναλλαγές δεδομένων βοηθούν στην απόδειξη ισοδυναμιών που διαφορετικά δεν θα μπορούσαν να αποδειχθούν με την ProVerif (ενώ ισχύουν). Η επέκταση αυτή είναι ορθή, δηλαδή αν μετά από κάποια εναλλαγή ισχύει η ισοδυναμία, τότε η ισοδυναμία ισχύει

και στην αρχική διεργασία. Για περισσότερες λεπτομέρειες σχετικά με την εντολή sync και την εναλλαγή δεδομένων βλ. [33].

Κεφάλαιο 4

4.1 Πρωτόκολλα Ηλεκτρονικής Ψηφοφορίας

Με τον όρο ηλεκτρονική ψηφοφορία (electronic voting ή e-voting) εννοούμε την ψηφοφορία εκείνη στην οποία χρησιμοποιούνται ηλεκτρονικά μέσα για την κατάθεση ή την καταμέτρηση των ψήφων. Τα ηλεκτρονικά μέσα που χρησιμοποιούνται ποικίλουν. Τα Μηχανήματα Ηλεκτρονικής Ψηφοφορίας (Electronic Voting System ή EVM) είναι αρκετά διαδεδομένα σε χώρες όπως η Ινδία και η Βραζιλία και η χρήση τους μειώνει το κόστος διενέργειας των εκλογών και το χρόνο που απαιτείται για να εκδοθεί το αποτέλεσμα [34]. Στην παρούσα εργασία ενδιαφερόμαστε για ηλεκτρονικές ψηφοφορίες μέσω του διαδικτύου, με χρήση ηλεκτρονικών υπολογιστών. Τέτοιου τύπου διαδικασίες δεν είναι ιδιαίτερα διαδεδομένες σε επίπεδο εθνικών εκλογών και μάλιστα ομάδες εργασίας σε διάφορες χώρες [3] προτείνουν να μη χρησιμοποιηθούν επί του παρόντος, αφού οι κίνδυνοι είναι περισσότεροι από τα οφέλη. Οι κίνδυνοι πηγάζουν από τα πιθανά κενά ασφαλείας που μπορεί να υπάρχουν στη διαδικασία και να επηρεάσουν το τελικό αποτέλεσμα καθώς και από τη μη αποδοχή της εγκυρότητας της διαδικασίας από τους ψηφοφόρους. Παρ' όλα αυτά, στο πλαίσιο πιο περιορισμένων εκλογικών διαδικασιών, πρωτόκολλα e-voting μέσω του διαδικτύου έχουν χρησιμοποιηθεί με επιτυχία [2].
Προκειμένου να ενισχυθεί τόσο η ασφάλεια της διαδικασίας της ψηφοφορίας όσο και η εμπιστοσύνη των ψηφοφόρων σε αυτή, ένα πρωτόκολλο e-voting πρέπει να ικανοποιεί πολλές ιδιότητες ασφαλείας. Αναφέρουμε παρακάτω τις βασικότερες από αυτές τις ιδιότητες (βλ. [4]). Στην επόμενη ενότητα θα ορίσουμε με σαφήνεια τις ιδιότητες που θα αναλύσουμε για το πρωτόκολλο που εξετάζουμε.

- Ορθότητα (soundness): το αποτέλεσμα της ψηφοφορίας πρέπει να αντιστοιχεί στο άθροισμα των εγκύρων ψήφων. Η ορθότητα ενός πρωτοκόλλου e-voting απαιτεί τρεις διαφορετικές ιδιότητες: δικαιωματικότητα (eligibility), που σημαίνει ότι μόνο οι ψηφοφόροι που έχουν το δικαίωμα μπορούν να συμμετέχουν στη διαδικασία, μη μεταβλητότητα (inalterability), δηλαδή κανείς δεν μπορεί να αλλάξει την ψήφο κάποιου ψηφοφόρου και μη επαναχρησιμοποίηση (non-reusability) που σημαίνει ότι οι ψηφοφόροι μπορούν να ψηφίσουν το πολύ μία φορά [35].

- Δυνατότητα επιβεβαίωσης (verifiability): το πρωτόκολλο e-voting πρέπει να επιτρέπει στους ψηφοφόρους (και πιθανώς σε τρίτους) να ελέγχουν την ορθότητα της διαδικασίας. Αυτό απαιτεί να δίνονται οι εξής δυνατότητες: επιβεβαίωση για τον ψηφοφόρο (individual verifiability), δηλαδή ο κάθε ψηφοφόρος πρέπει να μπορεί να ελέγξει ότι η ψήφος του καταγράφτηκε, καθολική επιβεβαίωση (universal verifiability), δηλαδή οποιοσδήποτε πρέπει να μπορεί να ελέγξει ότι το

αποτέλεσμα της διαδικασίας αντιστοιχεί στις ψήφους που έχουν καταγραφεί και επιβεβαίωση της δικαιωματικότητας (eligibility verifiability), που σημαίνει ότι οποιοσδήποτε πρέπει να είναι σε θέση να ελέγξει ότι οι ψήφοι που καταμετρήθηκαν είναι έγκυρες (προέρχονται δηλαδή από ψηφοφόρους με δικαίωμα ψήφου) και κάθε ψηφοφόρος (που έχει δικαίωμα ψήφου) ψήφισε το πολύ μία φορά [36].

- Εμπιστευτικότητα (privacy): η ψήφος είναι μυστική (vote-privacy) και παραμένει μυστική ακόμα και μετά το πέρασμα χρόνων[1] (everlasting vote-privacy [37]). Ακόμα, δεν πρέπει να είναι δυνατόν να δοθεί κάποια απόδειξη για το πώς ψήφισε ο κάθε ψηφοφόρος, για να αποτραπεί η εξαγορά ψήφων (receipt-freeness). Τέλος το πρωτόκολλο πρέπει να παρέχει αντίσταση στον εξαναγκασμό (coercion-resistance), δηλαδή ακόμα και αν κάποιος κακόβουλος (coercer) δίνει εντολές στον ψηφοφόρο, πρέπει να υπάρχει τρόπος για τον ψηφοφόρο να μην τις ακολουθήσει χωρίς να το μάθει ο coercer.

Στις παραπάνω ιδιότητες ασφαλείας μπορούν να προστεθούν και άλλες φυσικά, όπως για παράδειγμα η ανοχή στα σφάλματα (robustness), ώστε κάποια βλάβη (εκούσια ή ακούσια) στην υποδομή να μην επηρεάζει τη διαδικασία της ψηφοφορίας (πχ να μην χάνονται ψήφοι που έχουν ήδη κατατεθεί). Στην επόμενη ενότητα θα ορίσουμε τυπικά τις ιδιότητες της ορθότητας και της μυστικότητας της ψήφου, τις οποίες αναλύσαμε (και αποδείξαμε) με χρήση της ProVerif για το πρωτόκολλο από το [20].

1 Η δεύτερη ιδιότητα διακρίνεται από την πρώτη, διότι με τη χρήση της σύγχρονης κρυπτογραφίας επιτυγχάνονται εγγυήσεις υπολογιστικής ασφάλειας: δοθέντος αρκετού χρόνου η κρυπτογράφηση είναι δυνατόν να "σπάσει". Η αποκάλυψη της ψήφου ενός ψηφοφόρου όμως δεν θα πρέπει να μπορεί να γίνει για ένα μεγάλο χρονικό διάστημα, αφού κάτι τέτοιο μπορεί να οδηγήσει σε διώξεις εναντίον του.

4.2 Πρωτόκολλα e-voting στον Εφαρμοσμένο π-λογισμό

Στην ενότητα αυτή θα δώσουμε τυπικούς ορισμούς για την έννοια του πρωτοκόλλου ψηφοφορίας καθώς και για τις έννοιες της ορθότητας και της μυστικότητας της ψήφου. Θα χρησιμοποιήσουμε τη γλώσσα του Εφαρμοσμένου π-λογισμού και τους ορισμούς που δόθηκαν στο κεφάλαιο 2. Βασικές αναφορές αποτελούν τα [4] και [35].

Ορισμός 4.1. Ένα πρωτόκολλο ψηφοφορίας (voting protocol) είναι ένα tuple (V, A_1, A_2, ..., A_m, **n**), όπου V είναι η διεργασία που εκτελείται από έναν ψηφοφόρο, A_i είναι οι διεργασίες που εκτελούνται από τις υπεύθυνες για τις εκλογές αρχές (election authorities) και **n** ένα σύνολο ονομάτων. Θεωρούμε ότι η διεργασία V περιλαμβάνει κάποιες ελεύθερες μεταβλητές $\{x_i^{id}\}$ και $\{x_j^{vote}\}$, ότι κάποια διεργασία A_i περιλαμβάνει διεργασία N<M>.P όπου N είναι ελεύθερο όνομα, P διεργασία και M όρος και ότι υπάρχει διεργασία A_j (με j όχι απαραίτητα διαφορετικό του i) που περιλαμβάνει κάποια διεργασία vid_1. vid_2. ... vid_n.Q.

Αυτό που προσπαθούμε να μοντελοποιήσουμε στον παραπάνω ορισμό είναι ότι υπάρχει κάποια διεργασία V, η ακριβής μορφή της οποίας εξαρτάται από το εκάστοτε πρωτόκολλο, η οποία ακολουθείται από τους ειλικρινείς ή νόμιμους ψηφοφόρους (honest voters) με διαφορά κάποια συγκεκριμένα αναγνωριστικά στοιχεία που διακρίνουν τους ψηφοφόρους και χρησιμοποιούνται για την αναγνώρισή τους (πχ αριθμός ταυτότητας) καθώς και τα στοιχεία που υποδηλώνουν την ψήφο που επιλέγει ο καθένας (μεταβλητές $\{x_i^{id}\}$ και $\{x_j^{vote}\}$ αντίστοιχα). Ακόμα υπάρχουν κάποιες αρμόδιες αρχές οι οποίες εκτελούν τις διεργασίες A_i, ενώ τουλάχιστον μία εξ αυτών πρέπει να ανακοινώσει το αποτέλεσμα της ψηφοφορίας σε κάποιο δημόσιο κανάλι επικοινωνίας N. Κάποια από τις αρμόδιες αρχές είναι επίσης υπεύθυνη να δημιουργήσει τα αναγνωριστικά στοιχεία που δίνονται στους ψηφοφόρους. Τέλος υπάρχει ένα σύνολο ονομάτων τα οποία θα παίξουν το ρόλο ιδιωτικών καναλιών επικοινωνίας πχ για την απόκτηση κρυφών αναγνωριστικών ή κλειδιών ή άλλων στοιχείων. Καταλήγουμε έτσι στον ακόλουθο ορισμό για τη συνολική διεργασία που μοντελοποιεί το πρωτόκολλο ψηφοφορίας:

Ορισμός 4.2. Μία διεργασία ψηφοφορίας (voting process) ενός πρωτοκόλλου ψηφοφορίας (V, A_1, A_2, ..., A_m, **n**) είναι μια κλειστή απλή διεργασία της μορφής:

$$vn_1.\ vn_2.\ ...\ vn_k.\ (V\sigma_{id1}\sigma_{v1} \mid ... \mid V\sigma_{idn}\sigma_{vn} \mid A_1 \mid ... \mid A_l)$$

όπου $\{n_j\}$ ονόματα του **n**, $\{\sigma_{idi}\}$ αντικαταστάσεις στις μεταβλητές $\{x_i^{id}\}$, $\{\sigma_{vi}\}$ αντικαταστάσεις στις μεταβλητές $\{x_j^{vote}\}$ και $l \leq m$.

Στον παραπάνω ορισμό θεωρούμε ότι δεν μετέχουν όλες οι διεργασίες $\{A_i\}$ διότι κάποιες

από τις αρμόδιες αρχές μπορεί να μην είναι ειλικρινείς (honest authorities). Παρ'όλα αυτά η προσέγγιση που συνήθως ακολουθείται στη βιβλιογραφία της ProVerif (πχ [35]) και την οποία θα ακολουθήσουμε και εδώ είναι ότι υπάρχει μία και μόνο αρμόδια αρχή η οποία είναι ειλικρινής. Αυτό θα οδηγήσει και σε περαιτέρω απλοποιήσεις του πρωτοκόλλου που αναλύουμε. Κατά τα άλλα, όπως έγινε σαφές και παραπάνω, οι αντικαταστάσεις που εμφανίζονται στη διεργασία, μοντελοποιούν τα διακριτά αναγνωριστικά στοιχεία που δίνονται στον κάθε ψηφοφόρο καθώς και την ψήφο που επιλέγει ο καθένας να ψηφίσει. Υπενθυμίζουμε εδώ ότι η ProVerif δεν υποστηρίζει αντικαταστάσεις της προηγούμενης μορφής αλλά η διεργασία *let x=D in P else Q*, παίζει αντίστοιχο ρόλο. Επίσης τα αναγνωριστικά στοιχεία θα δίνονται στον ψηφοφόρο μεσω input από τη διεργασία της αρμόδιας αρχής.

Ορισμός 4.3. Έστω διεργασία ψηφοφορίας όπως στον ορισμό 4.2 και I := {i: 1 ≤ i ≤ n} σύνολο δεικτών. Ορίζουμε τότε το περιβάλλον αποτίμησης που προκύπτει από την ψηφοφορία (voting context) $VP_I[\_]:= vn_1.\ vn_2.\ ...\ vn_k.\ (\ |_{j\notin I}\ V\sigma_{id_j}\sigma_{\upsilon j}\ |\ \_\ |\ A_1\ |\ ...\ |\ A_l)$.

Το περιβάλλον αποτίμησης που ορίσαμε στον προηγούμενο ορισμό αντικαθιστά κάποιους από τους ψηφοφόρους – αυτούς που υποδεικνύει το σύνολο I - με άλλες διεργασίες. Είμαστε έτοιμοι τώρα να δώσουμε τον ορισμό της μυστικότητας ψήφου (vote privacy) για ένα πρωτόκολλο ψηφοφορίας:

Ορισμός 4.4. Έστω πρωτόκολλο ψηφοφορίας (V, A_1, A_2, ..., A_m, **n**). Θα λέμε ότι το πρωτόκολλο παρέχει μυστικότητα της ψήφου αν για κάθε διεργασία ψηφοφορίας και κάθε σύνολο δεικτών I, το περιβάλλον αποτίμησης που προκύπτει από την ψηφοφορία, VP_I, ικανοποιεί την εξής ιδιότητα:

$$VP_I[\ V\sigma_{id1}\sigma_{\upsilon 1}\ |\ V\sigma_{id2}\sigma_{\upsilon 2}\ ] \approx VP_I[\ V\sigma_{id1}\sigma_{\upsilon 2}\ |\ V\sigma_{id2}\sigma_{\upsilon 1}\ ]$$

Ο παραπάνω ορισμός απαιτεί ουσιαστικά ο εχθρός να μην μπορεί να διακρίνει αν κάποιος ψηφοφόρος (συγκεκριμένα αυτός που έχει αναγνωριστικά στοιχεία που δίνονται από την αντικατάσταση σ_{id1}) ψήφισε υ_1 ή υ_2 (θεωρούμε ότι οι αντικαταστάσεις σ_{id1}, $\sigma_{\upsilon 1}$, σ_{id2} και $\sigma_{\upsilon 2}$ δίνουν έγκυρα αναγνωριστικά και έγκυρες ψήφους). Ο ψηφοφόρος αυτός δεν μπορεί να θεωρηθεί μόνος του διότι αν δεν υπάρχει καμία άλλη ψήφος υ_1 ή υ_2 τότε ο εχθρός θα μπορέσει να ξεχωρίσει εκ του αποτελέσματος (που ανακοινώνεται σε δημόσιο κανάλι) τι από τα δύο ψήφισε ο ψηφοφόρος. Συνεπώς η ψήφος του πρέπει να αντισταθμιστεί από κάποιον άλλο ψηφοφόρο. Έτσι ο ορισμός μπορεί να διαβαστεί ως εξής: "Δεδομένου ότι υπάρχει ψηφοφόρος που ψηφίζει υ_2 και υ_1, ο εχθρός δεν πρέπει να μπορεί να ξεχωρίσει αν κάποιος ψηφοφόρος ψήφισε υ_1 στη μία περίπτωση και υ_2 στην άλλη". Θα πρέπει να σημειώσουμε ότι μέσω της ProVerif ελέγχουμε συνήθως την προηγούμενη ιδιότητα όταν υπάρχουν μόνο οι δύο ψηφοφόροι με αναγνωριστικά που δίνονται από τις σ_{id1} και σ_{id2}. Θεωρούμε ότι αυτές οι βασικές περιπτώσεις μπορούν να γενικευτούν και στην περίπτωση οποιουδήποτε αριθμού ψηφοφόρων (βλ. και [4]).

Ακολουθώντας το [35], θα επεκτείνουμε κάποιους από τους προηγούμενους ορισμούς προκειμένου να ορίσουμε την ορθότητα (soundness) ενός πρωτοκόλλου.

Ορισμός 4.5. Ένα πρωτόκολλο ψηφοφορίας (voting protocol) είναι ένα tuple (V, A_1, A_2, ..., A_m, **n**), όπως στον ορισμό 4.1. Επιπλέον, η διεργασία V περιλαμβάνει τα γεγονότα STARTID($(x_i^{id})_i$) και BEGINVOTE($(x_i^{id})_i$,$(x_j^{vote})_j$). Η διεργασία A_i που περιέχει την υποδιεργασία N<M>.P, περιλαμβάνει το γεγονός ENDVOTE(M). Τέλος η διεργασία A_j που περιέχει την υποδιεργασία vid_1. vid_2. ... vid_n.Q περιλαμβάνει το γεγονός NEWID($(id_i)_i$).

Ο παραπάνω ορισμός ουσιαστικά "μαρκάρει" (annotates) τις διεργασίες του πρωτοκόλλου με κάποια γεγονότα, προκείμενου να ορίσουμε την ιδιότητα της ορθότητας μέσω αντιστοιχιών (correspondences). Έτσι, μόλις ένας ψηφοφόρος αποκτήσει τα αναγνωριστικά στοιχεία του, ενεργοποιείται το γεγονός STARTID το οποίο περιέχει αυτά τα στοιχεία. Κατά την έναρξη της διαδικασίας κατάθεσης ψήφου, ενεργοποιείται το γεγονός BEGINVOTE το οποίο περιλαμβάνει τα αναγνωριστικά στοιχεία του ψηφοφόρου μαζί με την ψήφο του. Κατά τη δημιουργία των αναγνωριστικών στοιχείων ενός ψηφοφόρου από την κατάλληλη αρμόδια αρχή, ενεργοποιείται το γεγονός NEWID με τη σημασία ότι μόλις δημιουργήθηκαν τα καινούρια αναγνωριστικά $\{id_i\}$. Τέλος, κατά την καταμέτρηση μιας ψήφου ενεργοποιείται το γεγονός ENDVOTE μαζί με την ψήφο που καταμετρήθηκε.
Μια διεργασία ψηφοφορίας ορίζεται όπως και παραπάνω (βλ. ορισμό 4.2). Μπορούμε να δώσουμε τώρα τον ορισμό της ορθότητας ενός εκλογικού πρωτοκόλλου (βλ. [35]):

Ορισμός 4.6. Ένα πρωτόκολλο ψηφοφορίας (V, A_1, A_2, ..., A_m, **n**) θα λέγεται ορθό (sound) όταν κάθε διεργασία ψηφοφορίας του πρωτοκόλλου ικανοποιεί τα εξής:

1. τα γεγονότα NEWID, STARTID και BEGINVOTE εκτελούνται το πολύ μία φορά με τους ίδιους όρους σε κάθε ίχνος της διεργασίας ψηφοφορίας.
2. Ικανοποιείται η αντιστοιχία: ENDVOTE(x) $\Rightarrow^{inj}$ (BEGINVOTE(y, x) $\Rightarrow^{inj}$ (STARTID(y) $\Rightarrow^{inj}$ NEWID(y))) , όπου x, y μεταβλητές.

Στον ορισμό αυτό το (1) εξασφαλίζει τη σωστή χρήση των γεγονότων: δεν πρέπει για παράδειγμα ένας ψηφοφόρος να εκτελεί το γεγονός STARTID με τα δικά του αναγνωριστικά στοιχεία πάνω από μία φορά. Το (2) εξασφαλίζει τις τρεις διαφορετικές ιδιότητες που σχηματίζουν την ορθότητα ενός εκλογικού πρωτοκόλλου: η δικαιωματικότητα εξασφαλίζεται εφόσον κάθε ψήφος που θα καταμετρηθεί θα προέρχεται από ψηφοφόρο που έχει αναγνωριστικό (και το αναγνωριστικό δίνεται μόνο από την αρμόδια αρχή). Η μη μεταβλητότητα της ψήφου προκύπτει από το ότι κάθε ψήφος που έχει καταμετρηθεί προκύπτει από κάποιον ψηφοφόρο που κατέθεσε ακριβώς αυτή τη ψήφο. Η μη επαναχρησιμοποίηση προκύπτει από το (1) και από το ότι η αντιστοιχία στο (2) είναι ένα-προς-ένα, κατά συνέπεια κάθε ψήφος που κατατίθεται μπορεί να καταμετρηθεί μόνο μία φορά (διαφορετικά το BEGINVOTE θα εκτελείτο με τους ίδιους

όρους τουλάχιστον δύο φορές και δεν θα ίσχυε το (1)). Σημειώνουμε ότι στο [35], ένα πρωτόκολλο ψηφοφορίας μπορεί να περιλαμβάνει και κακόβουλους ψηφοφόρους (corrupted voters), οι οποίοι αφού λάβουν όλα τα απαραίτητα αναγνωριστικά από τις αρμόδιες αρχές, τα στέλνουν στον εχθρό (δηλαδή κάνουν output σε δημόσιο κανάλι). Για λόγους συμβατότητας των ορισμών με το [4] και τον ορισμό 4.4 της μυστικότητας της ψήφου που εμφανίζεται εκεί, επιλέξαμε να μη θεωρήσουμε τέτοιους ψηφοφόρους στην εργασία αυτή και να αποδείξουμε για το πρωτόκολλο που εξετάζουμε, τον ασθενέστερο ορισμό 4.6 που παρουσιάσαμε παραπάνω.

4.3 Το πρωτόκολλο των Pagourtzis et al. και η μοντελοποίησή του

Το πρωτόκολλο που θα μελετήσουμε με τη βοήθεια του εφαρμοσμένου π-λογισμού και της ProVerif προτάθηκε στο [20]. Πρόκειται για ένα πρωτόκολλο που ακολουθεί τις βασικές αρχές του πρωτοκόλλου των Juels, Catalano και Jakobsson [38], επιτυγχάνοντας την ισχυρή ιδιότητα της αντίστασης στον εξαναγκασμό, τη δυνατότητα επιβεβαίωσης από άκρη σε άκρη (end-to-end-verifiability) και συνεχιζόμενη μυστικότητα της ψήφου. Το πρωτόκολλο τρέχει σε γραμμικό χρόνο ως προς το πλήθος των ψηφοφόρων, υποθέτωντας σταθερό πλήθος διπλότυπων ψήφων ανα ψηφοφόρο. Στην παρούσα εργασία, θα προσπαθήσουμε να αποδείξουμε με χρήση της ProVerif και των προηγούμενων ορισμών, την ορθότητα του πρωτοκόλλου και την (απλή) μυστικότητα της ψήφου (ιδιότητα που έπεται από την αντίσταση στον εξαναγκασμό και το everlasting vote privacy). Πρόκειται σαφώς για ένα σύγχρονο πρωτόκολλο που χρησιμοποιεί πληθώρα εξελιγμένων τεχνικών και κρυπτογραφικών primitives. Συνεπώς, αποτελεί ένα καλό πεδίο δοκιμής των δυνατοτήτων της ProVerif. Δυστυχώς, για τους ίδιους λόγους, είμαστε αναγκασμένοι να κάνουμε ορισμένες απλοποιήσεις και μετατροπές σε κάποια σημεία του πρωτοκόλλου, αφού δεν είναι δυνατόν να μοντελοποιηθούν όλα τα στοιχεία του πρωτοκόλλου με ακρίβεια στην ProVerif. Ξεκινάμε με μια παρουσίαση των βασικών βημάτων του πρωτοκόλλου.

Οι βασικές οντότητες είναι οι αρμόδιες αρχές και οι ψηφοφόροι. Υπάρχει επίσης ένα δημόσια προσπέλασιμο bulletin board στο οποίο καταγράφονται διάφορες πληροφορίες για το πρωτόκολλο. Ένας ψηφοφόρος που θέλει να ψηφίσει, αποκτά πρώτα αναγνωριστικά στοιχεία (identification ή id) και διαπιστευτήρια (credentials) από μια διεργασία, μέρος των αρμοδιών αρχών, που ονομάζεται αρχή για την εγγραφή (registration authority). Η αρχή αυτή στέλνει το id και τα credentials στον ψηφοφόρο μέσω ιδιωτικού καναλιού και ταυτόχρονα κρυπτογραφεί τα διαπιστευτήρια και τα δημοσιεύει στο bulletin board. Επίσης δημοσιεύεται στο bulletin board λίστα με τις έγκυρες ψήφους που μπορούν να καταθέσουν οι ψηφοφόροι. Εδώ τελειώνει η λεγόμενη "φάση εγγραφής" (registration phase). Σημειώνουμε ότι στη φάση αυτή, πρέπει να εγγραφούν όλοι οι ψηφοφόροι, αφού όταν ξεκινήσει η επόμενη φάση, δεν υπάρχει πια η δυνατότητα για εγγραφή.

Η φάση της εξουσιοδότησης (authorization phase) ξεκινά μετά την εγγραφή. Εδώ κάθε ψηφοφόρος που συμμετέχει στη διαδικασία, δημοσιεύει στο bulletin board την κρυπτογραφημένη και "τυφλή" (blinded) ψήφο του προκειμένου να αποκτήσει μια ψηφιακή υπογραφή για την ψήφο του [39]. Προκειμένου να υπογραφεί η ψήφος του από την αρμόδια αρχή για την εξουσιοδότηση (authorization authority), πρέπει επίσης να κατατεθεί στο bulletin board το κρυπτογραφημένο διαπιστευτήριο του και το αναγνωριστικό id του. Μαζί με τα παραπάνω, δημοσιεύει μια μη διαδραστική απόδειξη

μηδενικής γνώσης (non-interactive zero knowledge proof, βλ. [40]) που αποδεικνύει την ορθή κρυπτογράφηση του διαπιστευτηρίου. Η αρχή έπειτα ελέγχει αν το διαπιστευτήριο που κατατέθηκε ταιριάζει με αυτό που έχει κατατεθεί στο bulletin board από το registration authority για το ίδιο id. Αυτός ο έλεγχος γίνεται μέσω μιας διαδικασίας που ονομάζεται plaintext equivalence test [41] και επιτρέπει τη σύγκριση κρυπτοκειμένων προκειμένου να διαπιστωθεί αν προέρχονται από το ίδιο plaintext. Αν αυτό πράγματι συμβαίνει (και αφού ελεγχθεί και η ορθότητα της απόδειξης μηδενικής γνώσης), το authorization authority υπογράφει την ψήφο του ψηφοφόρου. Σημειώνουμε ότι λόγω της "τυφλής" υπογραφής, η αρχή αυτή δεν μπορεί να μάθει την ψήφο που υπογράφει (παρ'όλο που μπορεί να αποκρυπτογραφήσει το μήνυμα). Ο ψηφοφόρος έπειτα παίρνει την τυφλή υπογραφή που περιέχει την ψήφο του από το bulletin board, "φανερώνει" την (κρυπτογραφημένη) ψήφο του (unblind) και καταθέτει έτσι μια κρυπτογραφημένη και υπογεγραμμένη από την αρμόδια αρχή ψήφο. Μαζί με την ψήφο, δημοσιεύει και μια απόδειξη μηδενικής γνώσης για την ορθότητα της κρυπτογράφησης της ψήφου καθώς και για το ότι η ψήφος είναι έγκυρη [42] – εντός της λίστας των αποδεκτών ψήφων δηλαδή, που έχει δημοσιευθεί στη φάση της εγγραφής από το registration authority. Εδώ τελειώνει η φάση της εξουσιοδότησης. Όλοι οι ψηφοφόροι πρέπει να έχουν αποκτήσει υπογραφή για την ψήφο τους πριν τελειώσει αυτή η φάση.

Μετά την εξουσιοδότηση ξεκινά η φάση της καταγραφής (tallying phase). Η αρμόδια αρχή για την καταμέτρηση (tallying authority ή tallier) συγκεντρώνει όλες τις έγκυρες ψήφους που υπάρχουν στο bulletin board. Προχωρά σε μια διαδικασία επανακρυπτογράφησης και ανακατέματος των ψήφων (reencrypt και shuffle, βλ. [43]). Αυτό είναι απαραίτητο προκειμένου να μην μπορεί να συνδεθεί κάποια κρυπτογραφημένη ψήφος με την plaintext ψήφο που περιέχει (πιθανώς θα οδηγούσε στην αναγνώριση της ψήφου ενός ψηφοφόρου) και να αποφευχθεί η συσχέτιση της σειράς καταμέτρησης των ψήφων με τη σειρά κατάθεσης τους (και πάλι πιθανώς θα οδηγούσε σε αποκάλυψη της ψήφου κάποιου ψηφοφόρου). Έπειτα ο tallier αποκρυπτογραφεί τις ψήφους που έχουν έγκυρη υπογραφή από το authorization authority και ανακοινώνει το αποτέλεσμα στο Bulletin board. Θεωρούμε ότι αυτό που ανακοινώνεται είναι ζεύγη (επανα-)κρυπτογραφημένων ψήφων και η plaintext ψήφος που περιέχεται. Το τελευταίο αυτό σημείο ίσως να μην είναι ακριβές, αλλά δεν υπάρχει η δυνατότητα καταμέτρησης των ψήφων σε ProVerif και η έκδοση ενός αριθμητικού αποτελέσματος. Πράγματι, στη βιβλιογραφία της ProVerif συνήθως θεωρείται ότι γίνεται output η plaintext ψήφος [35].

Σημειώνουμε ότι στα παραπάνω έχουμε αγνοήσει κάποια στοιχεία του πρωτοκόλλου, τα οποία κατ' ανάγκην θα πρέπει να αγνοήσουμε και στην ProVerif. Αρχικά, το πρωτόκολλο επιτρέπει στις αρμόδιες αρχές να λειτουργούν με κατανεμημένο τρόπο: μπορούν να υπάρχουν N διεργασίες για τις αρμόδιες αρχές σε διαφορετικούς εξυπηρετητές, οι οποίες διαμοιράζονται τα μυστικά στοιχεία που απαιτούνται για τα διάφορα κρυπτοσυστήματα, ενώ απαιτείται συγκεκριμένο πλήθος t αυτών ούτως ώστε να λειτουργήσουν σωστά τα κρυπτοσυστήματα. Αυτή η αρχή λειτουργίας ονομάζεται κρυπτογραφία κατωφλίου (threshold cryptography) [44] και διατηρεί την ασφάλεια του εκάστοτε κρυπτοσυστήματος (πχ του σχήματος κρυπτογράφησης) εφόσον η πλειοψηφία των

διεργασιών δεν ελέγχεται από τον εχθρό. Δυστυχώς, η ProVerif δεν μπορεί να διαχειριστεί τέτοιες περιπτώσεις και συνήθως τα πρωτόκολλα απλοποιούνται στην βασική περίπτωση N = t = 1 [35]. Αυτό έχουμε ακολουθήσει και στη δική μας ανάλυση. Ακόμα, αγνοήσαμε στα παραπάνω κάποιες αποδείξεις μηδενικής γνώσης που δίνονται από τις αρμόδιες αρχές στους ψηφοφόρους. Στην ανάλυση μας θεωρούμε γενικά ότι υπάρχει μόνο μία αρμόδια αρχή. Φυσικά, η αρχή αυτή επιτελεί διάφορους ρόλους (αρχή για την εγγραφή, για την εξουσιοδότηση και την καταμέτρηση όπως είδαμε) αναλόγως τη φάση που βρισκόμαστε στο πρωτόκολλο, αλλά στην πραγματικότητα είναι μία οντότητα. Θεωρούμε λοιπόν ότι η οντότητα αυτή είναι ειλικρινής και ακολουθεί το πρωτόκολλο, συνεπώς δεν υπάρχει η ανάγκη για κάποιες από τις αποδείξεις μηδενικής γνώσεις που προβλέπονται σε αυτό.

Θα προχωρήσουμε τώρα στην πιο λεπτομερή εξέταση του πρωτοκόλλου καθώς και στη μοντελοποίηση του σε εφαρμοσμένο π-λογισμό. Παρατηρούμε ότι το πρωτόκολλο περιλαμβάνει primitives για την κρυπτογράφηση, για ψηφιακές υπογραφές, για τη διαδικασία της επανακρυπτογράφησης και το plaintext equivalence test. Το πρωτόκολλο επίσης περιλαμβάνει αρκετές αποδείξεις μηδενικής γνώσης, των οποίων η μοντελοποίηση σε ProVerif στηρίζεται στο [45]. Θα πούμε περισσότερα για αυτό στη συνέχεια. Αγνοούμε τη διαδικασία του shuffling διότι δεν έχει νόημα στην ProVerif (τα μηνύματα που στέλνονται δεν έχουν κάποια συγκεκριμένη σειρά εδώ, όπως όταν θα ήταν καταγεγραμμένα σε ένα δημόσιο bulletin board). Καταλήγουμε έτσι στην εξής βάση E για τη θεωρία ισότητας $=_{\mathrm{E}}$ (ορισμός 2.3):

```
dec(penc(m,pk(skey),rand),skey) = m
versign(sign(x,skey,valid),pk(skey))=x
unblind(sign(blind(x,r),skey,valid),r)=sign(x,skey,valid)
getvalid(sign(x,skey,valid),skey)=valid
pet(penc(x,pk(y),r1),penc(x,pk(y),r2),y) = true
reencrypt(penc(x,y,r1),r2) = penc(x,y,f(r1,r2))
```

4.1 Η θεωρία ισότητας που χρησιμοποιούμε

όπου η υπογραφή Σ περιέχει τα εξής σύμβολα συναρτήσεων: *pk/1 , penc/3, dec/2, blind/2, sign/3, versign/2, unblind/2, getvalid/2, pet/3, f/2, reencrypt/2*. Εδώ πρέπει να κάνουμε μια βασική παρατήρηση. Το πρωτόκολλο e-voting που εξετάζουμε περιέχει μια νέα παραλλαγή του κρυπτογραφικού primitive των τυφλών υπογραφών η οποία ονομάζεται conditional blind signature και παρουσιάστηκε για πρώτη φορά στο [46]. Με την υπογραφή αυτή, η αρχή εξουσιοδότησης μπορεί ουσιαστικά να "μαρκάρει" μια υπογραφή ως μη έγκυρη, πληροφορία που μπορεί να ανακτηθεί μόνο από κάποια άλλη αρχή. Με αυτόν τον τρόπο είναι δυνατόν να δοθεί μια έγκυρη κατά τα άλλα υπογραφή σε μία ψήφο, αλλά έπειτα στη φάση της καταμέτρησης να αναγνωστεί αυτό το "bit εγκυρότητας" και η ψήφος να μην καταμετρηθεί. Μέσω του primitive αυτού, το πρωτόκολλο επιτυγχάνει coercion resistance. Επιλέξαμε να μοντελοποιήσουμε την υπογραφή αυτή όπως μια κανονική τυφλή υπογραφή [4] αλλά με ένα επιπλέον πεδίο *valid* ορατό μόνο σε κάποιον που έχει στη κατοχή του το ιδιωτικό κλειδί *skey* (δηλαδή κάποια

αρμόδια αρχή). Η μοντελοποίηση αυτή είναι νέα και θεωρούμε ότι καλύπτει επαρκώς το primitive που χρησιμοποιείται στο πρωτόκολλο χωρίς να απομακρύνεται πολύ από την υπάρχουσα βιβλιογραφία. Όσον αφορά τα υπόλοιπα primitives ακολουθήσαμε την βιβλιογραφία της ProVerif [9].
Μπορούμε τώρα να παρουσιάσουμε αναλυτικότερα τις διάφορες διεργασίες που παίρνουν μέρος στο πρωτόκολλο. Αρχικά, όπως αναφέραμε και παραπάνω, θεωρούμε ότι όλες οι αρμόδιες αρχές αποτελούν μία οντότητα, η οποία είναι ειλικρινής και ακολουθεί το πρωτόκολλο. Η οντότητα αυτή εκτελεί τις κατάλληλες διεργασίες για κάθε φάση του πρωτοκόλλου και παίζει αντίστοιχα το ρόλο της αρχής για την εγγραφή, την εξουσιοδότηση και την καταμέτρηση. Η διεργασία για το registration authority παρουσιάζεται στον πίνακα 4.2.

```
registration_authority :=
vid.
idCH<id>.
vcredential.
vrand.
let enc_cred = penc(credential,EA_pkey,rand) in
BB<(enc_cred,id)>.
safeCH<credential>.
internalCH<(id,enc_cred)>.
```

4.2 Η αρχή για την εγγραφή στον εφαρμοσμένο π-λογισμό

Στην παραπάνω διεργασία έχουμε την αρμόδια αρχή για την εγγραφή να δημιουργεί ένα νέο αναγνωριστικό id, το οποίο έπειτα στέλνει μέσω ασφαλούς καναλιού (*idCH*) στον ψηφοφόρο. Έπειτα δημιουργείται νέο διαπιστευτήριο credential, το οποία θα σταλεί τελικά στον ψηφοφόρο μέσω ασφαλούς καναλιού (*safeCH*). Πριν γίνει όμως αυτό το διαπιστευτήριο κρυπτογραφείται (με το δημόσιο κλειδί EA_pkey που ανήκει στις αρμόδιες αρχές) και δημοσιεύεται στο bulletin board (ελεύθερο όνομα *BB*) μαζί με το αναγνωριστικό id στο οποίο αντιστοιχεί. Τέλος, το ζεύγος αναγνωριστικού *id* και κρυπτογραφημένου διαπιστευτηρίου enc_cred, στέλνεται στο ιδιωτικό κανάλι *internalCH* προκειμένου να χρησιμοποιηθεί από τη διεργασία για την εξουσιοδότηση. Σημειώνουμε ότι η αρχή για την εξουσιοδότηση θα μπορούσε να χρησιμοποιήσει το ζεύγος αναγνωριστικού και διαπιστευτηρίου που υπάρχει στο bulletin board, όμως αυτό είναι εύκολο να αλλοιωθεί από τον εχθρό (θα μπορούσε δηλαδή ο εχθρός να στείλει ένα διαφορετικό ζεύγος αφού έχει πρόσβαση στο δημόσιο bulletin board). Στο σημείο αυτό θεωρούμε ότι σε μια υλοποίηση του πρωτοκόλλου, κάποιο μέρος του board θα είναι διαθέσιμο μόνο για ανάγνωση από τρίτους χωρίς δυνατότητα εγγραφής. Διαφορετικά, κάποιος κακόβουλος χρήστης θα μπορούσε απλώς να αλλοιώνει τα μηνύματα των αρμοδίων αρχών. Στην ProVerif παρ'όλα αυτά δεν μπορούμε να έχουμε κανάλια μόνο για ανάγνωση, συνεπώς αν και δημοσιεύονται όλα τα στοιχεία στο bulletin board, στέλνουμε κάποιες πληροφορίες μέσω ιδιωτικών καναλιών για να προσομοιώσουμε μια τέτοια read-only λειτουργία.
Προκειμένου να ακολουθήσουμε τη ροή του πρωτοκόλλου, δίνουμε παρακάτω τη

διεργασία που αντιστοιχεί στον ειλικρινή ψηφοφόρο. Οι δύο πρώτες εντολές αντιστοιχούν στο input του αναγνωριστικού και του διαπιστευτηρίου:

```
honest_voter :=
idCH(id).
safeCH(credential).
    vrand.
    let enc_vote = penc(vote,EA_pkey,rand) in
    vrand1.
    let enc_cred = penc(credential,EA_pkey,rand1) in
    vblind_factor.
    let blind_vote = blind(enc_vote,blind_factor) in
    let zkp = zk(vote,rand,credential,rand1;
enc_vote,a,b, EA_pkey,enc_cred,blind_vote; formula) in
    BB<(id,zkp)>.
    BB(blind_sig).
    if versign(blind_sig,EA_pkey) = blind_vote then
    let sig = unblind(blind_sig,blind_factor) in
    BB<(sig,zkp)>.
```

4.3 Η διεργασία που εκτελεί κάθε ψηφοφόρος

Ακολουθούν έπειτα οι εντολές που σχετίζονται με τη φάση της εξουσιοδότησης. Ο ψηφοφόρος επιλέγει την ψήφο του (ελεύθερη μεταβλητή *vote*) και έπειτα την κρυπτογραφεί. Επίσης κρυπτογραφεί το διαπιστευτήριό του και σχηματίζει την τυφλή ψήφο του χρησιμοποιώντας την τυχαιότητα *blind_factor*. Βλέπουμε έπειτα την απόδειξη μηδενικής γνώσης (*zkp*) που εμφανίζεται στο πρωτόκολλο. Πρόκειται για μια απόδειξη ότι η ψήφος είναι έγκυρη και έχει κρυπτογραφηθεί σωστά καθώς και ότι το διαπιστευτήριο έχει κρυπτογραφηθεί σωστά. Σημειώνουμε ότι έχουμε συμπεριλάβει και τις δύο αποδείξεις μηδενικής γνώσης που προβλέπει το πρωτόκολλο σε μία. Κάτι τέτοιο είναι σύνηθες στη βιβλιογραφία της ProVerif [35] και απλοποιεί τη διαδικασία της ανάλυσης. Κατόπιν, δημοσιεύονται στο bulletin board τα *id* και η απόδειξη μηδενικής γνώσης *zkp,* που περιλαμβάνει το κρυπτογραφημένο διαπιστευτήριο και την τυφλή ψήφο που θα πρέπει να υπογραφεί από την αρχή για την εξουσιοδότηση. Αφού ο ψηφοφόρος κάνει input από το bulletin board την υπογεγραμμένη τυφλή ψήφο του μπορεί πλέον να την φανερώσει (unblind) και να την κατεθέσει προς καταμέτρηση, μαζί με την απόδειξη μηδενικής γνώσης *zkp*, προκειμένου να ελεχθεί από την αρχή για την καταμέτρηση η εγκυρότητα της ψήφου και η ορθή κρυπτογράφησή της. Εξετάζουμε τώρα με περισσότερες λεπτομέρειες τη μοντελοποίηση των αποδείξεων μηδενικής γνώσης στον εφαρμοσμένο π-λογισμό και στην ProVerif.

Ο τρόπος με τον οποίο μοντελοποιούνται οι αποδείξεις μηδενικής γνώσης βασίζεται στο [45]. Συγκεκριμένα, ο "όρος" *zkp* που εμφανίζεται παραπάνω έχει την εξής σημασία: τα *vote*, *rand*, *credential* και *rand1* παραμένουν μυστικά ενώ τα *enc_vote*, *a*, *b*, *EA_pkey*, *enc_cred* και *blind_vote* είναι προσπελάσιμα από οποιονδήποτε. Τα στοιχεία αυτά (ιδιωτικά και δημόσια – διακρίνονται με τη χρήση του ';') σχετίζονται μεταξύ τους με έναν τύπο προτασιακής λογικής, εδώ τον τύπο *formula*, ο οποίος αποτελεί και την

πρόταση (statement) που προσπαθούμε να δείξουμε ότι ισχύει μέσω της απόδειξης μηδενικής γνώσης (το γεγονός ότι πρόκειται για απόδειξη μηδενικής γνώσης δικαιολογείται εφόσον κάποια στοιχεία παραμένουν μυστικά). Ο τύπος *formula* είναι ο εξής:

formula = and(eq(beta5, penc(alpha3, beta4, alpha4)), and(eq(beta1 ,penc(alpha1, beta4, alpha2)), or(eq(alpha1,beta2), eq(alpha1,beta3)))).

Οι μεταβλητές *betaX* αναφέρονται στα δημόσια στοιχεία της απόδειξης, με τη σειρά με την οποία αυτά έχουν γραφτεί. Ομοίως οι μεταβλητές *alphaY*, για τα ιδιωτικά στοιχεία. Τα *and* και *or* αποτελούν τους γνωστούς λογικούς συνδέσμους και το "κατηγόρημα" *eq* μοντελοποιεί την ισότητα[2]. Κατά συνέπεια, ο προηγούμενος τύπος δηλώνει ότι: *enc_cred = penc(credential, EA_pkey, rand1)* ∧ *enc_vote = penc(vote, EA_pkey, rand)* ∧ *(vote = a* ∨ *vote = b)*, δηλαδή ότι το *enc_cred* είναι μια κρυπτογράφηση του *credential* με το δημόσιο κλειδί *EA_pkey* με χρήση τυχαιότητας *rand1*, ομοίως το *enc_vote* είναι μια κρυπτογράφηση του *vote* με το κλειδί *EA_pkey* με χρήση της τυχαιότητας *rand* και το *vote* είναι *a* ή *b*. Τα ελεύθερα ονόματα *a* και *b* μοντελοποιούν τις διαθέσιμες έγκυρες ψήφους. Παρατηρούμε ότι η μοντελοποίηση αυτή θυμίζει το συμβολισμό που χρησιμοποιείται συνήθως στις περιγραφές πρωτοκόλλων για τις αποδείξεις μηδενικής γνώσης [20]. Φυσικά, ο εφαρμοσμένος π-λογισμός και η ProVerif δεν υποστηρίζουν τέτοιους όρους. Στο [45] όμως, έχει υλοποιηθεί ένας preprocessor για την ProVerif γραμμένος σε Java, ο οποίος αναλαμβάνει να μεταφράσει τα παραπάνω κατάλληλα και ορθά ώστε να γίνουν αποδεκτά από την ProVerif. Ουσιαστικά, ορίζονται κάποιοι επιπλέον constructors που αντικαθιστούν το *zkp* και το *zkver* (βλ. παρακάτω), ενώ η θεωρία ισότητας εμπλουτίζεται με κάποιες επιπλέον εξισώσεις για αυτούς, που μοντελοποιούν τις αποδείξεις μηδενικής γνώσης. Δίνονται επίσης εξισώσεις για τους constructors *publicN* (βλ. παρακάτω), που δίνουν τη δυνατότητα σε μία διεργασία να έχει πρόσβαση στο N-οστό δημόσιο στοιχείο της απόδειξης, με τη σειρά που αυτά εμφανίζονται στο *zkp* (ίδια σειρά με τα *betaX*). Θα χρησιμοποιήσουμε αυτόν τον τρόπο μοντελοποίησης και παρουσίασης του πρωτοκόλλου, όπως γίνεται και στο [35].
Στο πλαίσιο 4.4 που ακολουθεί, εμφανίζεται η αρχή για την εξουσιοδότηση. Η αρχή αυτή κάνει input από το ιδιωτικό κανάλι *internalCH*, το αναγνωριστικό *id* και το κρυπτογραφημένο διαπιστευτήριο *enc_cred* που αντιστοιχεί σε αυτό. Έπειτα, "αναζητά" στο bulletin board την απόδειξη μηδενικής γνώσης που έχει δημοσιευτεί γι' αυτό το αναγνωριστικό. Κατόπιν, ελέγχεται η ορθότητα της απόδειξης *zkp* μέσω της συνάρτησης *zkver*, η οποία ελέγχει ότι η *zkp* έχει όλα τα απαιτούμενα στοιχεία και είναι μια απόδειξη της πρότασης που περιγράφεται από τον τύπο *formula* που είδαμε παραπάνω. Εφόσον η απόδειξη είναι σωστή, χρησιμοποιούνται οι συναρτήσεις *publicN* προκειμένου να απομονωθούν τα δημόσια στοιχεία που μας ενδιαφέρουν, εδώ η τυφλή ψήφος *blind_msg*

2 Πρόκειται στην πραγματικότητα για ένα σύμβολο συνάρτησης eq/2 το οποίο ικανοποιεί την εξίσωση *eq(x,x)= true*. Ομοίως οι λογικοί σύνδεσμοι ορίζονται με τα σύμβολα and/2 και or/2 τα οποία ικανοποιούν τις εξισώσεις: *or(x,true) = true*, *or(true,x)=true* και *and(true,true)=true*.

και το κρυπτογραφημένο πιστοποιητικό *enc_cred2*. Τέλος, η αρχή προχωρά στον έλεγχο ότι τα δύο κρυπτοκείμενα *enc_cred* και *enc_cred2* αποτελούν κρυπτογραφήσεις του ίδιου plaintext. Αν ο έλεγχος για την ισοδυναμία των plaintext είναι επιτυχής, η αρχή υπογράφει την ψήφο με έγκυρο τρόπο (θέτει το πεδίο valid ως *true*) ενώ διαφορετικά την υπογράφει μεν αλλά με άκυρο τρόπο δε (θέτει *false* στο προηγούμενο πεδίο). Υπενθυμίζουμε ότι το πεδίο valid μπορεί να ελεγχθεί μόνο από κάποια άλλη αρμόδια αρχή (απαιτεί γνώση του ιδιωτικού κλειδιού).

```
authorization :=
internalCH(id,enc_cred).
BB(id2,zkp).
if id = id2 then
if zkver(4;6;formula;zkp) = true then
if public4(zkp) = EA_pkey then
let blind_msg = public6(zkp) in
let enc_cred2 = public5(zkp) in
if pet(enc_cred,enc_cred2,EA_skey) = true then
      out(BB,sign(blind_msg,EA_skey,true))
else
      out(BB,sign(blind_msg,EA_skey,false))
```

4.4 Η αρχή για την εξουσιοδότηση σε εφαρμοσμένο π-λογισμό

Τέλος, έχουμε την αρμόδια αρχή για την καταμέτρηση των ψήφων, της οποίας η διεργασία δίνεται παρακάτω:

```
tallying_authority:=
BB(sig,zkp).
if zkver(4;6;formula;zkp) = true then
let enc_vote = public1(zkp) in
if versign(sig,pk(EA_skey)) = enc_vote then

if public2(zkp)=a  then
if public3(zkp)=b  then
if public4(zkp)=EA_pkey then
vrand.
let reenc_vote = reencrypt(enc_vote,rand) in
if getvalid(sig,EA_skey) = true then
let vote = dec(reenc_vote,EA_skey) in
BB<(reenc_vote,vote)>
```

4.5 Η αρχή για την καταμέτρηση σε εφαρμοσμένο π-λογισμό

Αρχικά η διεργασία κάνει input από το bulletin board την υπογραφή και την απόδειξη μηδενικής γνώσης που κατέθεσε κάποιος ψηφοφόρος. Ελέγχεται έπειτα η ορθότητα της υπογραφής και της απόδειξης. Παρατηρούμε τον έλεγχο για τα δημόσια στοιχεία της απόδειξης μέσω των συναρτήσεων *public2* και *public3*: η αρχή πρέπει να είναι βέβαιη ότι έχει στα χέρια της μια έγκυρη ψήφο (δηλαδή είτε a είτε b) αλλιώς η ψήφος απορρίπτεται. Έπειτα η αρχή προχωρά στην επανακρυπτογράφηση της ψήφου κάνοντας χρήση του *reencrypt*: παράγει μια νέα τυχαιότητα *rand*, η οποία συνδυάζεται με την υπάρχουσα τυχαιότητα μέσω μιας συνάρτησης f (η ακριβής περιγραφή της οποίας είναι αδιάφορη) για

να παραγχθεί μια νέα κρυπτογράφηση για την ψήφο *reenc_vote*. Τέλος ελέγχεται η εγκυρότητα της υπογραφής (το πεδίο valid δηλαδή), διότι η αρχή για την εξουσιοδότηση θα μπορούσε να έχει θέσει το πεδίο *valid* ως *false* στην οποία περίπτωση η ψήφος θα έπρεπε να απορριφθεί. Αν το πεδίο αυτό είναι *true* τότε η ψήφος καταμετράται. Όπως αναφέρθηκε παραπάνω, δεν μπορούμε να καταμετρήσουμε όντως την ψήφο και να εκδώσουμε ένα αριθμητικό αποτέλεσμα, οπότε απλώς δημοσιεύουμε την ψήφο που περιέχεται στο *reenc_vote* αποκρυπτογραφώντας το.

4.4 Η ορθότητα του πρωτοκόλλου

Στην προηγούμενη ενότητα είδαμε μια λεπτομερή περιγραφή του πρωτοκόλλου που θα μελετήσουμε χρησιμοποιώντας τη γλώσσα του εφαρμοσμένου π-λογισμού για να μοντελοποιήσουμε τις διεργασίες που εμφανίζονται σε αυτό. Θα παρουσιάσουμε τώρα τις διεργασίες αυτές στη γλώσσα μοντελοποίησης της ProVerif, προκειμένου να αποδείξουμε την ιδιότητα της ορθότητας. Όπως είναι φανερό από τους ορισμούς 4.5 και 4.6 παραπάνω, οι διεργασίες πρέπει να επεκταθούν ώστε να περιέχουν τα κατάλληλα γεγονότα. Παράλληλα είμαστε αναγκασμένοι να προβούμε και σε μερικές ακόμα αλλαγές, τις οποίες θα σχολιάσουμε παρακάτω.
Ξεκινάμε με την διεργασία για το registration authority:

```
let registration_authority =
in(idCH,nonce);
new id;
event NEWID(id);
out(idCH,(id,nonce));
new credential;
new random_v;
let enc_cred = penc(credential,EA_pkey,random_v) in
out(BB,(enc_cred,id));
out(safeCH,(credential,nonce));
out(internalCH,(id,enc_cred))
.
```

4.6 Η αρχή για την εγγραφή σε ProVerif

Στην παραπάνω διεργασία παρατηρούμε καταρχάς το γεγονός *NEWID* που εκτελείται με τον όρο *id*, που είναι το νέο αναγνωριστικό που παράγεται, όπως προβλέπει ο ορισμός 4.5. Παρατηρούμε ακόμα ότι γίνεται χρήση κάποιου *nonce*. Ο λόγος για τον οποίο συμβαίνει αυτό, είναι ότι τα ιδιωτικά κανάλια *idCH* και *safeCH* χρησιμοποιούνται από πολλούς ψηφοφόρους ταυτόχρονα. Κατά συνέπεια υπάρχει ο κίνδυνος να πάρουν περισσότεροι του ενός ψηφοφόρου τα ίδια αναγνωριστικά ή/και διαπιστευτήρια. Χρησιμοποιώντας μια τέτοιου είδους "χειραψία nonce" (nonce handshake) αποφεύγουμε αυτό το πρόβλημα και πετυχαίνουμε κάθε ψηφοφόρος να παραλαμβάνει μοναδικό αναγνωριστικό και διαπιστευτήριο όπως προβλέπει το πρωτόκολλο. Η διαδικασία αυτή είναι συχνή στη βιβλιογραφία της ProVerif [35] και αναγκαία λόγω του ακαθόριστου πλήθους διεργασιών που τρέχουν παράλληλα όταν χρησιμοποιείται η αντιγραφή διεργασιών.
Η αρχή για την εξουσιοδότηση εκτελεί τη διεργασία που εμφανίζεται στο πλαίσιο 4.7. Πρόκειται για παρόμοια διεργασία με αυτή που δόθηκε παραπάνω σε εφαρμοσμένο π-λογισμό.

```
let authorization_authority =
in(internalCH,(id,enc_cred));
in(BB,(id2,zkp));
if id = id2 then
if zkver(4;6;formula;zkp)=true then
if public4(zkp)=EA_pkey then
let blind_msg=public6(zkp) in
let enc_cred2=public5(zkp) in
if pet(enc_cred,enc_cred2,EA_skey)=true then
      out(BB,sign(blind_msg,EA_skey,true))
else
      out(BB,sign(blind_msg,EA_skey,false))
.
```

4.7 Η αρχή για την εξουσιοδότηση σε ProVerif

Ακολουθεί η διεργασία που εκτελεί η αρχή για την καταμέτρηση των ψήφων:

```
let tallying_authority =
in(internal2CH,noncev);
new nonce;
out(nonceCH,(nonce,noncev));
in(BB,(sig,zkp));
in(internal2CH,(noncevv,sig2));
if noncevv = nonce then
if sig2 = sig then
if zkver(4;6;formula;zkp) = true then
let enc_vote = public1(zkp) in
if versign(sig,pk(EA_skey)) = enc_vote then

if public2(zkp) = a  then
if public3(zkp) = b  then
if public4(zkp) = EA_pkey then
new rand;
let reenc_vote = reencrypt(enc_vote,rand) in
if getvalid(sig,EA_skey) = true then
let vote = dec(reenc_vote,EA_skey) in
out(BB,(reenc_vote,vote));
event ENDVOTE(vote)
.
```

4.8 Η αρχή για την καταμέτρηση σε ProVerif

Καταρχάς παρατηρούμε την εκτέλεση του γεγονότος ENDVOTE με όρο vote, που είναι μια μεταβλητή που περιέχει τελικά την (plaintext) ψήφο που αποκρυπτογραφήθηκε. Αυτό που αξίζει να σχολιαστεί περαιτέρω, σε σχέση με τη διεργασία που δόθηκε σε π-λογισμό στην προηγούμενη ενότητα, είναι η χρήση της χειραψίας nonce με κάποιον ψηφοφόρο στις πρώτες τρεις εντολές της διεργασίας. Μέσω αυτών κάθε ψηφοφόρος αποκτά μοναδικό nonce, το οποίο έπειτα ελέγχεται από την παραπάνω διεργασία (5^{η} και 6^{η} εντολή). Στο σημείο αυτό παρεκκλίνουμε από το πρωτόκολλο λόγω κάποιων αδυναμιών της ProVerif στη μοντελοποίηση. Το πρωτόκολλο προβλέπει ότι η διεργασία αυτή θα πρέπει να διατηρήσει ψήφους με διακριτά κρυπτοκείμενα, προκειμένου να αποφευχθούν οι

διπλότυπες ψήφοι. Δηλαδή, αν κάποιος κακόβουλος ψηφοφόρος ήθελε, θα μπορούσε αφού λάβει την υπογεγραμμένη ψήφο του να την καταθέσει πολλές φορές με την ελπίδα να καταμετρηθεί περισσότερες από μία φορά. Τότε το κρυπτοκείμενο σε όλα τα διπλότυπα θα ήταν το ίδιο. Κατά συνέπεια και για να αποφευχθεί αυτό, η διεργασία της καταμέτρησης διατηρεί μόνο μία φορά το κάθε κρυπτοκείμενο, όπως προαναφέραμε. Το πρόβλημα εδώ είναι ότι κάτι τέτοιο δεν μπορεί να μοντελοποιηθεί σε εφαρμοσμένο π-λογισμό και σε ProVerif αφού τα μηνύματα δεν αποθηκεύονται σε κάποιο board αλλά απλώς ανταλλάσσονται μεταξύ των διεργασιών. Χρησιμοποιούμε μια τεχνική για να προσομοιώσουμε τη λειτουργία αυτή. Η τεχνική αυτή, που εμφανίζεται στη βιβλιογραφία της ProVerif [35], στηρίζεται στην παραγωγή ενός nonce από μία διεργασία, την αποστολή του nonce σε μία άλλη και έπειτα στον έλεγχο ενός μηνύματος το οποίο θα περιλαμβάνει το ίδιο nonce. Επειδή το nonce είναι μοναδικό, ένα τέτοιο μήνυμα μπορεί να ληφθεί μόνο μία φορά, λύνοντας έτσι το πρόβλημα των διπλοτύπων. Εδώ το nonce στέλνεται από τον ψηφοφόρο προς την αρχή για την καταμέτρηση στο ιδιωτικό κανάλι *internal2CH* μαζί με την υπογεγραμμένη ψήφο *sig2*, ούτως ώστε να ξέρει η διεργασία σε ποιο κρυπτοκειμένο αναφέρεται το nonce.
Δίνουμε, τέλος, τη διεργασία που εκτελεί ένας ψηφοφόρος:

```
let honest_voter =
 new idnonce;
 out(idCH,idnonce);
 in(idCH,(id,nonceid));
 if idnonce=nonceid then

 event STARTID(id);
 in(safeCH,(credential,noncecred));
 if noncecred=idnonce then

 new rand;
 in(voteCH,vote);
 event BEGINVOTE(id,vote);
 let enc_vote = penc(vote,EA_pkey,rand) in
 new rand1;
 let enc_cred = penc(credential,EA_pkey,rand1) in
 new blind_factor;
 let blind_vote = blind(enc_vote,blind_factor) in
 let zkp = zk(vote,rand,credential,rand1;
enc_vote,a,b,EA_pkey, enc_cred,blind_vote;formula) in
 out(BB,(id,zkp));

 in(BB,blind_sig);
 if versign(blind_sig,EA_pkey)=blind_vote then
 out(internal2CH,idnonce);
 in(nonceCH,(nonce,noncet));
 if noncet=idnonce then

 let sig = unblind(blind_sig,blind_factor) in
 out(BB,(sig,zkp));
 out(internal2CH,(nonce,sig))
```

4.9 Η διεργασία που εκτελούν οι ψηφοφόροι

Σε σχέση με τη διεργασία που παρουσιάστηκε παραπάνω στη γλώσσα του εφαρμοσμένου π-λογισμού, έχουμε επιπλέον τις εντολές που σχετίζονται με τις χειραψίες nonce, των οποίων η λειτουργία προσδιορίστηκε στις προηγούμενες παραγράφους. Βλέπουμε τέτοιες εντολές στην αρχή, προκειμένου να ληφθεί μοναδικό αναγνωριστικό, πριν το γεγονός *BEGINVOTE* προκειμένου να ληφθεί το σωστό διαπιστευτήριο και στο τέλος, προκειμένου να ληφθεί μοναδικό nonce που θα χρησιμοποιηθεί από τη διεργασία για την καταμέτρηση. Τέλος, παρατηρούμε τα γεγονότα *STARTID* με όρο το *id* που λαμβάνεται από την αρχή για την εγγραφή και *BEGINVOTE* με όρους το προηγούμενο *id* και την ψήφο *vote* που θα ψηφίσει ο ψηφοφόρος. Η διεργασία μέσω της οποίας αποφασίζει ο κάθε ψηφοφόρος την ψήφο του είναι η εξής:

```
let vote_chooser =
out(voteCH,a) |
out(voteCH,b)
.
```

Η διεργασία αυτή προσομοιώνει την τυχαία επιλογή της ψήφου για τον κάθε ψηφοφόρο. Μένει τώρα να συνδέσουμε όλες τις παραπάνω (υπο-) διεργασίες σε μια διεργασία που θα έχει τη μορφή του ορισμού 4.2. Η διεργασία αυτή δίνεται στον πίνακα 4.10 που ακολουθεί.

```
process

new EA_skey;
let EA_pkey = pk(EA_skey) in
out(publicCH,EA_pkey);
new safeCH;
new voteCH;
new internalCH;
new internal2CH;
new idCH;
new nonceCH;
(
    !registration_authority |
    !authorization_authority |
    !vote_chooser |
    !tallying_authority |
    !honest_voter
)
                                .
```

4.10 Η διεργασία για την απόδειξη της ορθότητας

Βλέπουμε ότι δημιουργούνται ένα νέο ζεύγος ιδιωτικού και δημόσιου κλειδιού για την αρμόδια αρχή των εκλογών και τα ιδιωτικά κανάλια που χρησιμοποιεί το πρωτόκολλο (αντιστοιχούν στα ονόματα **n** του ορισμού 4.2). Σημειώνουμε ότι σε σχέση με τον ορισμό 4.2, έχουμε μία αρμόδια αρχή $A_1 := !registration\_authority \mid !authorization\_authority \mid !tallying\_authority$. Τέλος, παρατηρούμε ότι οι ψηφοφόροι

βρίσκονται υπό αντιγραφή, προκειμένου να μοντελοποιήσουμε ένα απροσδιόριστο πλήθος πεπερασμένων ψηφοφόρων, όπως γίνεται συνήθως στη βιβλιογραφία [35].
Το ερώτημα που διερευνά η ProVerif για την παρούσα διεργασία ψηφοφορίας είναι το εξής:

```
query evinj: ENDVOTE(v) ==> (evinj:BEGINVOTE(i,v) ==> (evinj:STARTID(i) ==>
evinj:NEWID(i))).
```

Η ProVerif πράγματι αποδεικνύει, εντός μερικών δευτερολέπτων, ότι η αντιστοιχία που περιγράφεται στο ερώτημα ισχύει για το πρωτόκολλο, έτσι όπως το περιγράψαμε στις προηγούμενες παραγράφους. Αν και το πρωτόκολλο απλοποιήθηκε σε σχέση με την αρχική περιγραφή του, όπως αναφέραμε στην προηγούμενη ενότητα, οι αλλαγές που παρουσιάστηκαν στις προηγούμενες παραγράφους της ενότητας αυτής δεν θεωρούμε ότι επηρεάζουν την ακρίβεια του μοντέλου: ουσιαστικά προσθέσαμε τα nonces προκειμένου να είμαστε σίγουροι ότι κάθε ψηφοφόρος θα λάβει μοναδικό αναγνωριστικό και διαπιστευτήριο εφόσον οι ψηφοφόροι μοιράζονταν τα ιδιωτικά κανάλια και χρησιμοποιήσαμε ένα nonce για να αποφύγουμε μια επίθεση με διπλότυπες ψήφους, μοντελοποιώντας έτσι τη διαδικασία που ακολουθεί η αρχή για την καταμέτρηση στο πρωτόκολλο για τον ίδιο λόγο (διατηρεί ψήφους με διακριτά κρυπτοκείμενα, διαγράφοντας τα διπλότυπα από το bulletin board).

4.5 Μυστικότητα της ψήφου

Παρουσιάζουμε παρακάτω τις διεργασίες που χρησιμοποιήσαμε προκειμένου να αποδείξουμε ότι το πρωτόκολλο ικανοποιεί τον ορισμό 4.4 περί μυστικότητας της ψήφου. Θεωρώντας τη βασική περίπτωση των δύο μόνο ψηφοφόρων, όπως αναφέραμε παραπάνω, πρέπει να δείξουμε ότι:

$$\mathrm{VP}_{\{1,2\}}\,[\,\mathrm{V}\sigma_{\mathrm{id1}}\sigma_{\upsilon 1}\mid \mathrm{V}\sigma_{\mathrm{id2}}\sigma_{\upsilon 2}\,] \approx \mathrm{VP}_{\{1,2\}}\,[\,\mathrm{V}\sigma_{\mathrm{id1}}\sigma_{\upsilon 2}\mid \mathrm{V}\sigma_{\mathrm{id2}}\sigma_{\upsilon 1}\,]$$

το οποίο στην περίπτωσή μας (δύο μόνο ψηφοφόροι και μία αρμόδια αρχή) μετατρέπεται σε:

$$\mathrm{vn}_1.\ \mathrm{vn}_2.\ ...\ \mathrm{vn}_\mathrm{k}.\ (\mathrm{V}\sigma_{\mathrm{id1}}\sigma_{\upsilon 1}\mid \mathrm{V}\sigma_{\mathrm{id2}}\sigma_{\upsilon 2}\mid \mathrm{A}_1\ ) \approx \mathrm{vn}_1.\ \mathrm{vn}_2.\ ...\ \mathrm{vn}_\mathrm{k}.\ (\mathrm{V}\sigma_{\mathrm{id1}}\sigma_{\upsilon 2}\mid \mathrm{V}\sigma_{\mathrm{id2}}\sigma_{\upsilon 1}\mid \mathrm{A}_1\ )$$

Στην ενότητα 3.4 παρουσιάσαμε τον τρόπο με τον οποίο μπορούν να αποδειχτούν ισοδυναμίες όπως η προηγούμενη στην ProVerif. Πρέπει να χρησιμοποιηθεί η έννοια της δι-διεργασίας και όροι που να περιέχουν τον "όρο" choice, ώστε να κατασκευαστεί μια δι-διεργασία P που να ικανοποιεί diff-equivalence. Από όσα αναφέρονται στην παράγραφο εκείνη προκύπτει ότι πρέπει να σχηματιστεί biprocess P τέτοιο ώστε:

$$\begin{aligned}\mathrm{fst(P)} &= \mathrm{vn}_1.\ \mathrm{vn}_2.\ ...\ \mathrm{vn}_\mathrm{k}.\ (\mathrm{V}\sigma_{\mathrm{id1}}\sigma_{\upsilon 1}\mid \mathrm{V}\sigma_{\mathrm{id2}}\sigma_{\upsilon 2}\mid \mathrm{A}_1\ )\\ &\text{και}\\ \mathrm{snd(P)} &= \mathrm{vn}_1.\ \mathrm{vn}_2.\ ...\ \mathrm{vn}_\mathrm{k}.\ (\mathrm{V}\sigma_{\mathrm{id1}}\sigma_{\upsilon 2}\mid \mathrm{V}\sigma_{\mathrm{id2}}\sigma_{\upsilon 1}\mid \mathrm{A}_1\ )\end{aligned}$$

Έτσι, αν η διεργασία P τελικά ικανοποιεί diff-equivalence, θα ισχύει το ζητούμενο. Δεν είναι δύσκολο να δούμε ότι η διεργασία P πρέπει να έχει την εξής μορφή:

$$\mathrm{P}{:=}\ \mathrm{vn}_1.\ \mathrm{vn}_2.\ ...\ \mathrm{vn}_\mathrm{k}.\ (\mathrm{V}\sigma_{\mathrm{id1}}\,\mathrm{choice}[\sigma_{\upsilon 1},\sigma_{\upsilon 2}]\mid \mathrm{V}\sigma_{\mathrm{id2}}\,\mathrm{choice}[\sigma_{\upsilon 2},\sigma_{\upsilon 1}]\mid \mathrm{A}_1\ )$$

όπου καταχρηστικά εφαρμόσαμε την κατασκευή choice σε αντικαταστάσεις. Στην πραγματικότητα, δεν θα έχουμε αντικαταστάσεις στην ProVerif αλλά το choice θα εφαρμόζεται στα ελεύθερα ονόματα που μοντελοποιούν τις διαθέσιμες ψήφους.
Παρουσιάζουμε τώρα τις διεργασίες που χρησιμοποιούμε, στη γλώσσα μοντελοποίησης της ProVerif. Ξεκινάμε με την αρχή για την εγγραφή (πίνακας 4.11) που εκτελεί ουσιαστικά τη διεργασία που είδαμε και στην προηγούμενη ενότητα. Παρατηρούμε ότι φυσικά δεν υπάρχουν γεγονότα στην προηγούμενη διεργασία (και ούτε στις επόμενες - τα γεγονότα απαιτούνται μόνο για την απόδειξη ορθότητας). Ακόμα δεν χρησιμοποιούμε χειραψίες nonce. Ο λόγος που γίνεται αυτό είναι διότι έχουμε απλοποιήσει λίγο την μορφή της διεργασίας αφού εδώ έχουμε μόνο δύο ψηφοφόρους. Έτσι έχουμε θεωρήσει ότι οι δύο ψηφοφόροι έχουν ξεχωριστά ιδιωτικά κανάλια προς τις αρμόδιες αρχές,

επομένως δεν υπάρχει κίνδυνος να μπερδευτούν τα στοιχεία του ενός με του άλλου, άρα δεν υπάρχει λόγος να χρησιμοποιηθούν nonces που δυσχεραίνουν την ανάλυση. Σημειώνουμε ότι κάτι τέτοιο δεν μπορούσε να γίνει στην περίπτωση της ορθότητας, αφού είχαμε ένα απεριόριστο πλήθος ψηφοφόρων (η διεργασία *honest_voter* ήταν εκεί υπο αντιγραφή).

```
let registration_authority =
new id;
out(idCH,id);
new credential;
new random_v;
let enc_cred = penc(credential,EA_pkey,random_v) in
out(BB,(enc_cred,id));
out(safeCH,credential);
out(internalCH,(enc_cred,id))
.
```

4.11 Η αρχή για την εγγραφή (vote-privacy)

Ακολουθεί η διεργασία για την αρχή της εξουσιοδότησης:

```
let authorization_authority =
in(internalCH,(enc_cred,id));
sync 1;
in(BB,(id2,zkp));
if id=id2 then
if zkver(4;6;formula;zkp) = true then
if public4(zkp) = EA_pkey then
let blind_msg = public6(zkp) in
let enc_cred2 = public5(zkp) in
if pet(enc_cred,enc_cred2,EA_skey) = true then
      out(BB,sign(blind_msg,EA_skey,true))
else
      out(BB,sign(blind_msg,EA_skey,false))
```

4.12 Η αρχή για την εξουσιοδότηση (vote-privacy)

Παρατηρούμε ότι πρόκειται ακριβώς για την ίδια διεργασία με αυτή που χρησιμοποιήθηκε για την απόδειξη της ορθότητας στην προηγούμενη ενότητα, με μία βασική διαφορά, την εντολή *sync*. Με την εντολή αυτή, θέλουμε να συγχρονίσουμε τη φάση της εξουσιοδότησης (*sync 1*) ώστε να ξεκινά "ταυτόχρονα" για όλες τις διεργασίες. Θα χρησιμοποιήσουμε συγχρονισμό και για τη φάση της καταμέτρησης των ψήφων (*sync 2*) παρακάτω. Όπως αναφέραμε στην αρχική περιγραφή του πρωτοκόλλου, ο συγχρονισμός αυτός προβλέπεται στο [20] αλλά δεν μπορούσε να χρησιμοποιηθεί στην απόδειξη της ορθότητας διότι οι διεργασίες ήταν υπό αντιγραφή και όπως αναφέραμε στην ενότητα 3.4, στην παράγραφο για τις εντολές συγχρονισμού, η ProVerif δεν επιτρέπει να βρίσκονται οι εντολές sync υπό αντιγραφή. Η χρήση του *sync* επιτρέπει ακόμα να χρησιμοποιηθεί η

εναλλαγή δεδομένων που περιγράψαμε στην ενότητα 3.4, που βοηθά στις αποδείξεις παρατηρήσιμων ισοδυναμιών, όπως ακριβώς στην περίπτωσή μας.
Ακολουθεί η διεργασία για την αρχή καταμέτρησης:

```
let tallying_authority =
new nonce;
out(nonceCH,nonce);
in(BB,(sig,zkp));
in(internal2CH,(noncevv,sig2));
if noncevv = nonce then
if sig2 = sig then
if zkver(4;6;formula;zkp)=true then
let enc_vote = public1(zkp) in
if versign(sig,pk(EA_skey))=enc_vote then
if public2(zkp) = a  then
if public3(zkp) = b  then
if public4(zkp) = EA_pkey then
sync 2;
new rand;
let reenc_vote = reencrypt(enc_vote,rand) in
if getvalid(sig,EA_skey) = true then
let vote = dec(reenc_vote,EA_skey) in
out(BB,(reenc_vote,vote))
.
```

4.13 Η αρχή για την καταμέτρηση (vote-privacy)

Παρατηρούμε εδώ τη χρήση του *nonce*, προκειμένου να αποφύγουμε επιθέσεις με επανάληψη ψήφων από τον εχθρό, όπως αναφέραμε στην προηγούμενη ενότητα. Ακόμα χρησιμοποιείται η εντολή *sync* προκειμένου η διαδικασία της καταμέτρησης να ξεκινήσει αφού έχει ολοκληρωθεί η κατάθεση των ψήφων, όπως προβλέπει το πρωτόκολλο.
Για την απόδειξη της μυστικότητας της ψήφου, χρησιμοποιούνται δύο διεργασίες ψηφοφόρων, προκειμένου να έχουμε δύο ψηφοφόρους με διαφορετικά αναγνωριστικά. Αναφορικά με όσα περιγράψαμε στην αρχή αυτής της ενότητας, η μία διεργασία αντιστοιχεί στη διεργασία $V\sigma_{id1}$ και η άλλη στη $V\sigma_{id2}$. Η επιλογή της ψήφου γίνεται χρησιμοποιώντας την κατασκευή choice. Οι δύο διεργασίες παρουσιάζονται στον πίνακα 4.14. Προφανώς οι δύο διεργασίες είναι ίδιες εκτός από την αντιστροφή των ψήφων μέσα στο *choice*, όπως ακριβώς παρουσιάσαμε στην αρχή της ενότητας. Τέλος παρατηρούμε την εντολή συγχρονισμού *sync 1;* ώστε η κατάθεση των ψήφων να γίνει στη φάση της εξουσιοδότησης και όχι νωρίτερα.

```
let voter1 =
in(idCH,id);
in(safeCH,credential);
sync 1;
new rand;
let vote = choice[a,b] in
let enc_vote = penc(vote,EA_pkey,rand) in
new rand1;
let enc_cred = penc(credential,EA_pkey,rand1) in
new blind_factor;
let blind_vote = blind(enc_vote,blind_factor) in
let zkp = zk(vote,rand,credential,rand1; enc_vote,
a,b, EA_pkey, enc_cred,blind_vote;formula) in
out(BB,(id,zkp));

in(BB,blind_sig);
if versign(blind_sig,EA_pkey)=blind_vote then
in(nonceCH,nonce);
let sig = unblind(blind_sig,blind_factor) in
out(BB,(sig,zkp));
out(internal2CH,(nonce,sig))
.

let voter2 =
in(idCH,id);
in(safeCH,credential);
sync 1;
new rand;
let vote = choice[b,a] in
let enc_vote = penc(vote,EA_pkey,rand) in
new rand1;
let enc_cred = penc(credential,EA_pkey,rand1) in
new blind_factor;
let blind_vote = blind(enc_vote,blind_factor) in
let zkp = zk(vote,rand,credential,rand1;
enc_vote,a,b,EA_pkey,enc_cred,blind_vote;formula) in
out(BB,(id,zkp));

in(BB,blind_sig);
if versign(blind_sig,EA_pkey)=blind_vote then
in(nonceCH,nonce);
let sig = unblind(blind_sig,blind_factor) in
out(BB,(sig,zkp));
out(internal2CH,(nonce,sig))
.
```

4.14 Οι διεργασίες για τους δύο ψηφοφόρους (vote-privacy)

Η συνολική διεργασία που χρησιμοποιεί τις προηγούμενες υποδιεργασίες δίνεται στο πλαίσιο 4.15. Η διεργασία αυτή είναι παρόμοια με αυτή που είδαμε για την απόδειξη της ορθότητας, με τη διαφορά ότι εδώ, όπως αναφέραμε παραπάνω, ο κάθε ψηφοφόρος έχει δικά του ιδιωτικά κανάλια για την επικοινωνία με τις αρμόδιες αρχές και όχι διαμοιραζόμενα με τους άλλους ψηφοφόρους. Προς απλοποίηση της διεργασίας, θέσαμε

τα ίδια ονόματα στα ιδιωτικά κανάλια, διαφορετικά θα έπρεπε να έχουμε δύο διεργασίες για την κάθε αρχή (δηλαδή θα είχαμε κανάλια *idCH1* και *idCH2* και διεργασίες *registration_authority1* που θα χρησιμοποιούσε το κανάλι *idCH1* και *registration_authority2* που θα χρησιμοποιούσε το *idCH2* - αντίστοιχα για τα άλλα κανάλια και διεργασίες). Προφανώς, οι δύο αυτές μορφές είναι ισοδύναμες και δεν επηρεάζουν την ανάλυση.
Στην περίπτωση της παρατηρήσιμης ισοδυναμίας δεν χρειάζεται να διατυπώσουμε κάποιο ερώτημα. Η ProVerif προσπαθεί να αποδείξει το diff-equivalence για το παραπάνω biprocess κατευθείαν, εφόσον εμπεριέχονται όροι με την κατασκευή choice. Πράγματι η ProVerif καταφέρνει να αποδείξει το diff-equivalence εντός μερικών λεπτών, άρα ισχύει η ισοδυναμία για τις δύο διεργασίες που περιλαμβάνονται στη δι-διεργασία και κατά συνέπεια το πρωτόκολλο ικανοποιεί τη μυστικότητα της ψήφου.

```
process
new EA_skey;
let EA_pkey = pk(EA_skey) in
out(publicCH,EA_pkey);
(
new idCH;
new safeCH;
new nonceCH;
new internalCH;
      voterA |
      tallying_authority |
      registration_authority |
      authorization_authority
)
|
(
new idCH;
new safeCH;
new nonceCH;
new internalCH;
      voterB |
      tallying_authority |
      registration_authority |
      authorization_authority
)
```

4.15 Η διεργασία για την απόδειξη της μυστικότητας ψήφου

Κεφάλαιο 5

5.1 Συμπεράσματα και προοπτικές

Στην παρούσα εργασία μελετήθηκε η τυπική γλώσσα του εφαρμοσμένου π-λογισμού και η εφαρμογή του στη διατύπωση και ανάλυση ιδιοτήτων ασφαλείας κρυπτογραφικών πρωτοκόλλων. Επιπλέον, χρησιμοποιήθηκε το εργαλείο ανάλυσης ProVerif για να αποδειχθούν αυτόματα οι επιθυμητές ιδιότητες. Επιλέχθηκε ένα σύγχρονο πρωτόκολλο ηλεκτρονικής ψηφοφορίας, προκειμένου να δοκιμαστούν οι δυνατότητες αυτής της τυπικής μεθόδου. Στα παραπάνω περιλαμβάνονται αρκετά διαφορετικά αλλά όχι ανεξάρτητα ζητήματα.

Η τυπική γλώσσα του εφαρμοσμένου π-λογισμού είναι μια γλώσσα περιγραφής διεργασιών και των μηνυμάτων που αυτές ανταλλάσσουν. Επιπλέον, μπορούν να οριστούν έννοιες όπως η παρατηρήσιμη ισοδυναμία ή η εκτέλεση γεγονότων και οι αντιστοιχίες, μέσω των οποίων δίνονται ορισμοί των διαφόρων ιδιοτήτων ασφαλείας που θέλουμε να ισχύουν σε ένα πρωτόκολλο. Η γλώσσα αυτή περιγράφει τα πρωτόκολλα και τις ιδιότητες ασφαλείας στο συμβολικό μοντέλο. Τα κρυπτογραφικά primitives γίνονται εξισώσεις. Εισάγεται έτσι μια πηγή αβεβαιότητας ως προς την κρυπτογραφική ισχύ των αποτελεσμάτων που αποδεικνύονται στον εφαρμοσμένο π-λογισμό, στην περίπτωση που ένα πρωτόκολλο αποδεικνύεται ασφαλές στο μοντέλο αυτό. Όπως περιγράψαμε στην ενότητα 1.2, υπάρχουν κάποια θετικά αποτελέσματα, με την έννοια ότι σε ένα πρωτόκολλο που είναι αποδεδειγμένα ασφαλές στο συμβολικό μοντέλο, οι "ιδεατές" εξισώσεις για κάποια primitives (όπως για την κρυπτογράφηση ή για τις ψηφιακές υπογραφές) μπορύν να αντικατασταθούν από τα πραγματικά κρυπτογραφικά primitives του υπολογιστικού μοντέλου, με το πρωτόκολλο να διατηρεί την ασφάλειά του [22]. Παρ'όλα αυτά κάτι τέτοιο δεν ισχύει για όλα τα κρυπτογραφικά primitives (σημαντική εξαίρεση αποτελούν πχ οι συναρτήσεις κατακερματισμού). Βλέπουμε έτσι, ότι μια πιθανή μελλοντική ερευνητική πορεία είναι η περαιτέρω μελέτη των ιδιοτήτων ορθότητας που ισχύουν μεταξύ του συμβολικού και του υπολογιστικού μοντέλου. Στην κατεύθυνση αυτή θα μπορούσε για παράδειγμα να επεκταθεί η έννοια ορθότητας BRSIM (βλ. ενότητα 1.2) σε όσο το δυνατόν περισσότερα κρυπτογραφικά primitives (πχ αποδείξεις μηδενικής γνώσης).

Σχετικά με τον εφαρμοσμένο π-λογισμό συγκεκριμένα, θα ήταν ενδιαφέρον να μελετηθούν κάποιες επεκτάσεις του προκειμένου να μπορούν να μοντελοποιηθούν με μεγαλύτερη ακρίβεια στοιχεία, όπως για παράδειγμα το bulletin board που εμφανίζεται στο πρωτόκολλο που μελετήσαμε. Σημειώνουμε ότι μοντελοποιήσαμε το bulletin board ως ένα απλό δημόσιο κανάλι, αλλά αναγκαστήκαμε να κάνουμε κάποιες αλλαγές στο πρωτόκολλο λόγω αυτής της προσέγγισης. Ένα δημόσιο κανάλι δεν αποθηκεύει τα μηνύματα με κάποιον τρόπο άρα οι διεργασίες μας δεν μπορούσαν να ψάξουν τις δημοσιεύσεις με τις ψήφους των ψηφοφόρων ή με τα στοιχεία τους ή ακόμη και να

διαγράψουν τις διπλότυπες ψήφους, ενώ το πρωτόκολλο προβλέπει τις λειτουργίες αυτές. Προς την κατεύθυνση αυτή ενός εφαρμοσμένου π-λογισμού με καθολικές καταστάσεις (global states) γίνεται ήδη θεωρητική δουλειά [47]. Ακόμη, στο [48] δίνεται μια επέκταση της ProVerif, η οποία υποστηρίζει κάποια stateful στοιχεία όπως πίνακες (tables) και μετρητές (counters). Παρ' όλα αυτά δεν υποστηρίζονται ακόμη ένα-προς-ένα ιδιότητες αντιστοιχίας και αποδείξεις παρατηρήσιμης ισοδυναμίας συνεπώς δεν μπορούσαμε να χρησιμοποιήσουμε την επέκταση αυτή στην παρούσα ανάλυση.

Η ανάλυση που παρουσιάσαμε στο κεφάλαιο 4 βασίζεται σε κάποιους ορισμούς που έχουν δοθεί στον εφαρμοσμένο π-λογισμό για τις ιδιότητες ασφαλείας που πρέπει να ικανοποιεί ένα πρωτόκολλο e-voting. Οι ορισμοί αυτοί προσπαθούν να αποδώσουν στη γλώσσα του π-λογισμού τους αντίστοιχους ορισμούς που δίνονται στο υπολογιστικό μοντέλο. Παρ' όλα αυτά, στη βιβλιογραφία της ProVerif και του εφαρμοσμένου π-λογισμού δεν υπάρχει πάντοτε ομοφωνία σχετικά με τους ορισμούς αυτούς. Περιγράψαμε μια τέτοια κατάσταση στην ενότητα 4.2, όπου οι ορισμοί για τη μυστικότητα της ψήφου και την ορθότητα του πρωτοκόλλου εμφανίζονται στο [35] ελαφρώς διαφορετικοί από αυτούς που χρησιμοποιήσαμε εδώ. Μια μελλοντική ερευνητική κατεύθυνση, θα μπορούσε λοιπόν να είναι η συσχέτιση όλων αυτών των ορισμών και η ιεράρχησή τους με βάση τη συνεπαγωγή (δηλ. από τον ορισμό A έπεται ο ορισμός B κοκ). Μια τέτοια μελέτη γίνεται εν μέρει στο [4], αν και δεν θεωρούνται καθόλου οι corrupted voters που αναφέραμε στην ενότητα 4.2 και που εμφανίζονται στο [35]. Ακόμα, ο ορισμός για το coercion resistance στο [4] διαφέρει αρκετά από αυτόν στο [35] χωρίς να γίνεται κάποια ανάλυση της συσχέτισής τους.

Τα πρωτόκολλα ηλεκτρονικής ψηφοφόριας είναι σαφές πως θα παίξουν μεγάλο ρόλο στο μέλλον. Η δυνατότητα διοργάνωσης μιας ψηφοφορίας με μηδενικό σχεδόν κόστος και σε πολύ μικρό χρόνο είναι επιθυμητές ιδιότητες που μόνο τέτοια πρωτόκολλα μπορούν να προσφέρουν. Παρ' όλα αυτά, τα πρωτόκολλα e-voting πρέπει να ικανοποιούν πολλές ιδιότητες ασφαλείας προκειμένου να εξασφαλιστεί η εγκυρότητα του αποτελέσματος αλλά και η εμπιστοσύνη των ψηφοφόρων σε αυτό. Σήμερα, τα κενά ασφαλείας εντοπίζονται κυρίως στις υλοποιήσεις των πρωτοκόλλων και όχι στη θεωρητική σχεδίασή τους. Εργασίες όπως ίσως η παρούσα και σαφώς πολλές άλλες στη βιβλιογραφία, αυξάνουν την εμπιστοσύνη μας ως προς την ασφαλή σχεδίαση ενός πρωτοκόλλου στη θεωρία. Όμως θα ήταν θεμιτό να μπορούμε να αναλύσουμε με αντίστοιχους τρόπους και τον ίδιο τον κώδικα με τον οποίο υλοποιούνται τα διάφορα πρωτόκολλα. Προς την κατεύθυνση αυτή έχει γίνει βέβαια δουλειά όπως στο [49], που μέσω ενός νέου εργαλείου μεταφράζεται κώδικας από τη γλώσσα F# σε ProVerif. Η ProVerif αναλύει έτσι τον ίδιο τον κώδικα που υλοποιεί το πρωτόκολλο. Ακόμη, μπορεί να εξεταστεί η αντίστροφη πορεία, δηλαδή η παραγωγή ασφαλούς κώδικα μέσω μοντέλων που έχουν ήδη ελεγχθεί από την ProVerif [50].

Τέλος, θα πρέπει να αναφερθούμε στο εργαλείο αυτόματης ανάλυσης CryptoVerif [51], το οποίο είναι ένα σύστημα αντίστοιχο με την ProVerif αλλά για το υπολογιστικό μοντέλο. Λόγω αυτού, αποφεύγει κάποιες από τις "ανακρίβειες" του συμβολικού μοντέλου, είναι όμως πιο δύσκολη η χρήση του και οι δυνατότητες ανάλυσής του πιο

περιορισμένες προς το παρόν. Η εξέλιξη του εργαλείου αυτού, η χρήση του για αποδείξεις ασφαλείας σημαντικών πρωτοκόλλων της βιβλιογραφίας και η δυνατότητα ελέγχου της πραγματικής υλοποίησης των πρωτοκόλλων σε κάποια γλώσσα προγραμματισμού θα μπορούσαν να είναι σημαντικές ερευνητικές κατευθύνσεις.

Βιβλιογραφία

[1] "WhatsApp Encryption Overview Technical White Paper". [Online]. Available: https://www.whatsapp.com/security/. [Accessed September 2019].

[2] B. Adida, O. Pereira, O. de Marneffe and J. J. Quisquater, "Electing a University President using Open-Audit Voting: Analysis of real-world use of Helios", in *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections,* USENIX Association, 2009, pp. 10.

[3] Finnish Ministry of Justice Working Group, "Online Voting in Finland. A Feasibility Study 2017" (summary in English). [Online]. Available: https://oikeusministerio.fi/en/article/-/asset_publisher/tyoryhma-nettiaanestyksen-riskit-suuremmat-kuin-hyodyt. [Accessed September 2019].

[4] J. Dreier, "Formal Verification of Voting and Auction Protocols: From Privacy to Fairness and Verifiability", Ph.D. dissertation, Dept. Comp. Sci., Université Grenoble Alpes, 2013.

[5] G. Lowe, "An Attack on the Needham-Schroeder Public Key Authentication Protocol", *Information Processing Letters,* vol. 56 Issue 3, pp. 131-133, Elsevier, 1995.

[6] P. Rösler, C. Mainka and J. Schwenk, "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema", in *Proceedings of 3rd IEEE European Symposium on Security and Privacy (EuroS&P 2018),* IEEE, 2018, pp. 415-429.

[7] D. Dolev and A. Yao, "On the security of Public Key Protocols", *IEEE Transactions on Information Theory,* vol. 29 Issue 2, pp. 198-208, IEEE Press, 1983.

[8] M. Abadi, B. Blanchet and C. Fournet, "The Applied Pi Calculus: Mobile Values, New Names and Secure Communication", *Journal of the ACM,* vol. 65 issue 1, pp. 1-41, ACM, 2018.

[9] B. Blanchet, B. Smyth, V. Cheval and M. Sylvestre, "ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial". [Online]. Available: https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf. [Accessed October 2019].

[10] M. Burrows, M. Abadi and R. Needham. "A Logic of Authentication", *ACM Transactions on Computer Systems,* vol. 8, No. 1, pp. 18-36, ACM, 1990.

[11] G. Lowe, "Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR", in *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAs '96),* Springer, 1996, pp. 147-166.

[12] R. Needham and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers" *Communications of the ACM,* vol. 21 Issue 12, pp. 993-999, ACM, 1978.

[13] G. Lowe, "Casper: A Compiler for the Analysis of Security Protocols", *Journal of Computer Security,* vol. 6, issue 1-2, pp. 53-84, IOS Press, 1998.

[14] A. Armando, D. Basin, Y. Boichut et al., "The avispa tool for the automated validation of internet security protocols and applications", in *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, Springer, 2005, pp. 281–285.

[15] C. Cremers, "The Scyther Tool: Verification, falsification, and analysis of security protocols", in *Proceedings of the 20th International Conference on Computer Aided Verification,* Springer, 2008, pp. 414-418.

[16] C. Cremers, P. Lafourcade and P. Nadeau, "Comparing state spaces in automatic security protocol analysis", in *Formal to Practical Security,* V. Cortier, C. Kirchner, M. Okada, H. Sakurada (eds), Springer, 2009.

[17] V. Cortier and B. Smyth, "Attacking and fixing Helios: An analysis of ballot secrecy", *Journal of Computer Security,* vol. 21, Issue 1, pp. 89-148, IOS Press, 2013.

[18] B. Adida, "Helios: Web-based Open-audit Voting", in *Proceedings of the 17th Conference on Security Symposium*, USENIX Association, 2008, pp. 335-348.

[19] K. Bhargavan, B. Blanchet and N. Kobeissi, "Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate", in *Proceedings of 2017 IEEE Symposium on Security and Privacy,* pp 483-502.

[20] P. Grontas, A. Pagourtzis, A. Zacharakis and B. Zhang, "Towards everlasting privacy and efficient coercion resistance in remote electronic voting", in *Financial Cryptography and Data Security,* Zohar A. et al. (eds), Springer, 2018.

[21] J. Katz and Y. Lindell, *Introduction to Modern Cryptography.* Chapman & Hall/CRC, 2014.

[22] M. Backes, B. Pfitzmann and M. Waidner, "A Composable Cryptographic Library with Nested Operations", in *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA,* ACM, 2003, pp. 220-230.

[23] D. Micciancio and B. Warinschi, "Soundness of formal encryption in the presence of active adversaries", in *Proceedings of 1st Theory of Cryptography Conference (TCC),* vol. 2951, Springer, 2004, pp. 133–151.

[24] M. Backes, B. Pfitzmann and M. Waidner, "Limits of the Reactive Simulatability/UC of Dolev-Yao Models with Hashes", in *Proceedings of the 11th European Symposium on Research in Computer Security,* Springer, 2006, pp. 68.

[25] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway, "Relations among notions of security for public-key encryption schemes", *Advances in Cryptology:*

CRYPTO'98, vol. 1462, pp. 26–45, Springer, 1998.

[26] M. Backes and B. Pfitzmann, "Symmetric encryption in a simulatable Dolev-Yao style cryptographic library", in *Proceedings of the 17th IEEE CSFW,* IEEE Computer Society, 2004, pp. 204–218.

[27] R. Milner, *Communicating and Mobile Systems: the π-calculus.* Cambridge University Press, 1999.

[28] M. Ryan and B. Smyth, "Applied pi Calculus", in *Formal Models and Techniques for Analyzing Security Protocols,* V. Cortier and S. Kremer (eds), IOS Press, 2011.

[29] B. Blanchet, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif", *Foundations and Trends in Privacy and Security,* vol. 1 issue 1-2, pp. 1-135, Now Publishers Inc, 2016.

[30] B. Blanchet, "ProVerif Automatic Cryptographic Protocol Verifier User Manual for Untyped Inputs". [Online]. Available: https://prosecco.gforge.inria.fr/personal/bblanche/proverif/. [Accessed September 2019].

[31] M. Fitting, *First-Order Logic and Automated Theorem Proving*, Springer, 1996.

[32] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules", in *Proceedings of the 14th IEEE Workshop on Computer Security Foundations,* pp. 82, 2001.

[33] B. Blanchet and B. Smyth, "Automated Reasoning for Equivalences in the Applied Pi Calculus with Barriers", *Journal of Computer Security,* vol. 26, pp. 367-422, 2018.

[34] E. Walia and S. Kumar, "Analysis of Electronic Voting System in Various Countries", *International Journal on Computer Science and Engineering,* vol. 3, issue 5, pp. 1825-1830, Engg Journals Publications, 2011.

[35] M. Backes, C. Hritcu and M. Maffei, "Automated Verification of Remote Electronic Voting Protocols in the Applied Pi-calculus", in *Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium,* IEEE Computer Society, 2008, pp. 195-209.

[36] B. Smyth, M. Ryan, S. Kremer and M. Kourjieh, "Towards Automatic Analysis of Election Verifiability Properties", in *Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security,* A. Armando, G. Lowe (eds), pp. 146-163, Springer, 2010.

[37] M. Arapinis, V. Cortier, S. Kremer and M. Ryan, "Practical Everlasting Privacy", in *Principles of Security and Trust,* D. Basin, J. Mitchell (eds), pp. 21-40, Springer, 2013.

[38] A. Juels, D. Catalano and M. Jakobsson, "Coercion-Resistant Electronic Elections", in *Towards Trustworthy Elections,* D. Chaum et al. (eds) pp. 37-63, Springer, 2010.

[39] D. Chaum, "Blind Signatures for Untraceable Payments", in *Advances in*

Cryptology, D. Chaum, R. Rivest, A. Sherman (eds), pp. 199-203, Springer, 1983.

[40] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems", in *Proceedings on Advances in Cryptology - CRYPTO' 86,* Springer, 1987, pp. 186-194.

[41] M. Jakobsson and A. Juels, "Mix and match: Secure function evaluation via ciphertexts", in *Advances in Cryptology — ASIACRYPT 2000,* T. Okamoto (eds), pp. 162-177, Springer, 2000.

[42] R. Cramer, I. Damgard and B. Schoenmakers, "Proofs of partial knowledge and simplied design of witness hiding protocols", in *Advances in Cryptology - CRYPTO '94,* Y. Desmedt (eds), pp. 174-187, Springer, 1994.

[43] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle", in *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques,* Springer, 2012, pp. 263-280.

[44] Y. Desmedt and Y. Frankel, "Threshold Cryptosystems", in *Advances in Cryptology - CRYPTO' 89 Proceedings,* G. Brassard (eds), pp. 307-315, Springer, 1990.

[45] M. Backes, M. Maffei and D. Unruh, "Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol", in *Proceedings of the 2008 IEEE Symposium on Security and Privacy,* pp. 202-215, IEEE Computer Society, 2008.

[46] A. Zacharakis, P. Grontas and A. Pagourtzis, "Conditional Blind Signatures", *IACR Cryptology ePrint Archive,* 2017.

[47] M. Arapinis, J. Liu, E. Ritter and M. Ryan, "Stateful Applied Pi Calculus", in *Principles of Security and Trust,* M. Abadi, S. Kremer (eds), pp. 22-41, Springer, 2014.

[48] V. Cheval, V. Cortier and M. Turuani, "A little more conversation, a little less action, a lot more satisfaction: Global states in ProVerif", in *Proceedings of the 31th IEEE Computer Security Foundations Symposium (CSF'18),* IEEE, 2018, pp. 344-358.

[49] K. Bhargavan, C. Fournet, A. Gordon and S. Tse, "Verified Interoperable Implementations of Security Protocols", in *Proceedings of 19th IEEE Computer Security Foundations Workshop (CSFW'06),* IEEE, 2006, pp. 61.

[50] N. Kobeissi, K. Bhargavan and B. Blanchet, "Automated Verification for Secure Messaging Protocols and their Implementations: A Symbolic and Computational Approach", in *2nd IEEE European Symposium on Security and Privacy,* pp. 435-450, IEEE, 2017.

[51] B. Blanchet, CryptoVerif: A Computationally-Sound Security Protocol Verifier. [Online]. Available: https://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/cryptoverif.pdf. [Accessed October 2019].