ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ
ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΟΜΕΑΣ ΜΑΘΗΜΑΤΙΚΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΡΙΣΤΟΣ ΧΑΣΑΠΗΣ

"ΕΠΑΛΗΘΕΥΣΗ ΚΡΥΠΤΟΓΡΑΦΙΚΩΝ ΠΡΩΤΟΚΟΛΛΩΝ ΜΕ ΧΡΗΣΗ ΤΗΣ PROVERIF ΚΑΙ ΤΗΣ VERIFPAL »

Επιβλέπων:
Πέτρος Στεφανέας, Αναπληρωτής Καθηγητής, Σ.Ε.Μ.Φ.Ε.

Τριμελής Εξεταστική Επιτροπή:
Αριστείδης Παγουρτζής, Καθηγητής, Η.Μ.Μ.Υ.
Πέτρος Στεφανέας, Αναπληρωτής Καθηγητής, Σ.Ε.Μ.Φ.Ε.
Πέτρος Ποτίκας, Ε.ΔΙ.Π., Η.Μ.Μ.Υ.

Αθήνα, Σεπτέμβριος 2024

Περίληψη

Στην παρούσα εργασία μελετάμε τυπικές μεθόδους και εργαλεία που χρησιμοποιούνται στην ανάλυση κρυπτογραφικών πρωτοκόλλων. Στο Κεφάλαιο 2 παρουσιάζουμε τον Εφαρμοσμένο Π - Λογισμό, μια κατάλληλη λογική για να μοντελοποιήσει επικοινωνία μέσω καναλιών και να περιγράψει τις διεργασίες που συμβαίνουν σε ένα πρωτόκολλο. Στα επόμενα δύο Κεφάλαια παρουσιάζουμε δύο εργαλεία αυτόματης ανάλυσης που ο σκοπός τους είναι να επαληθεύουν ιδιότητες ασφαλείας ενός πρωτοκόλλου, την ProVerif που βασίζεται θεωρητικά στον Εφαρμοσμένο Π - Λογισμό, και την Verifpal. Στην συνέχεια ασχολούμαστε με την εφαρμογή αυτών των εργαλείων σε ένα πρωτόκολλο που χρησιμοποιείται ευρέως στο Internet, το QUIC Handshake Protocol, καταλήγοντας στα αποτελέσματα και την σύγκρισή τους.

Λέξεις Κλειδιά.

Κρυπτογραφία, Τυπικές Μέθοδοι, Ανάλυση και Επαλήθευση Κρυπτογραφικών Πρωτοκόλλων, Εφαρμοσμένος Π - Λογισμός, ProVerif, Verifpal, QUIC Handshake Protocol

Abstract

In this thesis we study formal methods and tools used in the cryptanalysis. In Chapter 2 we present Applied Π - Calculus, a suitable logic system to model communication through channels and describe processes which occur in a cryptographic protocol. In the following two Chapters we present two automated analysis tools which aim to verify security properties of a protocol. These are Proverif, a tool based on the Applied Π - Calculus, and Verifpal. Then we examine their application in the QUIC Handshake Protocol, a widely used protocol in the Internet, concluding with the results of this analysis and with a comparison between the two tools.

Key Words.

Cryptography, Formal Methods, Analysis and Verification of Cryptographic Protocols, Applied Π - Calculus, ProVerif, Verifpal, QUIC Handshake Protocol

Ευχαριστίες

Ευχαριστώ το μέλος Ε.ΔΙ.Π. κ. Πέτρο Ποτίκα για τις υποδείξεις και διορθώσεις της εργασίας καθώς και για τη γενικότερη επαφή που ξεκίνησε μέσω των μαθημάτων που διδάσκει στην Σ.Ε.Μ.Φ.Ε., η οποία με οδήγησε να ασχοληθώ με σχετικό θέμα διπλωματικής εργασίας. Ευχαριστώ τον επιβλέποντα Καθηγητή κ. Πέτρο Στεφανέα για την συνεργασία και τον Καθηγητή κ. Αριστείδη Παγουρτζή για την συμμετοχή του στην εξεταστική επιτροπή.

.....

Χρίστος Χασάπης

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Η ανάλυση ενός κρυπτογραφικού πρωτοκόλλου ή πρωτοκόλλου ασφαλείας χωρίζεται σε δύο κατηγορίες, στο υπολογιστικό μοντέλο και στο συμβολικό μοντέλο. Στο υπολογιστικό μοντέλο, τα μηνύματα που ανταλλάσσονται είναι ακολουθίες bit (bitstrings) και οι συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης αντιστοιχούν bitstrings σε bitstrings. Ο εχθρός είναι κάποια μη - ντετερμινιστική μηχανή Turing, που μεταβαίνει στις πιθανές καταστάσεις μέσω κάποιας κατανομής πιθανότητας (probabilistic Turing machine). Το υπολογιστικό μοντέλο θεωρείται ότι είναι κοντά στην πραγματικότητα, στις ακριβείς διεργασίες που εκτελούνται σε ένα πρωτόκολλο. Ένα σημαντικό μειονέκτημα είναι ότι οι αποδείξεις ασφαλείας είναι συχνά πολύ περίπλοκες για να αυτοματοποιηθούν.

Από την άλλη, στο συμβολικό μοντέλο ή Dolev - Yao μοντέλο, οι συναρτήσεις κρυπτογράφησης και αποκρυπτογράφησης θεωρούνται "τέλεια μαύρα κουτιά", δηλαδή αφηρημένες οντότητες που δίνουν κάποια έξοδο χωρίς να ορίζουμε τον τρόπο υπολογισμού και χωρίς να είναι ευάλωτες σε επιθέσεις. Τα μηνύματα είναι όροι που κατασκευάζονται από την σύνθεση αρχικών συνόλων ονομάτων και μεταβλητών μαζί με τα συναρτησιακά σύμβολα που ορίζουμε. Επιπλέον, χρησιμοποιούμε μαθηματικές δομές όπως "άλγεβρες διεργασιών", εκφράζοντας την συμπεριφορά του συστήματος με αλγεβρικούς όρους και αξιοποιώντας το θεωρητικό υπόβαθρο μιας τέτοιας δομής για την ανάλυσή μας. Ο εχθρός θεωρείται μια διεργασία που τρέχει παράλληλα με το πρωτόκολλο και μπορεί να ακούει και να στέλνει μηνύματα. Οι δυνατότητες του διαφέρουν με εκείνες ενός εχθρού του υπολογιστικού μοντέλου, για

παράδειγμα δεν μπορεί να σπάσει τις συναρτήσεις κρυπτογράφησης εφόσον θεωρούνται τέλειες και δεν υφίστανται επιθέσεις που να σχετίζονται με το μέγεθος ενός μηνύματος ή κλειδιού κρυπτογράφησης. Γενικά, η κατεύθυνση της ανάλυσης στο συμβολικό μοντέλο είναι να βρεθούν σφάλματα που αφορούν το πρωτόκολλο όπως περιγράφεται και όχι την προγραμματιστική υλοποίησή του.

Προκύπτουν ερωτήματα για την σχέση των δύο μοντέλων, όπως αν μια ιδιότητα ασφαλείας που αποδεικνύεται ότι ισχύει στο ένα μοντέλο ισχύει και στο άλλο. Ή αν βρεθεί μια επίθεση στο ένα μοντέλο να μπορεί να βρεθεί αντίστοιχη και στο άλλο. Συγκεκριμένα αυτά που μπορούμε να πούμε είναι ότι αν μια επίθεση βρεθεί στο συμβολικό μοντέλο τότε απευθείας οδηγεί και σε μία επίθεση στο υπολογιστικό. Από την άλλη, αν μια ιδιότητα ασφαλείας αποδειχθεί στο συμβολικό μοντέλο δεν εξασφαλίζει και την αντίστοιχη ασφάλεια στο υπολογιστικό. Η σύνδεση των δύο μοντέλων είναι αντικείμενο μελέτης, όπου καταγράφονται διάφορα θεωρήματα τα όποια όμως απαιτούν αρκετές παραδοχές λόγω της διαφορετικής φύσης του κάθε μοντέλου.

Το αντικείμενο της μελέτης μας θα είναι το συμβολικό μοντέλο, εξετάζοντας αρχικά τον Εφαρμοσμένο Π - Λογισμό, που κατηγοριοποιείται ως άλγεβρα διεργασιών, και βλέποντας στην συνέχεια την εφαρμογή του συμβολικού μοντέλου μέσω των δύο εργαλείων ProVerif, Verifpal. Εκτός από αυτά τα δύο, έχουν αναπτυχθεί αρκετά εργαλεία (AVISPA, Tamarin, Maude - NPA, FDR, Scyther, κ.α.) και εφαρμοστεί σε ευρεία γκάμα πρωτοκόλλων που αφορούν την καθημερινή μας ζωή, όπως στην ανταλλαγή μηνυμάτων μέσω εφαρμογών, τραπεζικές συναλλαγές, e - commerce, συναλλαγές σε ATM, υπηρεσίες της Google και πολλές άλλες περιπτώσεις. Σημαντικά αποτελέσματα και ευρήματα έχουν οδηγήσει σε αναπροσαρμογή διαδεδομένων πρωτοκόλλων, με χαρακτηριστικό παράδειγμα το Needham - Schroeder public key protocol, όπου θεωρούνταν ασφαλές μέχρι να βρεθεί μια επίθεση μέσω του συμβολικού μοντέλου αρκετά χρόνια μετά την πρώτη δημοσίευσή του, με την επακόλουθη διόρθωσή του [1]. Ένα αντίστοιχο παράδειγμα θα παρουσιάσουμε και εμείς στο Κεφάλαιο 5. Ερευνώντας το αντικείμενο, είναι ξεκάθαρο το εύρος του θεωρητικού υποβάθρου, η πληθώρα εφαρμογών, τα σημαντικά ήδη υπάρχοντα αποτελέσματα και οι προοπτικές που δημιουργούνται με την ανάπτυξη της πληροφορικής. Θα προσπαθήσουμε να ενισχύσουμε αυτούς τους ισχυρισμούς με τα θέματα που παρουσιάζουμε στην παρούσα εργασία.

Κεφάλαιο 2

Εφαρμοσμένος Π-Λογισμός

2.1 Όροι στον Εφαρμοσμένο Π-Λογισμό

Ξεκινάμε θεωρώντας τα αριθμήσιμα άπειρα σύνολα Name, Var και το πεπερασμένο σύνολο Σ. Το Σ ονομάζεται υπογραφή και περιέχει συναρτησιακά σύμβολα. Στο παράδειγμα ενός κρυπτογραφικού πρωτοκόλλου, το Σ μπορεί να περιέχει συναρτησιακά σύμβολα που παραπέμπουν σε συναρτήσεις κρυπτογράφησης δημοσίου κλειδιού, συναρτήσεις ψηφιακής υπογραφής κ.α.. Στοιχεία του Σ με πλήθος ορισμάτων (arity) = 0 ονομάζονται σταθερές. Στο ίδιο παράδειγμα, το σύνολο Name μπορεί να περιέχει μηνύματα, κανάλια επικοινωνίας, δημόσια κλειδιά κ.α.. Το σύνολο Var είναι σύνολο μεταβλητών. Οι έννοιες ονόματα, μεταβλητές και σταθερές έχουν ομοιότητες και θα μπορούσαν να ενωθούν σε μια περιεκτική έννοια, παρ' όλα αυτά ο διαχωρισμός τους είναι χρήσιμος για λόγους απλότητας.

Δεδομένων αυτών των συνόλων σχηματίζουμε τους όρους στον Εφαρμοσμένο Π-Λογισμό.

Ορισμός 2.1.1 *Ορίζουμε το σύνολο Term των όρων στον Εφαρμοσμένο Π-Λογισμό με $M \in Term$, ως εξής:*

$$\mathrm{M} \doteq \begin{cases} \mathrm{x} & \mathrm{x} \in \mathrm{Var} \\ \mathrm{n} & \mathrm{n} \in \mathrm{Name} \\ \mathrm{f(M_1,...M_n)} & \mathrm{f} \in \Sigma,\ \mathrm{M_1,...,M_n} \in \mathrm{Term} \end{cases}$$

Χρησιμοποιούμε για τις μεταβλητές τα x,y,z, για τα ονόματα τα a,b,..,s και για όρους που δηλώνουν είτε όνομα είτε μεταβλητή τα u,v,w. Επίσης δηλώνουμε ως $\overrightarrow{x}, \overrightarrow{a}$ πλειάδες (tuples) των μεταβλητών, ονομάτων αντίστοιχα. Για τους όρους χρησιμοποιούμε τα L,M,N.

2.2 Απλές και Επεκτεταμένες Διεργασίες

Τα σύνολα των απλών και επεκτεταμένων διεργασιών χρησιμοποιούνται για να περιγράψουν κάθε ενέργεια που συμβαίνει σε ένα πρωτόκολλο. Οι συμμετέχοντες ενός πρωτοκόλλου εκτελούν ένα σύνολο ενεργειών, συνεπώς κάθε συμμετέχων είναι μια διεργασία που αποτελείται από αυτό το σύνολο ενεργειών. Για να ορίσουμε τις απλές διεργασίες και τις επεκτεταμένες διεργασίες πρέπει να εισάγουμε μια θεωρεία ισότητας μεταξύ όρων. Αρχικά ορίζουμε την αντικατάσταση σε όρους.

Ορισμός 2.2.1 *Η αντικατάσταση σ είναι μια συνάρτηση: Term → Term και ορίζεται με τον παρακάτω τρόπο:*

i. $\mathrm{x}\sigma = \sigma'(\mathrm{x})$ με $\sigma'(\mathrm{x}) = \mathrm{M}$ αν $\{\mathrm{M}/\mathrm{x}\}$ αλλιώς $\sigma'(\mathrm{x}) = \mathrm{x}$

ii. $\mathrm{n}\sigma = \mathrm{n}$

iii. $\mathrm{f(M_1,...,M_n)}\sigma = \mathrm{f(M_1}\sigma,...,\mathrm{M_n}\sigma)$

(Στο *i.* συμβολίζουμε με {M/x} την αντικατάσταση της μεταβλητής x από τον όρο M.)

Έστω **E** ένα σύνολο από όρους. Το $=_{\mathrm{E}}$ ονομάζεται Θεωρία Ισότητας και έχει στοιχεία της μορφής M=N, όπου M, N ∈ **E**. Έτσι ορίζουμε την σχέση $=_{\mathrm{E}}$ μεταξύ όρων.

Ορισμός 2.2.2 *Ονομάζουμε θεωρία ισότητας από το σύνολο **E** την σχέση $=_E$, η οποία είναι η μικρότερη σχέση ισοδυναμίας μεταξύ όρων που ισχύουν τα εξής:*

i. αν $\mathrm{M} =_{\mathrm{E}} \mathrm{N}$ τότε $\mathrm{M}\sigma =_{\mathrm{E}} \mathrm{N}\sigma$

ii. αν $\mathrm{M}_1 =_{\mathrm{E}} \mathrm{N}_1,...,\mathrm{M}_n =_{\mathrm{E}} \mathrm{N}_n$ τότε $\mathrm{f}(\mathrm{M}_1,...,\mathrm{M}_n) =_{\mathrm{E}} \mathrm{f}(\mathrm{N}_1,...\mathrm{N}_n)$

Η θεωρία ισότητας είναι ο τρόπος για να κατασκευάσουμε κρυπτογραφικά primitives, καθώς δίνουμε νόημα στα συναρτησιακά σύμβολα που παραπέμπουν στα primitives, εξισώνοντάς τα με άλλους όρους.

Τώρα μπορούμε να ορίσουμε το σύνολο **P** των απλών διεργασιών.

Ορισμός 2.2.3 *Ορίζουμε το σύνολο **P** των απλών διεργασιών στον Εφαρμοσμένο Π-Λογισμό (P,Q,R ∈ **P**) ως εξής:*

$$P \doteq \begin{cases} 0 & \text{κενή διεργασία (ουδέτερο στοιχείο)} \\ \mathrm{Q} \mid \mathrm{R} & \text{παραλληλία} \\ !\mathrm{Q} & \text{αντιγραφή} \\ \mathrm{vn.Q} & \text{δημιουργία n και δέσμευση στην Q} \\ \text{if M=N then Q else R} & \text{συνθήκη if...then...else} \\ \mathrm{M(x).Q} & \text{input το x στο κανάλι M και μετά Q} \\ \mathrm{M<N>.Q} & \text{output το N στο κανάλι M και μετά Q} \end{cases}$$

Για να συμπεριλάβουμε αντικαταστάσεις στις διεργασίες, δημιουργούμε το σύνολο των επεκτεταμένων διεργασιών. Ο λόγος που δημιουργούμε αυτή τη νέα έννοια διεργασιών είναι ότι θέλουμε η σύνθεση των αντικαταστάσεων να γίνεται με συγκεκριμένες διεργασίες, όπως φαίνεται και από τον παρακάτω ορισμό.

Ορισμός 2.2.4 *Ορίζουμε το σύνολο **EP** των επεκτεταμένων διεργασιών (A,B,C ∈ **EP**) ως εξής:*

$$A \doteq \begin{cases} \mathrm{P} & \mathrm{P} \in \mathbf{P} \\ \mathrm{B} \mid \mathrm{C} & \text{παραλληλία} \\ \mathrm{vn.B} & \text{δημιουργία ονόματος n και δέσμευση στην B} \\ \mathrm{vx.B} & \text{δημιουργία μεταβλητής x και δέσμευση στην B} \\ \{\mathrm{M/x}\} & \text{αντικατάσταση της x με τον όρο M} \end{cases}$$

Τα ονόματα και οι μεταβλητές έχουν scopes, που οριοθετούνται από τις διεργασίες δέσμευση ονόματος/μεταβλητής και τη διεργασία input. Έτσι έχουμε για μια επεκτεταμένη διεργασία A τα σύνολα fn(A) και fv(A), δηλαδή τα σύνολα των ελεύθερων ονομάτων και ελεύθερων μεταβλητών αντίστοιχα, όπως και bn(A), bv(A) τα σύνολα των δεσμευμένων ονομάτων και δεσμευμένων μεταβλητών. Η αντικατάσταση {M/x} εφαρμόζεται σε κάθε διεργασία που έρχεται σε επαφή. Οι μεταβλητές που μια επεκτεταμένη διεργασία A εκθέτει στο περιβάλλον της, δηλαδή αντικαθίστανται χωρίς να βρίσκονται υπό κάποια δέσμευση, λέμε ότι ανήκουν στο dom(A). Κλειστή επεκτεταμένη διεργασία A ονομάζεται η διεργασία για την οποία ισχύει fv(A) = dom(A), με απλά λόγια όλες οι ελεύθερες μεταβλητές της A αντικαθίστανται.

Για να έχουμε αντικατάσταση σε μια συγκεκριμένη διεργασία P, μπορούμε να γράψουμε την διεργασία vx.({M/x} | P), με την λογική της εντολής let x = M in P, που χρησιμοποιείται στις γλώσσες συναρτησιακού προγραμματισμού (π.χ. ML, Haskell). Με βάση τα παραπάνω, εδώ η μεταβλητή x είναι δεσμευμένη και άρα δεν ανήκει στο dom(vx.({M/x} | P)). Επιπλέον, η {M/x} αφορά αντικατάσταση μιας μεταβλητής. Την αντικατάσταση πολλών μεταβλητών μπορούμε να την γράψουμε με την διεργασία $\{\mathrm{M}_1/\mathrm{x}_1\}$ |...| $\{\mathrm{M}_n/\mathrm{x}_n\}$.

Παρουσιάζουμε τώρα την έννοια του frame μιας διεργασίας, που θα μας χρησιμεύσει παρακάτω στην Σημασιολογία του Εφαρμοσμένου Π - Λογισμού και σε κάποιες ισοδυναμίες διεργασιών.

Ορισμός 2.2.5 *Ένα frame είναι μια επεκτεταμένη διεργασία που δημιουργείται από την σύνθεση των διεργασιών 0, {M/x}, |, νx, νn.*

Συγκρίνοντας τον τρόπο που κατασκευάζεται μια επεκτεταμένη διεργασία και τον τρόπο που κατασκευάζεται ένα frame, διαπιστώνουμε ότι ένα frame είναι μια επεκτεταμένη διεργασία που δεν περιέχει απλές διεργασίες πέραν της διεργασίας **0**. Επακόλουθο αυτού είναι ότι κάθε επεκτεταμένη διεργασία A μπορεί να αντιστοιχηθεί σε ένα frame φ(A) αντικαθιστώντας κάθε απλή διεργασία της A με την διεργασία **0**. Αυτή η αντιστοίχηση δείχνει την στατική γνώση της διεργασίας A, η οποία εκτίθεται στο περιβάλλον της μέσω των αντικαταστάσεων χωρίς να λαμβάνεται υπ' όψιν η δυναμική συμπεριφορά της A.

2.3 Λειτουργική Σημασιολογία

Η λειτουργική σημασιολογία που δίνεται στον Εφαρμοσμένο Π–Λογισμό βασίζεται στο chemical abstract machine, ο ενδιαφερόμενος μπορεί να απευθυνθεί στο [2], ένα μοντέλο που περιγράφει concurrent συστήματα, με την σημασιολογικές σχέσεις και τους κανόνες του να εμπνέονται από χημικές αντιδράσεις.

Στην λειτουργική σημασιολογία του Εφαρμοσμένου Π–Λογισμού, κύριο ρόλο έχουν οι σχέσεις Δομική Ισοδυναμία (structural equivalence) και Εσωτερική Αναγωγή (internal reduction). Όπως μπορεί να καταλάβει κάνεις και από τα ονόματα, η Δομική Ισοδυναμία θέλει να συνδέσει "όμοιες" διεργασίες (χρησιμοποιούμε περιγραφικά την λέξη "όμοιες") και η Εσωτερική Αναγωγή μοντελοποιεί υπολογιστικά βήματα.

Έχοντας μια υπογραφή Σ, μαζί με ένα σύνολο ονομάτων και ένα σύνολο μεταβλητών, την εφοδιάζουμε με μία θεωρία ισότητας από ένα σύνολο όρων

της Σ. Γράφουμε $\Sigma \vdash M = N$ όταν έχουμε $M = N$ στην θεωρία ισότητας και αντίστοιχα $\Sigma \nvdash M = N$ για την άρνηση.

Επιπλέον, ορίζουμε το Evaluation Context ως μια έκφραση με κενό. Το κενό δεν είναι υπό τις διεργασίες αντιγραφή, συνθήκη if...then...else, input και output. Γράφουμε E[_] και το κενό συμπληρώνεται από κάποια διεργασία δίνοντας μια νέα διεργασία. Λέμε ότι το E[_] κλείνει την διεργασία A όταν η διεργασία E[A] είναι κλειστή.

Είμαστε τώρα έτοιμοι να ορίσουμε την σχέση Δομική Ισοδυναμία.

Ορισμός 2.3.1 *Η Δομική Ισοδυναμία (Structural Equivalence)* $\equiv$ *είναι η μικρότερη σχέση ισοδυναμίας μεταξύ επεκτεταμένων διεργασιών, κλειστή με την εφαρμογή του evaluation context, για την οποία ισχύουν οι εξής κανόνες:*

Par 0:	$A \equiv A \mid 0$	
Par A:	$A \mid (B \mid C) \equiv (A \mid B) \mid C$	
Par C:	$A \mid B \equiv B \mid A$	
Repl:	$!P \equiv P \mid !P$	
New 0:	$\nu n.0 \equiv 0$	
New C:	$\nu u.\nu w.A \equiv \nu w.\nu u.A$	
New Par:	$A \mid \nu u.B \equiv \nu u(A \mid B)$	όταν $u \notin fn(A) \cup fv(A)$
Alias:	$\nu x.\{M/x\} \equiv 0$	
Subst:	$\{M/x\} \mid A \equiv \{M/x\} \mid A\{M/x\}$	
Rewrite:	$\{M/x\} \equiv \{N/x\}$	όταν $\Sigma \vdash M{=}N$

Δίνοντας ένα παράδειγμα δομικής ισοδυναμίας, αποδεικνύουμε ότι ισχύει $A\{M/x\} \equiv \nu x.(\{M/x\} \mid A)$, όταν $x \notin fv(M)$.

Παράδειγμα 2.3.1

A{M/x}≡	A{M/x} | 0	από Par 0
	≡ A{M/x} | vx.{M/x}	από Alias
	≡ vx.(A{M/x} | {M/x})	από New Par και x ∉ fv(M)
	≡ vx.({M/x} | A{M/x})	από Par C
	≡ vx.({M/x} | A)	από Subst

Ένα αποτέλεσμα της δομικής ισοδυναμίας είναι ότι για μία κλειστή επεκτεταμένη διεργασία, δηλαδή όλες οι ελεύθερες μεταβλητές της είναι υπό αντικατάσταση, μπορούμε να βρούμε μια δομικά ισοδύναμη που να αποτελείται από μια κλειστή απλή διεργασία P (δηλαδή fv(P)=0) μαζί με αντικατάσταση μεταβλητών και δέσμευση ονομάτων. Συγκεκριμένα:

Αν:
A ∈ **EP**, P∈ **P**, fv(P)=0, fv($\overrightarrow{M}$)=0, $\overrightarrow{n} \subseteq \overrightarrow{X}$
Τότε:
A≡ v.$\overrightarrow{n}$({$\overrightarrow{M}/\overrightarrow{x}$ | P})

Επακόλουθο είναι ότι κάθε κλειστό frame φ μπορεί να γραφεί παρόμοια ως:
φ≡ v.$\overrightarrow{n}$({$\overrightarrow{M}/\overrightarrow{x}$})

Παρουσιάζουμε τώρα την σχέση Εσωτερική Αναγωγή που είναι η κυρία σχέση στην λειτουργική σημασιολογία, καθώς όπως προαναφέραμε μοντελοποιεί τα υπολογιστικά βήματα.

Ορισμός 2.3.2 *Η σχέση Εσωτερική Αναγωγή (Internal Reduction)* $\rightarrow$ *είναι η μικρότερη σχέση μεταξύ επεκτεταμένων διεργασιών, που περιέχει την σχέση* ≡ *και είναι κλειστή ως προς την εφαρμογή του Evaluation Context, για την οποία ισχύουν οι εξής κανόνες:*

Comm: N<x>.P | N(x).Q → P | Q

If: if M=M then P else Q → P

Else: if M=N then P else Q $\rightarrow$ Q όταν $\Sigma \nvdash$ M=N και M,N κλειστοί όροι

Σημείωση: στον κανόνα else χρειαζόμαστε όρους M,N που δεν περιέχουν μεταβλητές και τυχόν αντικαταστάσεις μεταβλητών από όρους πρέπει να εφαρμοστούν πριν τον κανόνα else. Παράδειγμα, στην περίπτωση
{M/x} | if x = M then P else Q,
δεν μπορούμε να εφαρμόσουμε τον κανόνα else και να πάρουμε το {M/x} | Q, καθώς η αντικατάσταση σε παραλληλία με την διεργασία if...then...else δίνει ισοδύναμη διεργασία με if M=M.

Παρακάτω έχουμε ένα χαρακτηριστικό παράδειγμα της Σημασιολογίας της $\rightarrow$, με μια διεργασία που κάνει output τον όρο M και παράλληλα κάνει input την μεταβλητή x στο ίδιο κανάλι. Αναμένουμε ο M να γίνεται input εν τέλει και να αντικαθιστά την x στις διεργασίες που ακολουθούν το input. Συγκεκριμένα αν $x \notin fv(N) \cup fv(M) \cup fv(P)$, αποδεικνύεται ότι:

Παράδειγμα 2.3.2

$$\begin{aligned} N\langle M\rangle.P \mid N(x).Q \equiv \ldots &\equiv vx.(\{M/x\} \mid N\langle x\rangle.P \mid N(x).Q) \\ &\rightarrow vx(\{M/x\} \mid P \mid Q) \qquad \text{(από Com)} \\ &\equiv P \mid Q\{M/x\} \end{aligned}$$

2.4 Περαιτέρω Σημασιολογία

Σε αυτήν την ενότητα εξετάζουμε την Σημασιολογία που οδηγεί σε ελέγχους ασφαλείας κρυπτογραφικών πρωτοκόλλων και χρησιμοποιείται από εργαλεία όπως η ProVerif.

Ισοδύναμα παρατηρήσιμες διεργασίες μεταφράζονται σε ιδιότητες αυθεντικότητας και μυστικότητας. Θέλουμε διαφορετικές δομικά διεργασίες να μην ξεχωρίζουν. Ένα απλό παράδειγμα όπου δύο διεργασίες δεν ξεχωρίζουν είναι το εξής. Έστω ότι A=vx.c$\langle$g(x)$\rangle$ και B=vx.c$\langle$h(x) όπου g,h hash functions. Οι διεργασίες είναι διαφορετικές, αλλά ένας εχθρός που

έχει πρόσβαση στο κανάλι c δεν μπορεί να ξεχωρίζει τις A,B από τα μηνύματα g(x),h(x) που λαμβάνει. Αντίθετα, αν μια διεργασία κάνει output στο κανάλι c και η άλλη όχι τότε οι δύο διεργασίες ξεχωρίζουν.

Με A↓a συμβολίζουμε όταν η διεργασία A μπορεί να στείλει κάποιο μήνυμα στο κανάλι a. Τυπικά αυτό γράφεται ως A $\rightarrow^* \equiv$ E[a<M>.P], για κάποιο E Evaluation Context που δεν δεσμεύει το a. Δηλαδή η A μετά από μια σειρά υπολογιστικών βημάτων θα περιέχει μια διεργασία Output στο κανάλι a.

Ορισμός 2.4.1 *Η σχέση Παρατηρήσιμη Ισοδυναμία (Observational Equivalence)* $\approx$ *είναι η μεγαλύτερη συμμετρική σχέση μεταξύ κλειστών επεκτεταμένων διεργασιών με το ίδιο σύνολο domain, ώστε με A* $\approx$ *B να συνεπάγονται τα εξής:*

i. Αν A↓a τότε B↓a

ii. Αν A $\rightarrow^*$ A′ και A′ κλειστή διεργασία, τότε B $\rightarrow^*$ B′ και A′ $\approx$ B′

iii. E[A] $\approx$ E[B] για κάθε E[_] με E[A],E[B] κλειστές διεργασίες

Στην γενική περίπτωση, το πρόβλημα της Παρατηρήσιμης Ισοδυναμίας είναι μη - αποκρίσιμο λόγω της συνθήκης (iii) που απαιτεί τον έλεγχο για κάθε E[_]. Τα πράγματα μπορούν να γίνουν πιο απλοϊκά στο κομμάτι της απόδειξης, με την σχέση Labelled Bisimilarity $\approx_l$ η οποία αποδεικνύεται ότι ισούται συνολοθεωρητικά με την Παρατηρήσιμη Ισοδυναμία. Ορίζουμε την σχέση $\approx_l$ ξεκινώντας με κάποιες πρωταρχικές έννοιες.

Ορισμός 2.4.2 *Λέμε ότι δύο όροι M,N είναι ίσοι στο frame* φ*, συμβολίζουμε με (M=N)*φ*, αν και μόνο αν:*
$\varphi \equiv \nu\vec{n}.\sigma$ **και** Mσ=Nσ,
με fv(M) ∪ fv(N) ⊆ dom(φ) και $\{\vec{n}\} \cap$ (fn(M) ∪ fn(N)) = 0

Εξηγώντας τον παραπάνω ορισμό, έχουμε ένα frame $\boldsymbol{\varphi}$ που περιέχει μια σειρά αντικαταστάσεων. Οι M,N είναι ίσοι στο φ αν με την αντικατάσταση $\boldsymbol{\sigma}$ που γίνεται στο $\boldsymbol{\varphi}$ (βλέπουμε ότι κάθε ελεύθερη μεταβλητή των M,N αντικαθίσταται λόγω της $\boldsymbol{\sigma}$), οι όροι Mσ, Nσ που προκύπτουν είναι ίσοι.

Ενθυμούμενοι ότι κάθε επεκτεταμένη διεργασία αντιστοιχίζεται σε ένα frame, ορίζουμε την σχέση Στατική Ισοδυναμία (Static Equivalence) $\approx_s$ μεταξύ δύο κλειστών επεκτεταμένων διεργασιών μέσω των frame τους.

Ορισμός 2.4.3 *Έστω A,B κλειστές επεκτεταμένες διεργασίες και φ,ψ τα αντίστοιχα κλειστά frame τους. Λέμε ότι οι A,B είναι Στατικά Ισοδύναμες, $A \approx_s B$, όταν $\varphi \approx_s \psi$. Ισχύει $\varphi \approx_s \psi$ όταν dom(φ)=dom(ψ) και για κάθε όρο M,N έχουμε (M=N)φ αν και μόνο αν (M=N)ψ.*

Ουσιαστικά δύο στατικά ισοδύναμες διεργασίες δεν ξεχωρίζουν όταν συγκρίνουμε τα αποτελέσματα των αντικαταστάσεων τους.

Παράδειγμα 2.4.1 Έστω ότι έχουμε τα 3 παρακάτω frames:
φ_1 = vk,s({k/x} | {s/y})
φ_2 = vk.({f(k)/x} | {g(k)/y})
φ_3 = vk.({k/x} | {f(k)/y})

Τα φ_1, φ_2 είναι στατικά ισοδύναμα διότι οι τιμές k,s είναι ασυσχέτιστες στο φ_1 και οι τιμές f(k),g(k) στο φ_2 φαίνονται ασυσχέτιστες (έστω και αν οι f(k), g(k) βασίζονται στο k). Συνεπώς τα φ_1, φ_2 έχουν την ίδια συμπεριφορά. Αντίθετα στο φ_3, αν χρησιμοποιήσουμε τους όρους f(x),y έχουμε (f(x)=y)φ_3, το οποίο δεν ισχύει για κανένα από τα φ_1, φ_2.

Ο χειρισμός μιας διεργασίας μέσω του frame της μπορεί να απλοποιήσει τον έλεγχο μιας ισοδυναμίας, εδώ συγκεκριμένα της Στατικής Ισοδυναμίας, επειδή δεν εξαρτάται από την δυναμική συμπεριφορά της. Η πολυπλοκότητα του ελέγχου εξαρτάται από την υπογραφή Σ και την θεωρία ισότητας των όρων της.

Καταγράφουμε ορισμένα αποτελέσματα της Στατικής Ισοδυναμίας.

- Έστω A,B κλειστές επεκτεταμένες διεργασίες. Αν $A \equiv B$ ή $A \to B$ τότε $A \approx_s B$.
- Η $\approx_s$ είναι κλειστή ως προς την εφαρμογή του Evaluation Context.
- Έστω φ,ψ frames. Ισχύει $\varphi \approx_s \psi \Leftrightarrow \varphi \approx \psi$.

- Γενικά για διεργασίες ισχύει $\approx \subset \approx_s$.

Επεκτείνουμε την σημασιολογία, εισάγοντας μια αναγωγή με ετικέτα, που συμβολίζεται με $\xrightarrow{\alpha}$ και η ετικέτα α είναι μια από τις δύο παρακάτω μορφές:

$$\alpha = \begin{cases} \text{N(M)} \\ \text{vx.N<x>} \qquad \text{όπου x δεν εμφανίζεται στον όρο N} \end{cases}$$

Επιπλέον, ακολουθούμε τους παρακάτω κανόνες σχετικά με την αναγωγή με ετικέτα.

$$\text{In: } \text{N(x).P} \xrightarrow{\text{N(M)}} \text{P\{M/x\}}$$

$$\text{Out Var: } \frac{\text{x} \notin \text{fv(N<M>.P)}}{\text{N<M>.P} \xrightarrow{\text{vx.N<x>}} P|\{M/x\}}$$

$$\text{Scope: } \frac{A \xrightarrow{\alpha} A', \quad \text{u δεν εμφανίζεται στο } \alpha}{\text{vu.A} \xrightarrow{\alpha} \text{vu.A}'}$$

$$\text{Par: } \frac{A \xrightarrow{\alpha} A', \quad \text{bv}(\alpha) \cap \text{fv(B)}=0}{\text{A} \mid \text{B} \xrightarrow{\alpha} \text{A}' \mid \text{B}}$$

$$\text{Struct: } \frac{\text{A} \equiv \text{B} \quad \text{B} \xrightarrow{\alpha} \text{B}' \quad \text{B}' \equiv \text{A}'}{\text{A} \xrightarrow{\alpha} \text{A}'}$$

Ας δούμε ένα παράδειγμα αναγωγής με ετικέτες για την διεργασία:
vk.a<enc(M,k)>.a<k>.a(z).if z=M then c<M:revealed>

Η συγκεκριμένη διεργασία αρχικά δημιουργεί το όνομα k, το όποιο θα μπορούσε να είναι κάποιο κλειδί, και έπειτα το χρησιμοποιεί για να κρυπτογραφήσει το μήνυμα M και να στείλει το κρυπτογραφημένο μήνυμα μέσω του καναλιού a, ενώ στέλνει και το k μέσω του a. Στην συνέχεια το a κάνει input κάποιον όρο και αν αυτός όρος είναι το M, τότε το M έχει αποκαλυφθεί και λαμβάνουμε το μήνυμα ότι το M αποκαλύφθηκε. Τελικά από τα βήματα της αναγωγής, καταλήγουμε σε μια διεργασία που στέλνει το μήνυμα ότι το M αποκαλύφθηκε. Προφανώς αυτό συμβαίνει γιατί το κλειδί k γνωστοποιείται σε δημόσιο κανάλι και άρα μπορεί να χρησιμοποιηθεί για την αποκρυπτογράφηση του μηνύματος.

Παράδειγμα 2.4.2

$$\begin{aligned}
&\text{vk.a<enc(M,k)>.a<k>.a(z).if z=M then c<M:revealed>}\\
\xrightarrow{vx.a<x>}&\text{vk.\{enc(M,k)/x\} | a<k>.a(z).if z=M then c<M:revealed>}\\
\xrightarrow{vy.a<y>}&\text{vk.\{enc(M,k)/x\} | \{k/y\} | .a(z).if z=M then c<M:revealed>}\\
\xrightarrow{a(dec(x,y))}&\text{vk.\{enc(M,k)/x\} | \{k/y\} | if a(dec(x,y))=M then c<M:revealed>}\\
\rightarrow&\text{vk.\{enc(M,k)/x\} | \{k/y\} | c<M:revealed>}
\end{aligned}$$

(Στις πρώτες δύο αναγωγές χρησιμοποιείται ο κανόνας Out Var, στην 3η αναγωγή χρησιμοποιείται ο κανόνας In και στην τελευταία αναγωγή έχουμε τον κανόνα If της Εσωτερικής Αναγωγής)

Τώρα είμαστε έτοιμη να ορίσουμε την σχέση Labelled Bisimilarity. Συγκριτικά με την Παρατηρήσιμη Ισοδυναμία, γλιτώνουμε το κομμάτι της ισχύς για κάθε Evaluation Context με την εισαγωγή της Στατικής Ισοδυναμίας. Ο ορισμός απαιτεί δύο Labelled Bisimilar διεργασίες να είναι και Στατικά Ισοδύναμες, οπότε ο έλεγχος περνάει στην στατική συμπεριφορά αυτών μέσω των frame τους.

Ορισμός 2.4.4 *Η σχέση Labelled Bisimilarity $\approx_l$ είναι η μεγαλύτερη συμμετρική σχέση μεταξύ κλειστών επεκτεταμένων διεργασιών, έτσι ώστε με $A \approx_l B$ να ισχύουν τα εξής:*

i. $A \approx_s B$

ii. Αν $A \rightarrow A'$ και A' κλειστή διεργασία, τότε $B \rightarrow^* B'$ και $A' \approx_l B'$

iii. Αν $A \xrightarrow{\alpha} A'$, A' κλειστή διεργασία και $fv(\alpha) \subseteq dom(A)$,
τότε $B \rightarrow^* \xrightarrow{\alpha} \rightarrow^* B'$ και $A' \approx_l B'$

Καταλήγουμε με το σημαντικό αποτέλεσμα ως προς την υπολογιστική πολυπλοκότητα που προαναφέραμε, ότι η Παρατηρήσιμη Ισοδυναμία και η Labelled Bisimilarity ταυτίζονται. Η απόδειξη υπάρχει στο [3].

Θεώρημα 2.4.1 *Ισχύει: $\approx = \approx_l$, δηλαδή οι σχέσεις $\approx$, $\approx_l$ εκφράζουν τα ίδια σύνολα.*

2.5 Diffie - Hellman συμφωνία κλειδιού

Ας δούμε ένα αναλυτικό παράδειγμα πρωτοκόλλου στον Εφαρμοσμένο Π - Λογισμό και πως μπορούμε να χρησιμοποιήσουμε την σημασιολογία μας για να προσομοιώσουμε επίθεση πάνω σε αυτό το πρωτόκολλο.

Το πρωτόκολλο Diffie - Hellman επιτρέπει την συμφωνία κλειδιού μεταξύ δύο συμμετεχόντων με ανταλλαγή μηνυμάτων από ένα δημόσιο κανάλι, χωρίς να έχει προϋπάρξει κάποια μυστική επικοινωνία. Το Diffie - Hellman βασίζεται στη θεωρία αριθμών και στις ιδιότητες της συνάρτησης modulo p. Εδώ θεωρούμε τις συναρτήσεις f(x,y), g(x) ως μαύρα κουτιά, με το σκεπτικό ότι η f(x,y) είναι η $y^x mod p$ και η g(x) είναι η $\alpha^x mod p$ με p πρώτο αριθμό και α γεννήτορα της $\mathbb{Z}_p$, ώστε να πάρουμε την εξίσωση:
$(\alpha^y)^x mod p$ = f(x,g(y)) = f(y,g(x)) = $(\alpha^x)^y mod p$

A_1, A_2 : διεργασίες που αντιστοιχούν στους δύο συμμετέχοντες
c_{12}: δημόσιο κανάλι επικοινωνίας από τον A_1 στον A_2
c_{21}: δημόσιο κανάλι επικοινωνίας από τον A_2 στον A_1
n_1: τυχαιότητα που δημιουργεί ο A_1 για να το στείλει ως $g(n_1)$ από το c_{12}
n_2: τυχαιότητα που δημιουργεί ο A_2 για να το στείλει ως $g(n_2)$ από το c_{21}
$\sigma_1 = \{g(n_1)/x_1\}$: αντικατάσταση x_1 με $g(n_1)$
$\sigma_2 = \{g(n_2)/x_2\}$: αντικατάσταση x_2 με $g(n_2)$
$\varphi_1 = \{f(n_1,x_2)/y\}$: αντικατάσταση y με $f(n_1,x_2)$
$\varphi_2 = \{f(n_2,x_1)/y\}$: αντικατάσταση y με $f(n_2,x_1)$

Αρχικά ο A_1 δημιουργεί το n_1, υπολογίζει το $g(n_1)$ και το στέλνει στον A_2 μέσω του c_{12}. Αντίστοιχα κάνει και ο A_2 με το n_2, υπολογίζει το $g(n_2)$ και το στέλνει μέσω του c_{21} στον A_1. Έπειτα ο A_1 υπολογίζει το $f(n_1,g(n_2))$ ενώ ο A_2 υπολογίζει το $f(n_2,g(n_1))$, βγάζοντας και οι δύο το ίδιο αποτέλεσμα που είναι το κοινό κλειδί. Μετά θεωρούμε ότι ο καθένας ακολουθεί τις επόμενες διεργασίες του πρωτοκόλλου που τις ονομάζουμε P_1 για τον A_1 και P_1 για τον A_2. Έτσι μπορούμε να γράψουμε τους A_1, A_2 ως διεργασίες με τον εξής τρόπο:

$A_1 = vn_1.(c_{12}\langle\sigma_1 x_1\rangle \mid c_{21}(x_2).P_1\varphi_1)$
$A_2 = vn_2.(c_{21}\langle\sigma_2 x_2\rangle \mid c_{12}(x_1).P_2\varphi_2)$

Θα χρησιμοποιήσουμε την σημασιολογία με ετικέτες για να δείξουμε πως ένας παθητικός εχθρός C αλληλεπιδρά με την διεργασία $A_1 \mid A_2$ που μοντελοποιεί το πρωτόκολλο. Θεωρούμε ότι ο C έχει πρόσβαση στα κανάλια

c_{12}, c_{21}, μπορεί να διαβάσει τα ανταλλασσόμενα μηνύματα και να τα προωθήσει χωρίς αλλαγές. Ο C μπορεί να γραφεί ως διεργασία με τον εξής τρόπο:

$C = c_{12}(x_1).c_{12}<x_1>.c_{21}(x_2).c_{21}<x_2>.P$

Χρησιμοποιώντας τους κανόνες "In" και "Out Var" από την αναγωγή με ετικέτα, έχουμε το παρακάτω αποτέλεσμα:

$$\begin{aligned}
A_1 \mid A_2 &\xrightarrow{c_{12}<x_1>} vn_1.(\sigma_1 \mid c_{21}(x_2).P_1\varphi_1 \mid A_1) \\
&\xrightarrow{c_{12}(x_1)} vn_1.(\sigma_1 \mid c_{21}(x_2).P_1\varphi_1 \mid vn_2.(c_{21}<\sigma_2 x_2> \mid P_2\varphi_2\ )) \\
&\xrightarrow{c_{21}<x_2>} vn_1.(\sigma_1 \mid c_{21}(x_2).P_1\varphi_1 \mid vn_2.(\sigma_2 \mid P_2\varphi_2\ )) \\
&\xrightarrow{c_{21}(x_2)} vn_1.(\sigma_1 \mid P_1\varphi_1 \mid vn_2.(\sigma_2 \mid P_2\varphi_2\ )) \\
&\equiv ... \equiv vn_1.vn_2.vy.(P_1 \mid P_2 \mid \sigma_1 \mid \sigma_2 \mid \varphi_1) \\
&= vn_1.vn_2.vy.(P_1 \mid P_2 \mid \{g(n_1)/x_1\} \mid \{g(n_2)/x_2\} \mid \{f(n_1,x_2)/y\})
\end{aligned}$$

(Στην δομική ισοδυναμία χρησιμοποιήσαμε τους κανόνες "Alias", "New Par", "Subst" και την ισότητα όρων $f(n_1,g(n_2)) = f(n_2,g(n_1))$.)

Οι αναγωγές $\xrightarrow{c_i(x_j)}$, $\xrightarrow{c_i<x_j>}$ δείχνουν ότι ο C λαμβάνει και στέλνει τα μηνύματα x_j στο κανάλι c_i. Το αποτέλεσμα είναι ότι καταλήξαμε σε μία διεργασία όπου δεν υπάρχει ανταλλαγή άλλων μηνυμάτων έτσι ώστε το κλειδί να γνωστοποιηθεί στον C, ενώ κατά τις αναγωγές ο C λάμβανε και έστελνε μόνο τις τιμές $g(n_1)$, $g(n_2)$. Το πρωτόκολλο συνεχίζεται με τις P_1, P_2 να χρησιμοποιούν το κοινό κλειδί. Αποδεικνύεται ότι η διεργασία που καταλήξαμε είναι παρατηρήσιμα ισοδύναμη με μία διεργασία όπου δημιουργούνται κάποια αρχικά τυχαία ονόματα, ασυσχέτιστα με το κλειδί συνεδρίας που δημιουργείται και τρέχει η συνέχεια του πρωτοκόλλου. Ουσιαστικά έχουμε την καθιέρωση του κοινού κλειδιού χωρίς αυτό να φαίνεται ότι προέκυψε από υπολογιστικά βήματα και επικοινωνία των δύο συμμετεχόντων. Τυπικά, αυτό εκφράζεται ως εξής:

$vn_1.vn_2.vy.(P_1 \mid P_2 \mid \{g(n_1)/x_1\} \mid \{g(n_2)/x_2\} \mid \{f(n_1,x_2)/y\}) \approx$
$vk.\{k/y\}(P_1 \mid P_2) \mid vs_1.\{s_1/x_1\} \mid vs_2.\{s_2/x_2\}$

Κεφάλαιο 3

ProVerif

3.1 Συντακτικό

Η ProVerif είναι ένα εργαλείο που χρησιμοποιείται για την ανάλυση κρυπτογραφικών πρωτοκόλλων στο συμβολικό μοντέλο. Η γλώσσα που χρησιμοποιεί είναι "μια επέκταση του Π - Λογισμού με κρυπτογραφία, παρόμοια με τον Εφαρμοσμένο Π - Λογισμό". Η διαδικασία που ακολουθείται είναι να μεταφράζεται το πρωτόκολλο μας από τη γλώσσα εισαγωγής σε προτάσεις Horn και μέσω διατύπωσης queries να ελέγχονται οι διάφορες ιδιότητες ασφαλείας. Αυτές μπορούν να αφορούν μυστικότητα, επαλήθευση και κάποιες ισοδυναμίες διεργασιών. Σε αυτό το κεφάλαιο θα παρουσιάσουμε κάποια στοιχεία για το συντακτικό και την σημασιολογία της ProVerif, πως σχετίζεται με τον Εφαρμοσμένο Π - Λογισμό, θα περιγράψουμε τον τρόπο που σχηματίζονται οι προτάσεις Horn από το πρωτόκολλο μας και στο τέλος θα δούμε τον αλγόριθμο που χρησιμοποιεί η ProVerif για να ελέγξει αν η ιδιότητα ασφαλείας που θέλουμε να εξετάσουμε παράγεται από τις προτάσεις Horn. Σημειώνουμε ότι η ProVerif είναι ένα εργαλείο που εξελίσσεσαι και υπάρχουν αρκετές εκδόσεις, για αυτό και εμείς θα παρουσιάσουμε κάποια βασικά της στοιχεία.

Ξεκινάμε με το συντακτικό της ProVerif, αναλυτικά στα [4] και [5].

$$\text{M (Όροι)} \doteq \begin{cases} \text{x} & \text{μεταβλητές} \\ \text{n} & \text{ονόματα} \\ \text{f(M}_1\text{,...,M}_n\text{)} & \text{εφαρμογή constructor} \end{cases}$$

$$\text{D (Εκφράσεις)} \doteq \begin{cases} \text{fail} & \text{αποτυχία} \\ \text{M} & \text{όρος} \\ \text{h(D}_1\text{,...,D}_n\text{)} & \text{εφαρμογή constructor ή desrtuctor} \end{cases}$$

$$\text{P,Q (Διεργασίες)} \doteq \begin{cases} 0 & \text{κενή διεργασία} \\ \text{in(M,x);P} & \text{input την x στο κανάλι M} \\ \text{out(M,N);P} & \text{output τον όρο N στο M} \\ \text{P | Q} & \text{παραλληλία} \\ \text{let x=D in P else Q} & \text{αποτίμηση της έκφρασης D} \\ \text{!P} & \text{αντιγραφή} \\ \text{new n;P} & \text{δημιουργία του n στη P} \\ \text{if M=N then P else Q} & \text{συνθήκη if...then...else} \end{cases}$$

Οι όροι αποτελούνται από μεταβλητές, ονόματα και συναρτήσεις, όπως στον Εφαρμοσμένο Π - Λογισμό. Τα ονόματα αντιπροσωπεύουν δεδομένα όπως κλειδιά, κανάλια, τυχαιότητες και άλλα. Επιπλέον περιλαμβάνουμε και τις σταθερές true,false. Το ονόματα και οι μεταβλητές μπορούν να δηλωθούν και με τύπους, ας πούμε μπορούμε να δηλώσουμε ονόματα με τύπο channel ή μεταβλητές με τύπο bitstring. Οι συναρτήσεις που χρησιμοποιούνται στην κατασκευή όρων ονομάζονται constructors.
Εκτός από τις συναρτήσεις constructors συναντάμε και τις συναρτήσεις destructors, ουσιαστικά είναι μερικές συναρτήσεις πάνω σε όρους και ορίζονται από ένα σύνολο rewrite κανόνων. Για να μοντελοποιήσουμε αυτήν την συμπεριφορά, χρησιμοποιούμε το σύνολο των Εκφράσεων. Οι Εκφράσεις αποτιμώνται είτε σε κάποιο όρο είτε σε fail, συνοπτικά λέμε ότι αποτιμώνται σε κάποιο may - fail όρο. Το σύμβολο h στις Εκφράσεις μπορεί να είναι είτε constructor είτε destructor.
Αν g κάποιος destructor με arity n τότε αποτιμούμε το $\text{g(D}_1\text{,...,D}_n\text{)}$ από το σύνολο κανόνων της μορφής $\text{g(U}_1\text{,...}U_n\text{)} \rightarrow \text{U}$ που ορίζει τον g. Ειδικά για

την περίπτωση των κανόνων, τα U_i, U αποτελούνται από όρους χωρίς ονόματα ή από την τιμή fail. Η διαδικασία της αποτίμησης του $g(D_1,...,D_n)$ είναι η εξής. Αποτιμούμε αναδρομικά κάθε D_i σε κάποιον όρο M ή στην τιμή fail και εφόσον κανένα από τα D_i δεν παίρνει την τιμή fail, εξετάζουμε τους rewrite κανόνες που ορίζουν τον g ώστε να βρούμε κάποιον κάνονα και κάποια αντικατάσταση που να συμφωνεί με τις αποτιμήσεις των D_i. Αν δεν υπάρχει τέτοιος κανόνας και τέτοια αντικατάσταση τότε το $g(D_1,...,D_n)$ αποτιμάται σε fail. Αποτίμηση έχουμε και στους constructors, όπου αν f ένας constructor, το $f(D_1,...,D_n)$ αποτιμάται σε fail αν υπάρχει έστω ένα D_i που αποτιμάται σε fail.
Χρησιμοποιώντας τους constructors και τους destructors κατασκευάζουμε διάφορες δομές δεδομένων και κρυπτογραφικά primitives. Δίνουμε μερικά παραδείγματα.

- true, false σταθερές, δηλαδή constuctors με arity 0
  destructor equal: equal(x,x)→true
  equal(x,y)→false, όταν x≠y
- costructor tuple($M_1,...,M_n$), το tuple των όρων $M_1,...,M_n$
  destructor iTerm: iTerm(tuple($x_1,...,x_n$))→x_i,
  η προβολή του i-οστού όρου από ένα tuple
- constructor senc(M,N), η συμμετρική κρυπτογράφηση του μηνύματος M χρησιμοποιώντας το κλειδί N
  destructor sdec: sdec(senc(x,y),y)→x, αποκρυπτογράφηση του senc(x,y)
- constructor aenc(M,pk(sk),r), η πιθανοτική ασύμμετρη κρυπτογράφηση του M, χρησιμοποιώντας τον constructor pk για τη δημιουργία δημοσίου κλειδιού και την τυχαιότητα r
  destructor adec: adec(aenc(x,pk(y),z),y)→x, αποκρυπτογράφηση του aenc(x,pk(y),z)

Σε αρκετές περιπτώσεις κρυπτογραφικών primitives, οι rewrite κανόνες που ορίζουν τους destructors δεν επαρκούν για να τα μοντελοποιήσουν. Ένα παράδειγμα είναι η συμφωνία κλειδιού Diffie - Hellman που παρουσιάσαμε, όπου οι ιδιότητες του εκθετικού modulo δεν μπορούν να περιγραφούν από rewrite κανόνες κάποιου destructor. Σε αυτές τις περιπτώσεις, έχοντας μια θεωρίας ισότητας μεταξύ όρων όπως είδαμε στον Εφαρμοσμένο Π - Λογισμό, οι destructors και constructors ορίζονται μέσω εξισώσεων. Οι εξισώσεις

όμως είναι προσεγγιστικές, κάθως στην πράξη αυτό που συμβαίνει είναι ότι μεταφράζονται σε πεπερασμένο αριθμό rewrite κανόνων. Οπότε έχουμε ένα σύνολο rewrite κανόνων που αφορούν τους constructors και τους destructors και εξάγουμε όρους εφαρμόζοντας αυτούς τους κανόνες, δεδομένης της θεωρίας ισότητας. Ουσιαστικά η διαχείριση constructors και destructors είναι ίδια, σε σχέση με τη διαφοροποίηση των εννοιών που αναφέραμε. Η προσέγγιση έγκειται στο ότι μια πλήρης έννοια της ισότητας με αυτή τη μοντελοποίηση θα απαιτούσε άπειρο αριθμό rewrite κανόνων και για αυτό οι δυνατότητες μας είναι περιορισμένες, ας πούμε το XOR δεν μπορεί να υλοποιηθεί με αυτόν τον τρόπο. Υπάρχουν τροποποιήσεις για να ξεπεραστούν αυτοί οι περιορισμοί, οι οποίες όμως κοστίζουν ως προς την πολυπλοκότητα στην ανάλυση του πρωτοκόλλου, ενώ γενικά η χρήση εξισώσεων αποφεύγεται όταν η περιγραφή μπορεί να γίνει απλούστερα μέσω rewrite κανόνων.

Σχετικά με τις διεργασίες, η λειτουργικότητά τους είναι παρόμοια με τις αντίστοιχες του Εφαρμοσμένου Π - Λογισμού. Ειδικότερα, η διεργασία new n;P δημιουργεί το όνομα n και το δεσμεύει στην P. Το ρόλο του n μπορεί να παίξει κάποιο φρέσκο κλειδί ή κάποια τυχαιότητα. Η διεργασία let x=D in P else Q προσπαθεί να αποτιμήσει την έκφραση D σε κάποιο όρο. Αν η D αποτιμηθεί σε κάποιο όρο M, τότε το x παίρνει την τιμή M και εκτελείται η P. Αλλιώς αν η D αποτιμηθεί σε fail, εκτελείται η Q. Η διεργασία if M=N then P else Q ελέγχει την ισότητα των όρων M,N και εκτελεί την P ή Q ανάλογα με το αποτέλεσμα.

3.2 Σημασιολογία της ProVerif

Είδαμε ότι κάθε έκφραση D αποτιμάται σε έναν may - fail όρο, δηλαδή σε κάποιον όρο ή την τιμή fail. Σχετικά με τους κανόνες που ορίζουν τους destructors, χρησιμοποιήσαμε may - fail όρους που δεν είχαν ονόματα. Αντίθετα τώρα θα χρησιμοποιήσουμε κλειστούς may - fail όρους, δηλαδή ορούς που δεν έχουν μεταβλητές. Χρησιμοποιούμε για αυτούς τους όρους το σύμβολο V για να τους ξεχωρίσουμε από τους αντίστοιχους που είχαν μόνο μεταβλητές και χρησιμοποιήσαμε το σύμβολο U. Θέλουμε να δούμε πότε μια κλειστή έκφραση D αποτιμάται σε ένα κλειστό may - fail όρο το οποίο και συμβολίζουμε ως D⇓V. Στον ορισμό της σχέσης ⇓ χρησιμοποιούμε rewrite κανόνες, οπότε επειδή μια έκφραση D συντίθεται από constructors και destructors, πρέπει να αντιστοιχήσουμε σε κάθε constructor κάποιον rewrite

κανόνα. Έτσι για κάθε constructor f(M_1,...,M_n) αντιστοιχούμε τον ταυτοτικό κανόνα f(M_1,...,M_n)→f(M_1,...,M_n). Ορίζουμε τυπικά την σχέση ⇓, μεταξύ μιας κλειστής έκφρασης D και ενός κλειστού may - fail όρου V.

Ορισμός 3.2.1 *Μια κλειστή έκφραση D αποτιμάται σε κάποιον κλειστό may - fail όρο V, D ⇓ V, με τον παρακάτω τρόπο:*

i. M ⇓ M, M κλειστός όρος

ii. fail ⇓ fail

iii. h(D_1,...,D_n) ⇓ V αν και μόνο αν D_1 ⇓ V_1,...,D_n ⇓ V_n και υπάρχει κάποιος rewrite κανόνας h(U_1,...,U_n)→U και αντικατάσταση σ ώστε σU_1=V_1,...,σU_n=V_n

Τώρα χρειαζόμαστε να επεκτείνουμε την σημασιολογία μας και στις διεργασίες. Για τη βάση μας θα χρησιμοποιήσουμε δύο δυναμικά σύνολα που διαμορφώνονται κατά την εκτέλεση του πρωτοκόλλου. Το πρώτο είναι το ζεύγος **E**=(N_{pub},N_{priv}) που περιέχει τα ελεύθερα ονόματα που είναι δημόσια και τα ελεύθερα ονόματα που είναι ιδιωτικά και ονομάζεται περιβάλλον (environment). Το δεύτερο σύνολο **P** περιέχει τις κλειστές διεργασίες που τρέχουν στο πρωτόκολλο. Το ζεύγος (**E**,**P**) ονομάζεται semantic configuration και πάνω σε αυτό θα ορίσουμε μια σχέση αναγωγής όπου ουσιαστικά θα δείχνει πως μεταβάλλεται το σύνολο (**E**,**P**) όταν σε αυτό προστίθεται κάθε πιθανή διεργασία. Το σκεπτικό βασίζεται στις έννοιες Δομική Ισοδυναμία και Εσωτερική Αναγωγή του Εφαρμοσμένου Π - Λογισμού, έχοντας παρόμοιας λογικής κανόνες τροποποιημένους στο συγκεκριμένο πλαίσιο. Οι κανόνες που ορίζουμε για την αναγωγή → πάνω στα semantic configuration είναι οι εξής:

Nil: (**E**,**P** ∪ {0})→(**E**,**P**)

Par: (**E**,**P** ∪ {P | Q} →(**E**,**P** ∪ {P,Q})

Repl: (**E**,**P** ∪ {!P} →(**E**,**P** ∪ {P,!P})

Res: ((N_{pub},N_{priv},),**P** ∪ {new a;P})→((N_{pub},N_{priv}∪ a',),**P** ∪ {P{a′/a}}),
με a′ ∉ (N_{pub} ∪ N_{priv})

I/O: (**E**,**P** ∪ {out(N,M);P;in(N,x);Q} →(**E**,**P** ∪ {P,Q{M/x}})

Eval 1: (**E**,**P** ∪ {let x=D in P else Q} →(**E**,**P** ∪ {P{M/x}}),
εάν D ⇓ M για κάποιον όρο M

Eval 2: (**E**,**P** ∪ {let x=D in P else Q} →(**E**,**P** ∪ {Q}),
εάν D ⇓ fail

Cond 1: (**E**,**P** ∪ {if M=N then P else Q} →(**E**,**P** ∪ {P}),
εάν ισχύει M=N

Cond 2: (**E**,**P** ∪ {if M=N then P else Q} →(**E**,**P** ∪ {Q}),
εάν ισχύει M≠N

Όπως είπαμε, για κάθε πιθανή διεργασία ορίζουμε και έναν κανόνα. Αν εξετάσουμε κάθε διεργασία και το αποτέλεσμα της αναγωγής της όταν αυτή προστίθεται στο σύνολο **P** του (**E,P**), παρατηρούμε ότι μπορούμε να πάρουμε το ίδιο αποτέλεσμα Εσωτερικής Αναγωγής και για κάθε αντίστοιχη διεργασία του Εφαρμοσμένου Π - Λογισμού μέσω μιας σειράς βημάτων εφαρμογής των σχέσεων ≡ και →. Επιπλέον αναφέρουμε χωρίς να το αναπτύξουμε ότι ορίζεται αντίστοιχα στην ProVerif η Παρατηρήσιμη Ισοδυναμία πάνω στα semantic configurations ([5]).

Κλείνουμε αυτό το κομμάτι παρουσιάζοντας μερικές συμπληρωματικές έννοιες.

Ορίζουμε τυπικά την Μυστικότητα ενός όρου, δηλαδή πότε ένας όρος δεν γνωστοποιείται στον εχθρό. Αρχικά πρέπει να ορίσουμε την έννοια του εχθρού, ακολουθώντας το μοντέλο Dolev-Yao. Έχοντας το σύνολο των δημοσίων ονομάτων N_{pub}, μια κλειστή διεργασία Q αποτελεί εχθρό (συγκεκριμένα N_{pub} - εχθρό, αφού εξαρτάται από το N_{pub}) αν και μόνο αν fn(Q) ⊆ N_{pub} και κάθε συναρτησιακό σύμβολο που εμφανίζεται στην Q είναι δημόσιο. Μια ακόμη έννοια που χρειαζόμαστε είναι το Ίχνος (trace), το οποίο είναι μια ακολουθία από αναγωγές στα semantic configurations. Ορίζουμε πότε ένας όρος M δημοσιοποιείται σε κάποιο ίχνος tr.

Ορισμός 3.2.2 *Λέμε ότι το ίχνος* $tr = (\boldsymbol{E}_0,\boldsymbol{P}_0)\rightarrow^*(\boldsymbol{E}_f,\boldsymbol{P}_f)$ *δημοσιοποιεί τον όρο M αν και μόνο αν το tr περιέχει κάποια αναγωγή*
*(**E**,**P** ∪ {out(N,M);P, in(N,x);Q} →(**E**,**P** ∪ {P,Q{M/x}}), όπου ο όρος N (αντιπροσωπεύει κάποιο κανάλι) ανήκει στο* N_{pub} *του* $\boldsymbol{E}_0$.

Τώρα μπορούμε να ορίσουμε την Μυστικότητα ενός όρου.

Ορισμός 3.2.3 *Η κλειστή διεργασία P κρατάει τον όρο M μυστικό από το σύνολο N_{pub} αν και μόνο αν για κάποιο περιβάλλον (N_{pub},N_{priv}) τέτοιο ώστε fn(M) ∪ fn(P) ⊆ N_{pub} ∪ N_{priv}, για κάθε N_{pub} - εχθρό Q και για κάθε ίχνος tr=((*N_{pub},N_{priv}),{P,Q})→*($\boldsymbol{E}_f$,$\boldsymbol{P}_f$), το tr δεν δημοσιοποιεί τον όρο M.*

Για να μοντελοποιήσουμε ιδιότητες αυθεντικοποίησης και αντιστοιχίας χρησιμοποιούμε την έννοια του "γεγονότος". Επεκτείνουμε το συντακτικό μας με τη διεργασία event e(M);P, όπου το e(M) εκτελείται σιωπηλά και ακολουθεί η P. Για αυτό προσθέτουμε και στην σημασιολογία τον κανόνα (**E**,**P** ∪ {e(M);P})→(**E**,**P** ∪ {P}).
Το event e(M) δεν επηρεάζει πρακτικά το πρωτόκολλο και λειτουργεί ως "σήμανση" κάποιου συγκεκριμένου σημείου του πρωτοκόλλου. Όταν εκτελείται το e(M) ο όρος M δεν γνωστοποιείται στον εχθρό. Χρησιμοποιώντας τα γεγονότα μπορούν να τεθούν ερωτήματα αντιστοιχίας όπως "εάν εκτελείται το γεγονός e_2(M), τότε το γεγονός e_1(N) έχει εκτελεστεί πριν", το οποίο γράφεται ως
query event (e_2(M)) $\Longrightarrow$ event (e_1(N)).
Συγκεκριμένα έχουμε ότι για κάθε εκτέλεση του e_2(M) προηγείται εκτέλεση του e_1(N). Αυτό κρύβει το ότι το e_1(N) μπορεί να αντιστοιχίζεται σε παραπάνω από μία εκτελέσεις του e_2(M). Για να το αποφύγουμε αυτό, θέτουμε το ερώτημα
query inj - event (e_2(M)) $\Longrightarrow$ inj - event (e_1(N))
το οποίο αντιστοιχεί το e_1(N) σε μία μόνο εμφάνιση του e_2(M)

3.3 Μετάφραση ενός Πρωτοκόλλου σε Προτάσεις Horn

Μια πρόταση Horn έχει την μορφή $F_1 \wedge ... \wedge F_n \Longrightarrow F_0$. Τα F_i είναι κατηγορήματα, pred(t_1,...,t_n) και ονομάζονται γεγονότα. Οι παράμετροι των κατηγορημάτων είναι όροι δηλαδή ονόματα, μεταβλητές και συναρτήσεις, ενώ μπορούμε να έχουμε και την τιμή fail. Ιδιαίτερα όμως, τα ονόματα θεωρούνται ως μια συνάρτηση που εξαρτάται από όρους n[t_1,...,t_n] (χρησιμοποιούμε τα [ ] για να τα ξεχωρίσουμε από τις συναρτήσεις f(t_1,...,t_n) των όρων). Κατά τη διάρκεια

ενός πρωτοκόλλου παράγονται φρέσκα ονόματα από τους συμμετέχοντες τα όποια διαφέρουν σε κάθε εκτέλεση του πρωτοκόλλου και εξαρτώνται από την ανταλλαγή μηνυμάτων που λήφθηκαν πριν από τη δημιουργία τους. Οπότε με τη θεώρηση των ονομάτων ως συναρτήσεις όρων μπορούμε να μοντελοποιήσουμε την ιδιότητα του φρέσκου ονόματος.

Μπορούμε να έχουμε διάφορα γεγονότα. Αυτό που θα χρησιμοποιήσουμε στην ανάλυση μας είναι το γεγονός attacker(t), που ερμηνεύεται ως "ο εχθρός μπορεί να έχει τον όρο t στην κατοχή του". Η λέξη "μπορεί" έχει να κάνει με τις προσεγγίσεις που γίνονται κατά τη μετάφραση του πρωτοκόλλου σε προτάσεις Horn και θα αναφερθούμε πιο συγκεκριμένα παρακάτω. Άλλα κατηγορήματα που υπάρχουν είναι το mess(M, C), το οποίο δηλώνει ότι το μήνυμα M ίσως εμφανίζεται στο κανάλι C (τα M, C είναι όροι προφανώς), το κατηγόρημα event(e) και άλλα.

Με το βασικό μας κατηγόρημα το attacker(t) θέλουμε να δούμε ποιες προτάσεις Horn σχηματίζονται από την μετάφραση του πρωτοκόλλου με αυτό το κατηγόρημα, με άλλα λόγια ποια είναι η αρχική γνώση του εχθρού.

- Σε κάθε πρωτόκολλο θεωρούμε ότι ο εχθρός μπορεί να δημιουργήσει τουλάχιστον ένα όνομα n[ ], οπότε δημιουργείται το κατηγόρημα attacker(n[ ]). (Χρησιμοποιούμε πάλι τα [ ] για να δείξουμε ότι το n[ ] είναι ένα φρέσκο όνομα, διαφέρει σε κάθε εκτέλεση του πρωτοκόλλου και εξαρτάται από τα μηνύματα που ανταλλάσσονται.) Επιπλέον, για κάθε όνομα a που δηλώνεται ως free στο πρωτόκολλο μας, δημιουργείται το γεγονός attacker(a[ ]). Τετριμμένα έχουμε και το γεγονός attacker(fail).

- Ο εχθρός μπορεί να χρησιμοποιήσει κάθε κανάλι που γνωρίζει είτε για να ακούσει τα μηνύματα που μεταδίδονται σε αυτό είτε για να στείλει μηνύματα μέσω αυτού. Αυτές οι δυνατότητες γράφονται αντίστοιχα με τις παρακάτω δύο προτάσεις:
  $\text{mess}(x,y) \wedge \text{attacker}(x) \Longrightarrow \text{attacker}(y)$
  $\text{attacker}(x) \wedge \text{attacker}(y) \Longrightarrow \text{mess}(x,y)$
  (Αν το x είναι δημόσιο τότε το mess(x,y) αντικαθίσταται με το attacker(y).)

- Στις υπολογιστικές δυνατότητες του εχθρού είναι ο κάθε constructor που χρησιμοποιούμε στο πρωτόκολλο. Για κάθε f constructor σχηματίζεται η πρόταση:

attacker(x_1) ∧ ... ∧ attacker(x_n) ⟹ attacker(f(x_1,...,x_n))

π.χ. attacker(sk) ⟹ attacker(pk(sk))
attacker(m) ∧ attacker(pk(sk)) ⟹ attaker(aenc(m,pk(sk)))

- Επίσης κάθε destructor g είναι στις υπολογιστικές δυνατότητες του εχθρού. Κάθε destructor περιγράφεται από κάποιο σύνολο rewrite κανόνων g(U_1,...,U_n) → U. Για κάθε τέτοιο κανόνα έχουμε την αντίστοιχη πρόταση:
attacker(U_1) ∧ ... ∧ attacker(U_n) ⟹ attacker(U)

  π.χ. attacker(aenc(m,pk(sk))) ∧ attacker(pk(sk)) ⟹ attacker(m) (αποκρυπτογράφηση της συνάρτησης aenc).

Εκτός από τις παραπάνω περιπτώσεις, στην αρχική γνώση του εχθρού έχουμε και προτάσεις που σχηματίζονται από το ίδιο το πρωτόκολλο, δηλαδή από τις διεργασίες του. Κάθε διεργασία P μεταφράζεται σε ένα σύνολο προτάσεων Horn και διαδικασία της μετάφρασης περιγράφεται αναλυτικά στο [5]. Επιγραμματικά, οι προτάσεις που σχηματίζονται σχετικά με τη γνώση του εχθρού προέρχονται από κάποια διεργασία output και οι όροι που περιέχονται εξαρτιόνται από τις υπόλοιπες διεργασίες. Ας δούμε ένα ενδεικτικό παράδειγμα, μια απλοποιημένη εκδοχή του Denning-Saco distribution key protocol.

Έστω ότι έχουμε τους συμμετέχοντες A,B. Ο A δημιουργεί ένα φρέσκο κλειδί k, το υπογράφει με το ιδιωτικό του κλειδί υπογραφής ssk_a και το στέλνει κρυπτογραφημένο με το δημόσιο κλειδί $pk_b(sk_b)$ του B, στον B. Ο B το αποκρυπτογραφεί μέσω του ιδιωτικού του κλειδιού sk_b, λαμβάνει το k και βλέποντας την υπογραφή υποθέτει ότι το έλαβε από τον A. Έπειτα στέλνει το μυστικό s κρυπτογραφημένο με το κλειδί k στον A, το οποίο και αποκρυπτογραφεί μέσω του k. Αυτή η διεργασία γράφεται στην ProVerif όπως παρακάτω. Το sska είναι το ιδιωτικό κλειδί υπογραφής του A, το skb είναι το ιδιωτικό κλειδί του B, το c είναι ένα ελεύθερο κανάλι επικοινωνίας, τα spka,pkb είναι τα δημόσια κλειδιά των A,B που δίνονται από τον constructor pk(x), το k είναι το κλειδί που δημιουργεί ο A. Επιπλέον χρησιμοποιούμε τον constructor aenc(x,y) για την ασύμμετρη κρυπτογράφηση και τον αντίστοιχο destructor adec(x,y), τον costructor sign(x,y) για την υπογραφή του A και τον αντίστοιχο destructor check(x,y) για τον έλεγχο της υπογραφής από τον B, τον constructor senc(x,y) για την συμμετρική κρυπτογράφηση και τον αντίστοιχο destructor sdec.

P_0 = new sska;new skb; let spka=pk(sska) in let pkb=pk(skb) in

out(c,spka); out(c,pkb); (Pa(sska,pkb) | Pb(skb,spka))
Pa(sska,pkb) = ! new k; out(c,aenc(sign(k,sska),pkb)); in(c,x); let z=sdec(x,k) in 0
Pb(skb,spka) = ! in(c,y); let w=adec(y,skb) in let xk=check(w,spka) in out(c,senc(s,xk)

Τα ελεύθερα ονόματα που δηλώνονται στην αρχή του script είναι το κανάλι c και το μυστικό s. Το κανάλι c το δηλώνουμε ως δημόσιο ενώ το s το δηλώνουμε ως ιδιωτικό Τα υπόλοιπα ονόματα είναι υπό κάποιο περιορισμό αρά δεν ανήκουν στα ελεύθερα ονόματα. Συνεπώς το αρχικό μας περιβάλλον είναι το σύνολο $\mathbf{E}$=(N_{pub},N_{priv})=({c},{s}). Σημειώνουμε επιπλέον ότι το κατηγόρημα mess(c,m) αντικαθίσταται με το κατηγόρημα attacker(m) εφόσον το κανάλι c είναι δημόσιο. Έτσι, οι πρώτες προτάσεις Horn που σχηματίζονται από το παραπάνω πρωτόκολλο που γράψαμε στην ProVerif είναι οι εξής:

1. attacker(pk(sska))
2. attacker(pk(skb))
3. attacker(x) ⇒ attacker(aenc(sign(k,sska),x))
4. attacker(aenc(sign(x,sska),pk(skb))) ⇒ attacker(senc(s,x))

Οι προτάσεις (1), (2) αφορούν την διεργασίες out(c,spka), out(c,pkb) στην διεργασία P_0. Η πρόταση (3) αφορά τη διεργασία out(c,aenc(sign(k,sska),x)) της P_a και το ότι ο εχθρός μπορεί να στείλει ένα οποιοδήποτε όνομα που να παίξει το ρόλο του pkb, που είναι input στην διεργασία P_a και να πάρει το μήνυμα aenc(sign(k,sska),x) κρυπτογραφημένο με το όνομα που έχει δώσει ο εχθρός. Η πρόταση (4) αφορά τη διεργασία out(c,senc(s,xk)) της P_b και το ότι ο εχθρός μπορεί να στείλει το μήνυμα aenc(sign(x,sska),pk(skb)), το όποιο περιμένει η P_b, με ένα δικό του κλειδί και καθώς οι όροι για την αποκρυπτογράφηση και την υπογραφή είναι σωστοί, η P_b στέλνει το senc(s,x) στο δημόσιο κανάλι c και το λαμβάνει ο εχθρός.

Αναφέραμε πιο πριν ότι γίνονται κάποιες προσεγγίσεις όταν μεταφράζεται ένα πρωτόκολλο σε προτάσεις Horn. Μια βασική προσέγγιση είναι ότι

αγνοείται ο αριθμός των επαναλήψεων σε κάποιο βήμα του πρωτοκόλλου και μπορεί να δημιουργούνται παραπάνω της μίας προτάσης Horn για αυτό το βήμα, όπως και να συνεχίζονται να παράγονται προτάσεις Horn για αυτό το βήμα ακόμα και αν έχουν ολοκληρωθεί επόμενα βήματα του πρωτοκόλλου. Ενδεικτικά, μπορεί σε ένα βήμα ο συμμετέχων A να στέλνει το m_1 ή το m_2, άρα να μην στέλνει ποτέ και τα δύο. και από την άλλη να δημιουργούνται δύο προτάσεις Horn έτσι ώστε ο A φαίνεται ότι έχει στείλει εν τέλει και το m_1 και το m_2. Αυτό μπορεί να οδηγήσει σε false attacks, που είναι και ο λόγος που το κατηγόρημα attacker(t) το ερμηνεύουμε ως ο εχθρός μπορεί να γνωρίζει το t. Παρ΄ όλα αυτά, το σημαντικό είναι ότι καλύπτουμε όλα τα μηνύματα που γίνονται γνωστά στον εχθρό, επομένως αποκλείεται η περίπτωση να θεωρηθεί ένα μήνυμα μυστικό που στην πραγματικότητα αποκαλύπτεται στον εχθρό. Επιπλέον, η μετάφραση σε προτάσεις Horn μας επιτρέπουν να χειριστούμε μεγάλα και περίπλοκα πρωτόκολλα, ενώ γενικά όπως αποδεικνύεται από πειράματα [6], οι περιπτώσεις να έχουμε false attacks είναι σπάνιες.

3.4 Αλγόριθμος Resolution

Ο αλγόριθμος Resolution της ProVerif είναι ο τρόπος που επιλέγονται οι αρχικές προτάσεις Horn για να εφαρμόσουμε τον λογικό κανόνα resolution με σκοπό να πάρουμε ως απάντηση την ύπαρξη ή όχι του επιθυμητού γεγονότος. Ένας όρος παραμένει μυστικός όταν το γεγονός που το συνδέουμε (π.χ. ο όρος s με το γεγονός attacker(s)) δεν παράγεται από τις προτάσεις που σχηματίζονται στον αλγόριθμο Resolution. Παρακάτω θα δούμε τυπικά τι σημαίνει ένα γεγονός να παράγεται από τις προτάσεις.

Χρησιμοποιούμε τα παρακάτω σύμβολα:

R: πρόταση
F: γεγονός
H: υπόθεση (γεγονότα με διάζευξη)
C: συμπέρασμα (γεγονός)
π.χ. $R \doteq (H \implies C) \doteq (F_1 \wedge ... \wedge F_n = F_0)$

Ορισμός 3.4.1 *Λέμε ότι η πρόταση* $H_1 \implies C_1$ *περιλαμβάνει την πρόταση* $H_2 \implies C_2$*, συμβολίζουμε με* $(H_1 \implies C_1) \sqsupseteq (H_2 \implies C_2)$ *αν και μόνο αν υπάρχει αντικατάσταση* $\boldsymbol{\sigma}$ *ώστε* $\boldsymbol{\sigma}H_1 \subseteq H_2$ *και* $\boldsymbol{\sigma}C_1 = C_2$

Αυτό που λέει ο παραπάνω ορισμός είναι ότι αν έχουμε $R_1 \sqsupseteq R_2$ μπορούμε να πάρουμε την R_2 από την R_1 με αντικατάσταση. Κατά συνέπεια μπορούμε να εξάγουμε κάθε συμπέρασμα της R_2 από την R_1, με τις ίδιες ή και λιγότερες υποθέσεις.

Ορισμός 3.4.2 *Έστω F ένα κλειστό γεγονός, δηλαδή ένα γεγονός χωρίς μεταβλητές, και* $\boldsymbol{R}$ *ένα σύνολο προτάσεων. Λέμε ότι το F παράγεται από το* $\boldsymbol{R}$ *αν και μόνο αν υπάρχει ένα πεπερασμένο κατευθυνόμενο δένδρο με την παρακάτω μορφή:*

i. Κάθε κόμβος εκτός της ρίζας έχει ως ετικέτα μια πρόταση R από το σύνολο $\mathbf{R}$

ii. Κάθε ακμή έχει ως ετικέτα ένα κλειστό γεγονός F′

iii. Αν ένας κόμβος ετικέτας R έχει μια εισερχόμενη ακμή ετικέτας F_0 και k εξερχόμενες ακμές ετικέτας F_1,...,F_k, τότε ισχύει
$R \sqsupseteq (F_1,...,F_k \implies F_0)$

iv. Η ρίζα έχει μόνο ένα γιό και η εξερχόμενη ακμή από τη ρίζα στον γιό έχει την ετικέτα F που αντιστοιχεί στο παραγόμενο γεγονός.

Ενδεικτικά, η μορφή ενός δένδρου που δείχνει την παραγωγή του κλειστού γεγονότος F από το σύνολο προτάσεων $\mathbf{R}$.

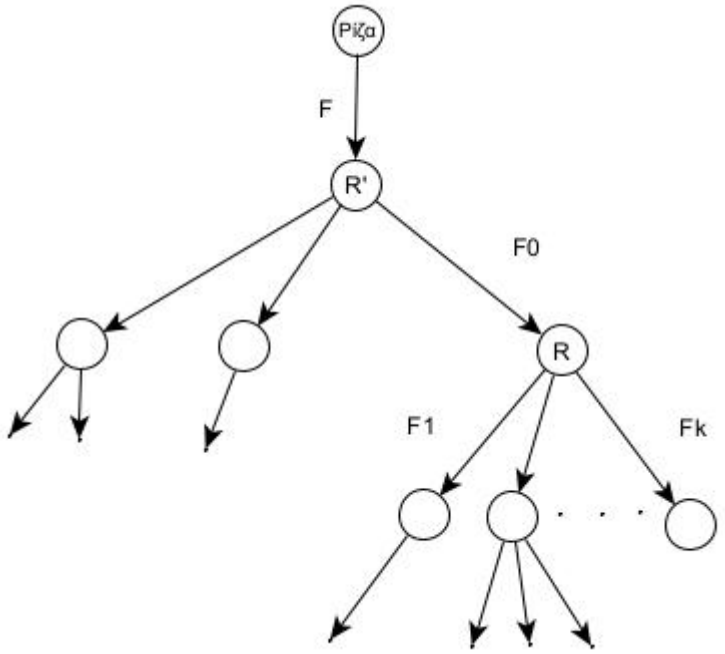

Figure 3.1: Παραγωγή F από **R**

Από το σύνολο των προτάσεων Horn που περιγράφουν το πρωτόκολλο οδηγούμαστε μέσω του λογικού κανόνα resolution σε σχηματισμούς νέων προτάσεων με στόχο να δούμε αν παράγεται το επιθυμητό γεγονός. Όμως μια τέτοια ανεξέλεγκτη τακτική θα κάνει τον έλεγχο μας να μην τερματίζει. Πρέπει λοιπόν να κάνουμε μια επιλογή στις προτάσεις που θα χρησιμοποιήσουμε για να σχηματίσουμε νέες προτάσεις, αυτή η επιλογή να μειώνει ανά βήμα τον αριθμό των προτάσεων που ελέγχουμε και στο τέλος τα αποτελέσματα μας να είναι αυτά που θα εξαγάγαμε με το αν ακολουθούσαμε μια ανεξέλεγκτη τακτική.

Λέμε ότι δύο γεγονότα F_1, F_2 είναι ενοποιήσιμα (unifiable) όταν υπάρχει αντικατάσταση σ έτσι ώστε $\sigma F_1 = \sigma F_2$, η σ αποκαλείται unifier. Αν σ,τ είναι δύο unifiers των F_1, F_2, ο σ είναι πιο γενικός unifier από τον τ αν υπάρχει αντικατάσταση υ ώστε $\sigma\upsilon = \tau$. Γράφουμε ως MGU τον πιο γενικό unifier.

Ορισμός 3.4.3 *Έστω $R \doteq (H \Longrightarrow C)$, $R' \doteq (H' \Longrightarrow C')$ και έστω ότι υπάρχει $F_0 \in H'$ και αντικατάσταση $\boldsymbol{\sigma}$, ώστε $\boldsymbol{\sigma}C = \boldsymbol{\sigma}F_0$ και η $\boldsymbol{\sigma}$ να είναι ο MGU. Το αποτέλεσμα του κανόνα Resolution μεταξύ των προτάσεων R, R΄ πάνω στο F_0, συμβολίζουμε με $R \circ_{F_0} R'$, ορίζεται ως εξής:*

$$\frac{\mathrm{R} \doteq \mathrm{H} \implies \mathrm{C} \qquad \mathrm{R}' \doteq \mathrm{H}' \implies \mathrm{C}'}{\mathrm{R} \circ_{F_0} \mathrm{R}' \doteq \sigma(\mathrm{H} \cup (\mathrm{H}' \setminus \mathrm{F}_0)) \implies \sigma\mathrm{C}'}$$

Ας δούμε πως κατευθύνουμε την επιλογή προτάσεων για Resolution. Αν έχουμε μια πρόταση $\mathrm{R} \circ_{F_0} \mathrm{R}'$, θέλουμε να εφαρμόζουμε τον κανόνα πάνω στα γεγονότα F_0 που δεν είναι της μορφής attacker(x) όπου x οποιαδήποτε μεταβλητή. Το γεγονός attacker(x) είναι ενοποιήσιμο με κάθε άλλος γεγονός της μορφής attacker(p), όπου p γενικός όρος, οπότε αν επιλέγουμε προτάσεις R′ με το attacker(x) στις υποθέσεις μπορεί είτε να καταλήξουμε σε περιττές ενοποιήσεις που καθυστερούν τον αλγόριθμο είτε χειρότερα σε κάποιο loop εφαρμογής του κανόνα. Οπότε η πρώτη μας κατεύθυνση είναι να μην επιλέγουμε αυτές τις προτάσεις ως R′.
Από την άλλη για την R θέλουμε στις υποθέσεις να έχουμε μόνο γεγονότα της μορφής attacker(x) ή να μην έχουμε καθόλου υποθέσεις. Αυτό γίνεται γιατί με το να έχουμε τέτοια γεγονότα ως υποθέσεις ουσιαστικά περιλαμβάνουμε κάθε αντικατάσταση της μεταβλητής x στο γεγονός attacker(x) που θα παραχθεί από την ανάλυση μας, όπως θα δούμε και παρακάτω στην ανάλυση του αλγορίθμου. Επιπλέον, συνήθως όταν έχουμε υποθέσεις της μορφής attacker(x), το συμπέρασμα δεν θα είναι και αυτό αντίστοιχα κάποιο attacker(x), που θα οδηγούσε πάλι σε αρκετές ενοποιήσεις. Βεβαία αυτή η τακτική δεν μας εξασφαλίζει γενικά τον τερματισμό, έτσι και αλλιώς η ProVerif δεν υπόσχεται τον τερματισμό στην ανάλυση κάθε πρωτοκόλλου. Παρ' όλα αυτά είναι μια καλή τακτική για να τερματίζει σε αρκετές περιπτώσεις, ενώ ενδεικτικά τερματίζει πάντα στα επονομαζόμενα πρωτόκολλα με ετικέτα (tagged protocols).
Παρουσιάζουμε την συνάρτηση επιλογής προτάσεων sel(R), σύμφωνα με τα χαρακτηριστικά που αναφέραμε.

$$\mathrm{sel}(\mathrm{R} \doteq \mathrm{H} \implies \mathrm{C}) \doteq \begin{cases} 0 & \text{εάν για κάθε } \mathrm{F}_i \in \mathrm{H},\ \mathrm{F}_i{=}\text{attacker(x)} \\ F_0 & \text{εάν } \mathrm{F}_0 \neq \text{attacker(x)},\ \mathrm{F}_0 \in \mathrm{H} \end{cases}$$

Χρησιμοποιώντας αυτήν την συνάρτηση για να επιλέξουμε τις προτάσεις που θα εφαρμόσουμε τον κανόνα resolution, βελτιώνουμε σημαντικά την ταχύτητα του αλγόριθμου με τα ίδια αποτελέσματα που θα είχαμε αν δεν χρησιμοποιούσαμε κάποια συνάρτηση επιλογής. Επιπλέον μπορούν να γίνουν και άλλες βελτιστοποιήσεις. Ενδεικτικά, όταν έχουμε πολλές φορές το ίδιο

γεγονός στις υποθέσεις μίας πρότασης, τότε την απλοποιούμε κρατώντας μόνο μια εμφάνιση του. Όταν έχουμε μια πρόταση όπου το συμπέρασμα βρίσκεται και στις υποθέσεις, τότε η πρόταση είναι ταυτολογία και αφαιρείται. Ακόμα, όταν έχουμε μια πρόταση που περιέχει στις υποθέσεις της το γεγονός attacker(x) και η μεταβλητή x δεν εμφανίζεται πουθενά αλλού στην πρόταση, τότε αφαιρούμε το attacker(x). Αυτό γίνεται διότι τετριμμένα σε κάθε πρωτόκολλο ο εχθρός πάντα έχει στην κατοχή του ένα μήνυμα, δηλαδή υπάρχει ως γεγονός το attacker(a) για κάποιο a και άρα ικανοποιείται το attacker(x) για τουλάχιστον ένα x. Τέλος, ανάλογα με το πρωτόκολλο μπορούμε να κάνουμε manually απλοποιήσεις. Παράδειγμα, αν γνωρίζουμε εκ των προτέρων ότι κάποιος όρος s παραμένει μυστικός, τότε το δηλώνουμε και η ProVerif αφαιρεί το γεγονός attacker(s) από τις προτάσεις που το έχουν ως υπόθεση, μειώνοντας έτσι τον χώρο που θα κάνει αναζήτηση. Επιπλέον μπορούμε να επέμβουμε και στην συνάρτηση sel, για παράδειγμα αν επιλέγονται στιγμιότυπα ενός γεγονότος F που οδηγούν σε loop παραγωγής προτάσεων, μπορούμε να χρησιμοποιήσουμε την εντολή nounif F για να μην επιλέγονται στιγμιότυπα του F, ενώ αυτόματα και η ProVerif προσπαθεί να ανιχνεύσει τέτοιες περιπτώσεις.

Δεδομένης της συνάρτησης sel(R), της πρότασης R $\circ_{F_0}$R′ που προκύπτει από τον κανόνα resolution των R,R′ πάνω στο γεγονός F_0, παρουσιάζουμε τον αλγόριθμο saturate($\mathbf{R}_0$), όπου ξεκινώντας από το αρχικό σύνολο προτάσεων Horn $\mathbf{R}_0$ που περιγράφει το πρωτόκολλο μας, καταλήγουμε σε ένα άλλο σύνολο προτάσεων μέσω του κανόνα resolution και των κριτηρίων επιλογής που αναφέραμε και από αυτό το σύνολο θα ελέγξουμε αν παράγεται εν τέλει το γεγονός που μας ενδιαφέρει. Η εντολή elim(R $\cup$ $\mathbf{R}$) που θα χρησιμοποιήσουμε στον αλγόριθμο ελέγχει αν στο σύνολο προτάσεων $\mathbf{R}$ υπάρχει κάποια πρόταση R′ $\in$ $\mathbf{R}$ ώστε R′ $\sqsupseteq$ R. Αν δεν υπάρχει τότε η R προστίθεται στο $\mathbf{R}$.

saturate($\mathbf{R}_0$):

i. $\mathbf{R}$=0
   Για κάθε R $\in$ $\mathbf{R}_0$
      $\mathbf{R}$= elim(R $\cup$ $\mathbf{R}$)

ii. Επανάλαβε έως ένα fix point
      Για κάθε R $\in$ $\mathbf{R}$ με sel(R) = 0
       Για κάθε R′ $\in$ $\mathbf{R}$ με sel(R′) $\neq$ 0
        Για κάθε F_0 που ορίζεται το R $\circ_{F_0}$R′

$$\mathbf{R} = \text{elim}(\text{R} \circ_{F_0} \text{R}' \cup \mathbf{R})$$

iii. Επίστρεψε $\text{R} \in \mathbf{R}$ με $\text{sel}(\text{R}) = 0$

Στο πρώτο βήμα κάνουμε ουσιαστικά ένα ξεκαθάρισμα του συνόλου $\mathbf{R}_0$ από προτάσεις που περιλαμβάνονται σε άλλες προτάσεις μέσα στο σύνολο.
Στο δεύτερο βήμα κάνουμε resolution με τον τρόπο που αναφέραμε, δηλαδή στην πρόταση $\text{R} \circ_{F_0} \text{R}'$, η R έχει γεγονότα μόνο της μορφής attacker(x), η R′ δεν έχει κανένα γεγονός της μορφής attacker(x) και η υπόθεση F_0 της R′ που χρησιμοποιούμε στην resolution ενοποιείται με το συμπέρασμα της R. Σχηματίζουμε λοιπόν τέτοιες προτάσεις και μετά ελέγχουμε να μην περιλαμβάνονται στο σύνολο $\mathbf{R}$ ώστε να μπορούμε να τις προσθέσουμε.
Στο τρίτο βήμα κρατάμε τις προτάσεις του συνόλου $\mathbf{R}$ που έχουν μόνο υποθέσεις της μορφής attacker(x). Επίσης κρατούνται οι προτάσεις χωρίς υποθέσεις, δηλαδή της μορφής $\implies$ attacker(t) για κάποιο όρο t (αφού όταν ισχύει το attacker(t) ισχύει και η πρόταση attacker(x) $\implies$ attacker(t)). Ο λόγος που κρατάμε μόνο αυτές τις προτάσεις είναι διότι ουσιαστικά περιέχουν όλη την γνώση που έχουμε. Αυτό το βλέπουμε εξετάζοντας την αντίθετη περίπτωση. Αν έχουμε κάποια πρόταση που έχει ως υπόθεση το γεγονός attacker(p), όπου p κάποιος όρος που δεν είναι μεταβλητή, και το attacker(p) ικανοποιείται, τότε θα έχει γίνει resolution με αυτήν την πρόταση και θα κρατήσουμε το αποτέλεσμα της, ενώ αν το attacker(p) δεν ικανοποιείται η πρόταση που το έχει ως υπόθεση θα είναι άχρηστη για την παραγωγή γνώσης.

Προφανώς θέλουμε κάθε πρόταση που παράγεται από το $\mathbf{R}_0$ να παράγεται και από το saturate($\mathbf{R}_0$) και αντίστροφα. Στο [7] υπάρχει αναλυτική απόδειξη. Επιγραμματικά, αν έχουμε ένα δένδρο παραγωγή της F από το $\mathbf{R}_0$ και R, R′ δύο προτάσεις - κόμβοι σε αυτό το δένδρο όπου η R είναι παιδί της R′, τότε αυτές οι δύο προτάσεις ενοποιούνται πάνω στο γεγονός που είναι ετικέτα στην ακμή από την R′ στην R. Παίρνοντας το χαμηλότερο κόμβο - πρόταση με $\text{sel}(\text{R}') \neq 0$ (τετριμμένα μια πρόταση - φύλλο έχει $\text{sel}(\text{R}) = 0$ καθώς δεν έχει υποθέσεις) , ενοποιούμε με κάποιο παιδί R με $\text{sel}(\text{R}) = 0$ και κρατάμε στο δένδρο παραγωγής την πρόταση $\text{R} \circ_{F_0} \text{R}'$ αντί των R, R′ μειώνοντας επιπλέον κατά ένα τον αριθμό των προτάσεων. Συνεχίζοντας αυτή τη διαδικασία καταλήγουμε σε προτάσεις R με $\text{sel}(\text{R}) = 0$ που παράγουν τα ίδια γεγονότα με το $\mathbf{R}_0$. Το αντίστροφο είναι προφανές, καθώς κάθε πρόταση του saturate($\mathbf{R}_0$) περιέχεται στο $\mathbf{R}_0$ ή είναι αποτέλεσμα του κανόνα resolution από προτάσεις του $\mathbf{R}_0$.

Έχοντας τροποποιήσει το αρχικό μας σύνολο μέσω του saturate($\mathbf{R}_0$) = $\mathbf{R}_1$, θα προσπαθήσουμε να βρούμε μια παραγωγή του ζητούμενου γεγονότος F

από το $\mathbf{R}_1$. Οπότε κάνουμε αναζήτηση στις προτάσεις του $\mathbf{R}_1$ με τον παρακάτω αλγόριθμο.

$$\text{deriv(R, } \mathbf{R}\text{, } \mathbf{R}_1) \begin{cases} 0 & \text{εάν υπάρχει R}' \in \mathbf{R} \text{ ώστε R} \sqsubseteq \text{R}' \\ \text{R} & \text{αλλιώς, εάν sel(R)} = 0 \\ \bigcup[\text{deriv}(\text{R}'\circ_{F_0}\text{R}, \text{R} \cup \mathbf{R}_1, \mathbf{R}_1)| \text{ για τα R}' \in \mathbf{R}_1 \\ \text{ που ορίζεται το R}'\circ_{F_0}\text{R}] & \text{σε κάθε άλλη περίπτωση} \end{cases}$$

Η αναζήτηση γίνεται δίνοντας την εντολή deriv(F $\Longrightarrow$ F, 0, $\mathbf{R}_1$). Γενικά, αν το F είναι ένα γεγονός με μεταβλητές, το deriv(F $\Longrightarrow$ F, 0, $\mathbf{R}_1$) θα αναζητήσει προτάσεις R: H $\Longrightarrow$ C όπου το C είναι κάποιο στιγμιότυπο του F, δηλαδή θα ισχύει C = $\boldsymbol{\sigma}$F για κάποια αντικατάσταση $\boldsymbol{\sigma}$, ενώ η αντικατάσταση $\boldsymbol{\sigma}$ θα πρέπει να δίνει υποθέσεις $\boldsymbol{\sigma}$H που ικανοποιούνται από το saturate($\mathbf{R}_0$). Αν λοιπόν το F είναι κλειστό γεγονός, τότε για κάθε πρόταση που προκύπτει από το deriv(F $\Longrightarrow$ F, 0, $\mathbf{R}_1$) θα ισχύει C = F. Οπότε ουσιαστικά αναζητούμε από το saturate($\mathbf{R}_0$) "τουλάχιστον μία" πρόταση H $\Longrightarrow$ F. Η πρόταση H $\Longrightarrow$ F ενοποιείται με την F $\Longrightarrow$ F, το F δεν έχει μεταβλητές άρα δεν χρησιμοποιείται κάποια αντικατάσταση για την ενοποίηση και τα H όπως είπαμε είναι της μορφής attacker(x) και ικανοποιήσιμα (τετριμμένα, αφού πάντα ισχύει το κατηγόρημα attacker(a) για τουλάχιστον έναν κλειστό όρο a), συνεπώς η εύρεση μιας πρότασης H $\Longrightarrow$ F συνεπάγεται ότι ισχύει το γεγονός F. Εν τέλει, αν το deriv(F $\Longrightarrow$ F, 0, $\mathbf{R}_1$) για το κλειστό γεγονός F δώσει έστω και μια πρόταση R, τότε το F παράγεται από το αρχικό σύνολο $\mathbf{R}_0$, ενώ αν deriv(F $\Longrightarrow$ F, 0, $\mathbf{R}_1$) = 0 τότε το F δεν παράγεται από το $\mathbf{R}_0$.

Συνοψίζοντας την διαδικασία, αρχικά το πρωτόκολλο μεταφράζεται σε προτάσεις Horn. Μέσω του saturate, επιλέγουμε από αυτό το σύνολο προτάσεις που δεν περιλαμβάνονται σε άλλες προτάσεις και προχωράμε σε εφαρμογή του κανόνα resolution όπου ορίζεται και σύμφωνα με τα κριτήρια που αναφέραμε, κρατώντας στο τέλος τις προτάσεις με sel(R) = 0. Μέσω του deriv κάνουμε αναζήτηση στις προτάσεις του saturate για να λάβουμε ως απάντηση αν παράγεται το επιθυμητό γεγονός F.

Κεφάλαιο 4

Verifpal

4.1 Συντακτικό

Σε αυτό το κεφάλαιο θα εξετάσουμε το εργαλείο Verifpal με απώτερο στόχο να κάνουμε μια σύγκριση με την ProVerif σε ένα πραγματικό πρωτόκολλο γραμμένο και στα δύο εργαλεία. Σύμφωνα με τους δημιουργούς της Verifpal, ο σκοπός της είναι να απευθύνεται σε απλούς χρήστες, "real word practitioners" όπως χαρακτηριστικά σημειώνεται [8]. Η γλώσσα που χρησιμοποιείται είναι φιλική και διαισθητική, ούσα πολύ κοντά στον τρόπο που θα περιέγραφε κάποιος τις διαδικασίες ενός πρωτοκόλλου. Ουσιαστικά για να χρησιμοποιήσει κάποιος την Verifpal δεν απαιτείται να ασχοληθεί με κάποια λογική και τη θεωρία γύρω από αυτήν, όπως κάποιος χρήστης της ProVerif.

Παρουσιάζουμε τα βασικά στοιχεία του συντακτικού.

"constant" ::= (string)

"principal" ::= principal (string) ["knows" | "generates" | "leaks" | "assignment"]

"attacker" ::= attacker [(passive | active)]

"knows" ::= knows (private | public | password) "constant"

"generates" ::= generates "constant"

"leaks" ::= leaks "constant"

"assignment" ::= "constant" = ("primitive" | "equation")

"equation" := "constant"^"constant"

"primitive" ::= "primitiveName" ("constant" | "primitive" | "equation")

"primitiveName" ::= (ENC | DEC | AEAD_ENC | AEAD_DEC | MAC | PW_HASH | SIGN | SIGNVERIF | BLIND | UNBLIND | CONCAT | SPLIT | SHAMIR_SPLIT | SHAMIR_JOIN | etc...)

"message" ::= "principal" → "principal" : "constant"

"phase" ::= phase (number)

"queries" ::= queries ["confidentiallityQuery" | "authetication Query" | "freshnessQuery" | "unlinkabilityQuery"]

Η Verifpal έχει τρεις θεμελιώδεις τύπους, τις σταθερές (constants), τα primitives και τις εξισώσεις (equations).

Υπάρχουν διάφοροι τρόποι για να δηλώσεις και να μεταχειριστείς μια σταθερά. Με την εντολή "principal knows" δηλώνουμε την γνώση ενός συμμετέχοντα σε μία σταθερά ενώ μπορούμε να χρησιμοποιήσουμε τους χαρακτηρισμούς public ή private για να κάνουμε αυτήν την σταθερά δημόσια προς όλους ή ιδιωτική. Επιπλέον υπάρχει και ο χαρακτηρισμός password για ιδιωτικές σταθερές που είναι εύκολο να τις μαντέψει κάποιος εχθρός. Για να μοντελοποιήσουμε φρέσκες οντότητες που δημιουργούνται από έναν συμμετέχοντα, όπως ένα ιδιωτικό κλειδί ή μια τυχαιότητα, χρησιμοποιούμε την εντολή "principal generates". Η εντολή leaks γνωστοποιεί μια σταθερά στον εχθρό και μπορεί να χρησιμοποιηθεί όταν θέλουμε να γίνει αυτή η διαρροή σε συγκεκριμένη στιγμή. Τέλος, οι σταθερές μπορούν να είναι αποτέλεσμα κάποιου primitive ή κάποιας εξίσωσης και αυτοί είναι οι τρόποι για να δημιουργήσουμε σταθερές μέσω άλλων σταθερών.

Η Verifpal παρέχει μια πληθώρα κρυπτογραφικών primitives, απαγορεύοντας στον χρήστη να τα κατασκευάσει ο ίδιος, τα οποία θεωρούνται τέλειες κατασκευές που δεν είναι ευάλωτες σε επιθέσεις. Για παράδειγμα οι συναρτήσεις encryption θεωρούνται τέλειες ψευδό - τυχαίες μεταθέσεις και οι hash συναρτήσεις θεωρούνται συναρτήσεις μονής κατεύθυνσης. Υπάρχουν

τέσσερις κανόνες αναγωγής που ουσιαστικά αποτελούν την σημασιολογία των primitives. Δεδομένων του arity και του αριθμού των outputs, κάθε primitive ορίζεται από κάποιους από αυτούς τους κανόνες,

- Decompose: Από ένα output του primitive και ενός υποσυνόλου των inputs οδηγούμαστε σε κάποιο input (π.χ. ENC(k,m) ∧ k → m)

- Recompose: Από ένα σύνολο outputs του primitive οδηγούμαστε σε κάποιο input (π.χ. το primitive SHAMIR_SPLIT(x) σπάει το x σε τρία κομμάτια a,b,c και η γνώση δύο από αυτών μπορεί να μας ανακτήσει το x, οπότε έχουμε a ∧ b → x.)

- Rewrite: Κανόνες αναγωγής που βασίζονται σε pattern matching (π.χ. DEC(k,ENC(k,m)) → m).

- Rebuild: Αντίστοιχα με τον κανόνα rewrite, μόνο που τα inputs είναι outputs αποκλειστικά κάποιου άλλου συσχετιζόμενου primitive (π.χ. δεδομένων των outputs a,b,c του SHAMIR_SPLIT(x) έχουμε SHAMIR_JOIN(a,b) → x).

Ας δούμε κάποια primitives που παρέχονται, χωρίζοντάς τα σε κατηγορίες ανάλογα με τον σκοπό χρήσης τους.

Core: ASSERT(x,x): Ελέγχει την ισότητα δύο τιμών.
CONCAT(a,b,...): Παράθεση των τιμών a,b,.., λαμβάνοντας δύο έως πέντε τιμές.
SPLIT(CONCAT(a,b): Χωρίζει τιμές που έχουν γίνει CONCAT.

Encryption: ENC(k,m): Συμμετρική κρυπτογράφηση
DEC(k,ENC(k,m)): Συμμετρική αποκρυπτογράφηση
AEAD_ENC(k,m,ad): Κρυπτογράφηση με αυθεντικοποίηση, όπου η τιμή ad αυθεντικοποιεί τον χρήστη.
AEAD_DEC(k,AEAD_ENC(k,m,ad),ad): Αποκρυπτογράφηση με αυθεντικοποίηση.
PKE_ENC(G^k, m): Κρυπτογράφηση δημοσίου κλειδιού. Το G χρησιμοποιείται πάντα ως γεννήτορας στην Verifpal.
PKE_DEC(G^k, PKE_ENC(G^k,m)): Αποκρυπτογράφηση δημοσίου κλειδιού.

Hashing: HASH(a,b,...): Hash συνάρτηση, παίρνει από ένα εως πέντε inputs και επιστρέφει ένα output.
MAC(k,m): Hash συνάρτηση με κάποιο κλειδί k, που χρησιμοποιείται συνήθως για αυθεντικοποίηση του μηνύματος λόγω του παράγοντα k.
HDKF(a,b,c): Hash συνάρτηση που εξάγει περισσότερα του ενός κλειδιού από μια κρυφή τιμή, επιστρέφει από ένα έως πέντε outputs. Βασισμένη στο σχήμα Krawczyk HKDF.
PW_HASH(a): Hash συνάρτηση για σταθερές τύπου password, που είναι ευάλωτες στον εχθρό.

Signature: SIGN(k,m): Primitive υπογραφής με το ιδιωτικό κλειδί k.
SIGNVERIF(G^k,mess,SIGN(k,m)): Επιβεβαίωση υπογραφής μέσω του G^k αν το k έχει χρησιμοποιηθεί ως κλειδί υπογραφής.
BLIND(k,m): Primitive που χρησιμοποιείται για τυφλές υπογραφές.
UNBLIND(k,m,SIGN(a,BLIND(k,m))): Επιστρέφει το SIGN(a,m). Συνδυαστικά με το BLIND(k,m), είναι χρήσιμα και στις ηλεκτρονικές ψηφοφορίες για την εγκυρότητα του ψηφοφόρου να ψηφίσει, στέλνοντας μια τυφλή υπογραφή στην Αρχή και λαμβάνοντας την έγκριση της αρχής.

Secret Sharing: SHAMIR_SPLIT(k): Τεμαχίζει το k σε τρία κομμάτια a,b,c.
SHAMIR_JOIN(a,b): Εάν τα a,b αποτελούν οποιαδήποτε δύο από τα τρία κομμάτια του SHAMIR_SPLIT(k) τότε επιστρέφεται το k.

Η χρήση εξισώσεων γίνεται συγκεκριμένα για την κρυπτογράφηση δημοσίου κλειδιού, όπως την ανταλλαγή κλειδιού Diffie - Hellman. Από το συντακτικό βλέπουμε ότι μια σταθερά εξισώνεται με τη φράση "σταθερά"^"σταθερά", δηλαδή η σταθερά στο αριστερό κομμάτι της εξίσωσης είναι το αποτέλεσμα της πράξης ^. Με αυτό τον τρόπο μπορούμε να δημιουργήσουμε σταθερές πάνω σε άλλες σταθερές, κάτι που όπως είπαμε γίνεται μόνο με εξισώσεις ή με τη χρήση κάποιου primitive. Κάθε εξίσωση χρησιμοποιεί άμεσα είτε αναδρομικά το γεννήτορα G, μια σταθερά που υπάρχει built-in στην Verifpal.

Η ανταλλαγή μηνυμάτων γράφεται πολύ απλοϊκά. Αν σε κάποιο βήμα θέλει ο A να στείλει το m στον B, απλά γράφουμε A→B : m. Βάζοντας brackets στο μήνυμα m, κάνουμε το m προστατευμένο από κάποιον ενεργητικό εχθρό, όπου μπορεί να διαβάσει το m αλλά δεν μπορεί να το τροποποιήσει. Επιπλέον με την εντολή phase μπορούμε να χωρίσουμε το πρωτόκολλο μας σε φάσεις, όπου κάθε φάση ξεκινάει αφού έχει τελειώσει η προηγούμενη. Με αυτόν τον

τρόπο μπορούμε να επιλέξουμε τη χρονική στιγμή όπου κάποια μηνύματα γνωστοποιούνται στον εχθρό και χρησιμοποιούνται από τον εχθρό, μοντελοποιώντας ιδιότητες ασφαλείας όπως την "forward secrecy".

Ολοκληρώνουμε την περιγραφή του συντακτικού με τα queries. Τα queries αποτελούνται από τέσσερις κατηγορίες και γράφονται με τις παρακάτω εντολές.

Confidentiality? m:
Ελέγχουμε αν ο εχθρός έχει στην κατοχή του το μήνυμα m. Αυτό το ερώτημα μπορεί να χρησιμοποιηθεί συνδυαστικά και με φάσεις, για να ελέγξουμε ιδιότητες όπως την forward secrecy.

Authentication? Bob → Alice: m:
Εξετάζεται αν το μήνυμα m, που μπορεί να παίζει το ρόλο κάποιας υπογραφής ή κάποιου μηνύματος αυθεντικοποίησης, έχει σταλεί από τον Bob στην Alice και όχι από κάποιον εχθρό που προσπαθεί να αναπαραστήσει την Bob.

Freshness? c:
Ελέγχεται αν η τιμή c είναι φρέσκια, δηλαδή αν αλλάζει σε κάθε εκτέλεση του πρωτοκόλλου. Με αυτό τον έλεγχο ανιχνεύονται επιθέσεις τύπου replay, όπου ο εχθρός μπορεί να χρησιμοποιήσει επαναλαμβανόμενα την τιμή c που έχει υποκλέψει.

Unlinkability? a, b:
Η Verifpal ορίζει την έννοια unlinkability μεταξύ δύο τιμών με το αν ο εχθρός δεν μπορεί να καταλάβει αν σε διαφορετικές εκτελέσεις ενός πρωτοκόλλου, οι τιμές ανήκουν στον ίδιο ή διαφορετικό συμμετέχοντα. Σημειώνεται ότι ο τρόπος που γίνεται αυτός ο έλεγχος είναι προβληματικός καθώς δεν συμπεριλαμβάνονται όλες οι περιπτώσεις που δύο τιμές δεν συνδέονται, και έτσι αφού ολοκληρωθεί ο έλεγχος μπορεί να θεωρηθεί εσφαλμένα ότι συνδέονται.

4.2 Ανάλυση της Verifpal

Όπως και στην Proverif, ο στόχος μας είναι να βρούμε τις τιμές που γίνονται γνωστές στον εχθρό. Η Verifpal προσπαθεί να συγκεντρώσει όσες

περισσότερες τέτοιες τιμές μπορεί και κάθε ερώτημα απαντάτε από αυτό το σύνολο τιμών. Για κάποια ερωτήματα απαιτούνται και καταγράφονται κάποιοι "ιχνηλάτες" για το πώς αποκτήθηκαν ορισμένες τιμές. Στο [8] παρουσιάζεται η διαδικασία που ακολουθείται με μία περιγραφική ματιά, χωρίς να υπεισέρχεται σε αναλυτικές λεπτομέρειες, και αυτό θα κάνουμε και εμείς εδώ.

Αρχικά το πρωτόκολλο αναλύεται και μεταφράζεται σε μια αμετάβλητη δομή που λέγεται "χάρτης γνώσης". Ο χάρτης γνώσης καταγράφει τους συμμετέχοντες και αποδίδει τιμές σε κάθε σταθερά. Οι σταθερές που δηλώνονται με την εντολή "knows" ή "generates" ουσιαστικά ανάγονται στον εαυτό τους και οι σταθερές που είναι αποτέλεσμα κάποιου primitive ή κάποιας εξίσωσης ανάγονται σε τιμές που αφορούν το αντίστοιχο primitive ή την αντίστοιχη εξίσωση. Επιπλέον καταγράφονται τιμές που αφορούν τον δημιουργό μιας σταθεράς και τιμές που σχετίζονται με ενδεχόμενες φάσεις του πρωτοκόλλου. Από τον χάρτη γνώσης προκύπτουν για τον εχθρό και κάθε συμμετέχοντα κάποια δυναμικά σύνολα τιμών που ονομάζονται καταστάσεις, συμβολίζουμε με V_A την κατάσταση του εχθρού και V_{P_i} την κατάσταση κάθε συμμετέχοντα P_i. Στην πρώτη φάση, τα σύνολα V_A και V_{P_i} αποτελούνται από τιμές του χάρτη γνώσης και τιμές που αφορούν την ανταλλαγή μηνυμάτων του πρωτοκόλλου. Κάθε σταθερά που μεταδίδεται ως μη - προστατευμένο μήνυμα γίνεται γνωστή και στον εχθρό και εισάγεται στο V_A, καθώς και οι πληροφορίες που αφορούν το δημιουργό της και τα ίχνη της στο πρωτόκολλο, αφού για κάθε σταθερά καταγράφονται και τιμές που αφορούν την ιχνηλάτισή της.

Στην επομένη φάση, ο εχθρός επεξεργάζεται τις τιμές που έχει στην κατοχή του ωστέ να εξάγει και άλλες τιμές, χρησιμοποιώντας τέσσερις μετασχηματισμούς.

Resolve: Ανάλυση της τιμής κάθε νέας σταθεράς που μαθαίνει ο εχθρός συγκριτικά με τις ήδη υπάρχουσες τιμές του V_A. Όπως είπαμε, κάθε σταθερά ανάλογα με το πως κατασκευάζεται ανάγεται στον εαυτό της, σε κάποιο primitive ή σε κάποια εξίσωση.

Equivalize: Ο εχθρός εξετάζει αν μπορεί να εξισώσει ή να κατασκευάσει τιμές των V_{P_i} μέσω τιμών του V_A. Αν υπάρχουν τέτοιες τιμές τότε τις προσθέτει στο V_A.

Deconstruct: Αποδόμηση μίας εξίσωσης ή ενός primitive. Για την αποδόμηση μιας εξίσωσης ο εχθρός πρέπει να κατέχει όλα τα στοιχεία που την αποτελούν

εκτός από ένα. Τα στοιχεία που χρειάζονται για την αποδόμηση ενός primitive προκύπτουν από τους κανόνες που το ορίζουν.

Reconstruct: Κατασκευή μιας εξίσωσης ή ενός primitive. Ο εχθρός πρέπει να κατέχει όλα τα στοιχεία που κατασκευάζουν την εξίσωση ή το primitive.

Αφού έχουν συγκεντρωθεί νέες τιμές μέσω των παραπάνω μετασχηματισμών, δημιουργείται ένας πίνακας με όλους τους πιθανούς συνδυασμούς αντικαταστάσεων που περιέχουν τις νέες τιμές του V_A και εξάγονται οι έγκυρες αντικαταστάσεις που μπορούν να χρησιμοποιηθούν στις τιμές των συνόλων V_{P_i}, έτσι ώστε να τροποποιηθούν τα μηνύματα που ανταλλάσσονται κατά την εκτέλεση του πρωτοκόλλου. Δηλαδή εκτελούνται επιθέσεις "Man in the Middle" με τις νέες τιμές. Έπειτα επανερχόμαστε στο βήμα συλλογής τιμών, που την πρώτη φορά ήταν από τον χάρτη γνώσης και την ανταλλαγή των αρχικών μηνυμάτων και τις επόμενες φορές η συλλογή τιμών προέρχεται από τα τροποποιημένα μηνύματα.

Ο σκοπός της Verifpal είναι να συγκεντρωθούν όλες οι τιμές που είναι δυνατόν να γνωρίζει ο εχθρός. Όμως η κατάσταση του εχθρού, οι καταστάσεις των συμμετεχόντων και οι πιθανοί συνδυασμοί αντικαταστάσεων μπορεί να εξελιχθούν σε τεράστια σύνολα , το όποιο προφανώς μπορεί να οδηγήσει σε μη τερματισμό, ειδικά σε περίπλοκα πρωτόκολλα. Αυτό το πρόβλημα αντιμετωπίζεται μέσω ευριστικών τεχνικών. Η ανάλυση της Verifpal στο κομμάτι της αντικατάστασης τιμών στα V_{P_i} (για να γίνουν οι επιθέσεις Man in the Middle) εξελίσσεται σταδιακά, ξεκινώντας από συγκεκριμένες τιμές και καταλήγοντας στην επεξεργασία όλου του συνόλου των τιμών. Περιγράφουμε τη διαδικασία για την αντικατάσταση τιμών σε ένα σύνολο V_{P_i}, χωρίζοντάς την σε φάσεις.

1η φάση: Όλες οι τιμές που προκύπτουν από την ανάλυση ενός παθητικού εχθρού, μαζί με τις σταθερές και τα στοιχεία των εξισώσεων του V_{P_i} μπορούν να αντικατασταθούν μόνο σε nil (παράδειγμα, η εξίσωση G^a αντικαθίσταται σε G^nil).

2η φάση: Κάνουμε αντικαταστάσεις με όλες τις τιμές της 1ης φάσης και την προσθήκη όλων των primitives που είναι non - explosive. Ο διαχωρισμός των primitives σε explosive και non - explosive είναι ακόμα ένα μέσο βελτιστοποίησης της ανάλυσης. Στις αντικαταστάσεις που γίνονται στα non - explosive primitives υπάρχουν περιορισμοί στο

βάθος, παράδειγμα από το HASH(HASH(x)) δεν θα προκύψει το HASH(HASH(HASH(y))). Επίσης τα inputs που αντικαθίσταται πρέπει να είναι του ίδιου είδους, παράδειγμα από το ENC(HASH(k),G^y) δεν θα προκύψει το ENC(PW_HASH(k),k) καθώς απαιτείται να χρησιμοποιηθεί το primitive HASH στο πρώτο input και στο δεύτερο input απαιτείται να χρησιμοποιηθεί κάποια εξίσωση όπως το G^y και όχι μια σταθερά όπως το k.

3η φάση: Προσθέτουμε και τα explosive primitives.

4η φάση: Επιτρέπουμε τώρα και τις αντικαταστάσεις σταθερών με σταθερών, είτε αυτές είναι μόνες τους είτε ως στοιχεία μίας εξίσωσης (παράδειγμα, η εξίσωση G^a μπορεί να αντικατασταθεί από κάποια εξίσωση G^b).

5η φάση: Αυξάνουμε το βάθος της αντικατάστασης σε ένα primitive, διατηρώντας τους περιορισμούς στα inputs όπως είδαμε στη 2η φάση.

Άλλες παρεμβάσεις που κάνει η Verifpal για την αποφυγή μη - τερματισμού είναι ο περιορισμός των input και των output ενός primitive έως και πέντε και η χρήση τεχνικών όπως το multithreading για πιο γρήγορη ανάλυση. Βεβαίως όλα αυτά δεν μας εξασφαλίζουν τον τερματισμό, απότοκο του ότι η Verifpal, όπως και η ProVerif, ακολουθεί μη φραγμένη ανάλυση.

Η εγκυρότητα των αποτελεσμάτων της Verifpal ελέγχεται μέσω άλλων εργαλείων. Αρκετά πρωτόκολλα δύναται να μεταφραστούν στην ProVerif και στο Coq Theorem Prover, αξιοποιώντας την δική τους ανάλυση. Εναλλακτικός τρόπος είναι η σύγκριση των αποτελεσμάτων με άλλα εργαλεία αυτόματης ανάλυσης που είναι αποδεδειγμένη η εγκυρότητα των αποτελεσμάτων τους (ProVerif, Tamarin, CryptoVerif). Όπως αναφέρεται στο [8], τα αποτελέσματα της ανάλυσης 54 πρωτοκόλλων συμβαδίζουν με τα αντίστοιχα άλλων έγκυρων εργαλείων. Προφανώς αυτή η αξιολόγηση μέσω τρίτων γεννά προβληματισμό ως προς τη θεμελίωση της Verifpal και δημιουργεί εμπόδια στη χρήση της ως ένα αυτόνομο εργαλείο. Άλλωστε η Verifpal δεν εξασφαλίζει ότι εντοπίζει κάθε επίθεση σε κάθε πρωτόκολλο που εκφράζεται στη γλώσσα της, καθιστώντας απαραίτητο τον συγκριτικό έλεγχο με άλλα εργαλεία. Έτσι, ο απλοϊκός τρόπος χρήσης που προωθείται από τους δημιουργούς της Verifpal συνοδεύεται από αυτές τις αδυναμίες.

Κεφάλαιο 5

Εφαρμογή και Σύγκριση των ProVerif, Verifpal

5.1 Παρουσίαση του QUIC Handshake Protocol

Σε αυτό το κεφάλαιο θα ασχοληθούμε με την εφαρμογή και την σύγκριση των δύο εργαλείων πάνω στο ίδιο πρωτόκολλο, όπως παρουσιάζεται στο [9]. Το πρωτόκολλο που χρησιμοποιείται είναι το QUIC Handshake Protocol, σχεδιασμένο από την Google, είναι στα standards του IETF (Internet Engineering Task Force) και έχει γίνει βασική επιλογή στην μεταφορά δεδομένων και το streaming για εταιρείες όπως η Google κατά κύριο λόγο, η Facebook, η Amazon, το Netflix και άλλες. Διαστρωματικά τοποθετείται στο στρώμα μεταφοράς, πάνω από το UDP, και χρησιμοποιεί κρυπτογραφημένη επικοινωνία όπως θα δούμε, με ταυτόχρονη μείωση του τελικού χρόνου μεταφοράς των πακέτων δεδομένων. Έρχεται ως εναλλακτική λύση στις συνδέσεις που χρησιμοποιούν τα TCP/TLS, αντικαθιστώντας κομμάτια τους και βελτιώνοντας την απόδοση συγκριτικά, ενώ το σημαντικό είναι ότι επιτυγχάνει να προσφέρει ασφάλεια και αξιοπιστία ως ένα πρωτόκολλο με βάση το UDP. Στο [9], το QUIC Handshake Protocol έχει γραφτεί και στην ProVerif και στην Verifpal και γίνεται έλεγχος σε ορισμένες ιδιότητες ασφαλείας, καταλήγοντας σε σύγκριση των δύο εργαλείων όσον αφορά την συγκεκριμένη περίπτωση, οδηγώντας όμως και σε γενικά συμπεράσματα.

Παρουσιάζουμε περιγραφικά και σχηματικά το πρωτόκολλο, πριν καταγράψουμε την υλοποίησή του στα δύο εργαλεία. Έχουμε την επικοινωνία ενός Client και ενός Server σε δημόσιο κανάλι όπου μπορεί να έχει πρόσβαση και κάποιος εχθρός. Μια χειραψία (handshake) μέσω QUIC μπορεί να γίνει σε 1-RTT (Round Trip Time), όπου αρχικά ο Client στέλνει ένα αναγνωριστικό μήνυμα για να λάβει τις απαραίτητες πληροφορίες ώστε το επόμενο μήνυμα που θα στείλει να είναι ικανό για να πραγματοποιηθεί επιτυχημένη χειραψία. Αν η χειραψία δεν είναι επιτυχημένη, ο Server στέλνει μηνύματα απόρριψης που περιέχουν πληροφορίες για να ξαναπροσπαθήσει ο Client. Προφανώς το πρώτο μήνυμα που στέλνει ο Server είναι μήνυμα απόρριψης καθώς ο Client δεν διαθέτει πληροφορίες εκ των προτέρων. Αφού έχει γίνει επιτυχημένη χειραψία, τυχόν επόμενες συνδέσεις μπορούν να γίνουν άμεσα χωρίς ανάγκη χειραψίας (0 - RTT).

Θα χρησιμοποιήσουμε τα εξής:

CEPri: Προσωρινή ιδιωτική τιμή Diffie - Hellman του Client.

CEPub: Προσωρινή δημόσια τιμή Diffie - Hellman του Client.

SEPri: Προσωρινή ιδιωτική τιμή Diffie - Hellman του Server.

SEPub: Προσωρινή δημόσια τιμή Diffie - Hellman του Server.

LPri: Μακροπρόθεσμη ιδιωτική τιμή Diffie - Hellman του Server.

LPub: Μακροπρόθεσμη δημόσια τιμή Diffie - Hellman του Server.

g: Γεννήτορας.

InitKc: Αρχικό κλειδί του Client.

InitKS: Αρχικό κλειδί του Server.

skS: Ιδιωτικό κλειδί υπογραφής του Server.

pkS: Δημόσιο κλειδί υπογραφής του Server.

FSKC: Κλειδί του Client που είναι forward - secure.

FSKS: Κλειδί του Server που είναι forward - secure.

{ }skS : Το { } υπογράφεται με το skS.

{ }InitKC : Το { } υπογράφεται με το InitKC.

{ }InitKS : Το { } υπογράφεται με το InitKS.

{ }FSKS : Το { } υπογράφεται με το FSKS.

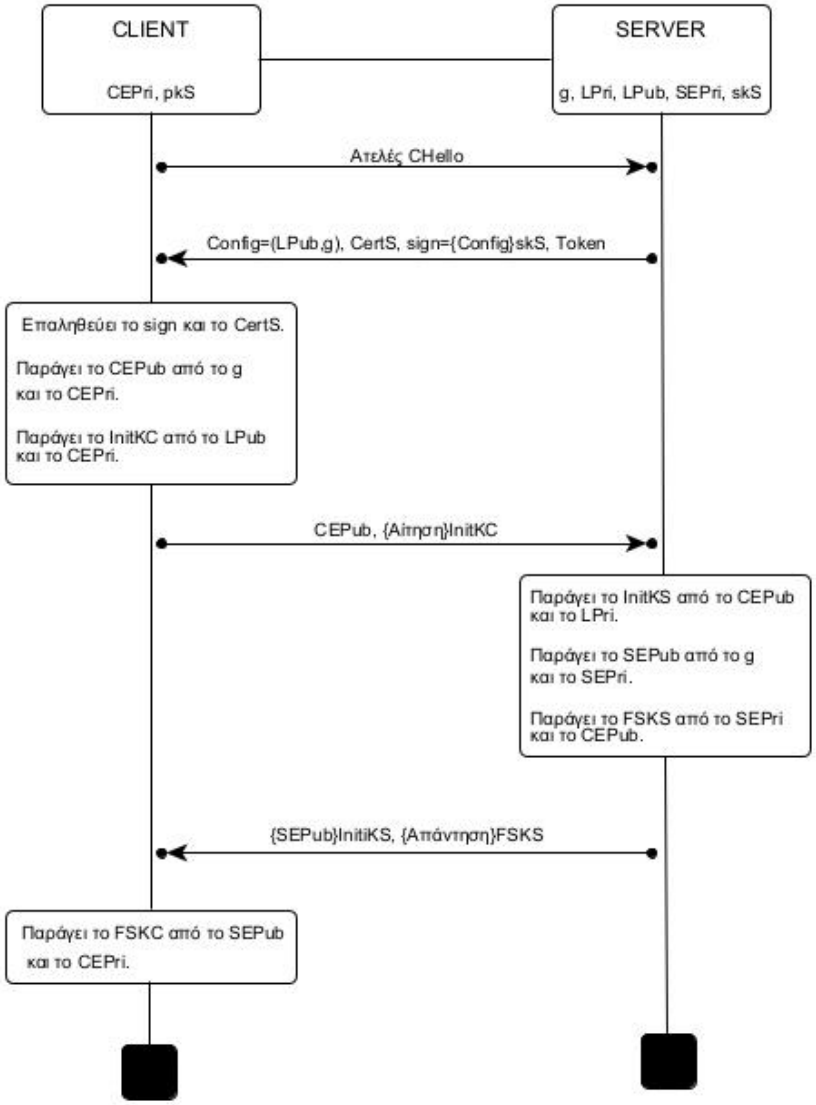

Figure 5.1: QUIC Handshake Protocol

Ο Client στέλνει ένα μήνυμα Hello (Ατελές CHLO) ώστε να λάβει ως απάντηση ένα μήνυμα απόρριψης που θα του παρέχει κάποια στοιχεία. Το

μήνυμα απόρριψης περιέχει τη μακροπρόθεσμη δημόσια τιμή Diffie - Hellman (LPub) μέσα σε ένα σχήμα πληροφοριών (Config), ένα πιστοποιητικό που επαληθεύει τον Server (CertS), την υπογραφή του Config υπογεγραμμένη με το skS και ένα Token ως διεύθυνση πηγής. Αφού τα λάβει ο Client, επαληθεύει τις πληροφορίες από το πιστοποιητικό και την υπογραφή και υπολογίζει το αρχικό του κλειδί συνεδρίας (InitKC) χρησιμοποιώντας την LPub και το δικό του προσωρινό ιδιωτικό κλειδί Diffie - Hellman (CEPri). Επίσης υπολογίζει το δικό του προσωρινό δημόσιο κλειδί (CEPub) από τα g και CEPri. Τώρα είναι έτοιμος να στείλει ένα πλήρες μήνυμα Hello με τα στοιχεία που έλαβε κρυπτογραφημένο με το InitKC, στέλνοντας ταυτόχρονα και το CEPub. Εάν έχουμε επιτυχημένη χειραψία, ο Server υπολογίζει το δικό του αρχικό κλειδί (InitKS) χρησιμοποιώντας το CEPub και τη δική του μακροπρόθεσμη ιδιωτική τιμή Diffie - Hellman (LPri). Επιπρόσθετα, υπολογίζει την δική του προσωρινή δημόσια τιμή Diffie - Helman (SEPub) χρησιμοποιώντας την δική του προσωρινή ιδιωτική τιμή Diffie - Hellman (SEPriv) και υπολογίζει το τελικό του κλειδί (FSKS) μέσω των CEPub και της SEPriv. Ως επιβεβαίωση της χειραψίας, στέλνει ένα Hello μήνυμα στον Client που αποτελείται από το SEPub κρυπτογραφημένο με το InitKS, μαζί με επιπρόσθετα δεδομένα επικοινωνίας κρυπτογραφημένα με το FSKS. Τέλος ο Client παράγει το δικό του τελικό κλειδί (FSKC) από τα SEPub και CEPri, ώστε να το χρησιμοποιήσει για την κρυπτογράφηση των μηνυμάτων στην επακόλουθη επικοινωνίας τους.

5.2 QUIC στην ProVerif

Θα χρειαστούμε τους παρακάτω constructors και destructors. Στο [9], η υλοποίηση παρουσιάζεται με τύπους. Θα το ακολουθήσουμε και εμείς εδώ, αν και στο κεφάλαιο της ProVerif δεν είχαμε ασχοληθεί με τύπους, καθώς γίνεται και πιο κατανοητή η περιγραφή για τον αναγνώστη όταν αναφέρεται ο τύπος σε κάθε στοιχείο που ορίζεται.

Δηλώσεις: type key, type bitstring, type skey, type pkey, type G, type exponent, type channel, const g:G, free c:channel

Συμμ. Κρυπτ.: fun senc(bitstring,key): bitstring
reduc forall m:bitstring, k:key; sed(senc(m,k),k)=m

Υπογραφή:
```
fun pk(skey): pkey
fun sign(G,skey): bitstring
reduc forall m:G, k:skey; getmess(sign(m,k))=m
reduc forall m:G, k:skey; checksign(sign(m,k),pk(k))=m
```

Diffie-Hellman:
```
fun exp(G,exponent): G
equation forall x:exponent, y:exponent;
exp(exp(g,x),y)=exp(exp(g,y),x)
```

Περνάμε στην διεργασία Client:

```
let Client(pkS:pkey, InitKC:G) =
    out(c,CHLO);
    in(c,(x1:G, x2:bitstring, x3:bitstring, x4:bitstring, pkX:pkey));
    if x3 = CertS then
    if pkX = pkS then
    let (=x1) = checksign(x2, pkX) in
    new CEPri: exponent;
    let InitKC = exp(x1, CEPri) in
    new ReqM: bitstring;
    event InitC(CHLO);
    event sendReqM(ReqM);
    out(c, (exp(g, CEPri), enc(ReqM, InitKC)));
    in(c, (x5:G, x6:bitstring));
    let x7 = decG(x5, InitKC)
    let FSKC = exp(x7, CEPri) in
    let ResMx = dec(x6, FSKC) in
    event acceptResM(ResMx); event EndC(x1)
```

Ο Client στέλνει ένα μήνυμα Hello, το ατελές Hello, και περιμένει να λάβει μια απάντηση από τον Server που περιέχει πέντε μεταβλητές, οι τιμές τους φαίνονται στη διεργασία Server. Πρώτα ελέγχεται αν η x3 είναι το πιστοποιητικό CertS και αν η pkX είναι το δημόσιο κλειδί του Server pkS. Έπειτα ελέγχεται αν η μεταβλητή x1 ισούται με το αποτέλεσμα του checksign, με ορίσματα την υπογραφή που λαμβάνεται ως μεταβλητή x2 και το pkS. Το ζητούμενο αποτέλεσμα του checksign είναι η τιμή exp(g, LPri). Αν ισχύει η ισότητα, ο Client υπολογίζει το αρχικό του κλειδί InitKC, τύπου G (αποτέλεσμα ύψωσης του g σε εκθέτη), χρησιμοποιώντας το exp(g, LPri) και

το εκθετικό CEPri και στέλνει το CEPub = exp(g, CEPri) και το πλήρες μήνυμα Hello (ReqM) κρυπτογραφημένο με το InitKC. Μετά περιμένει να λάβει τις μεταβλητές x5, x6. Η x5 αποκρυπτογραφείται με το InitKC για να πάρουμε το SEPub τύπου G και να υπολογίσουμε το FSKC χρησιμοποιώντας το SEPub και το CEPri. Αντίστοιχα η x6 αποκρυπτογραφείται χρησιμοποιώντας το FSKC για να μας δώσει το ResM, που είναι το μήνυμα επιτυχημένης χειραψίας από τη μεριά του Server.

```
let Server(skS:skey, pkS:pkey, InitKS:G) =
    in(c, x1:bitstring);
    if x1 = CHLO then
    new LPri: exponent;
    new Token: bitstring;
    out(c, (exp(g, LPri), sign(exp(g, LPri),skS), CertS, Token, pkS));
    in(c, (x2:G, x3:bitstring));
    let InitKS = exp(x2, LPri) in
    let ReqMx = dec(x3, InitKS) in
    event acceptReqM(ReqMx); new SEPri: exponent;
    let SEPub = exp(g, SEPri) in
    let FSKS = exp(x2, SEPri) in
    new ResM: bitstring;
    event InitS(exp(g, LPri));
    event sendResM(ResM);
    out(c, (encG(SEPub, InitKS), enc(ResM, FSKS)));
    event EndS(x1);
    phase 1; out(c, FSKS)
```

Το πρώτο βήμα του Server είναι να παραλάβει το πρώτο μήνυμα του Client, ως μεταβλητή x1, και αν αυτό είναι το ατελές μήνυμα Hello, στέλνει πέντε στοιχεία που αντιστοιχούν στις πέντε μεταβλητές που αναφέραμε στη διεργασία Client. Τα στοιχεία που στέλνει είναι το LPub= exp(g, LPri), την υπογραφή του χρησιμοποιώντας το exp(g, LPri) και το skS, το πιστοποιητικό CertS και το Token που είναι bitstrings, και το δημόσιο κλειδί pkS. Έπειτα περιμένει να λάβει τα exp(g, CEPri) και enc(ReqM, InitKC) μέσω των μεταβλητών x2, x3. Την x2 την χρησιμοποιεί μαζί με το LPri στον υπολογισμό του InitKS τύπου G, όπως και μαζί με το SEPri στον υπολογισμό του FSKS τύπου G. Την x3 την αποκρυπτογραφεί μέσω του InitKS για να δει αν περιέχει την αίτηση ReqM. Αφού έχει υπολογίσει το SEPub από το SEPri, στέλνει τα κρυπτογραφημένα

SEPub, ResM που χρησιμοποιούν κλειδιά κρυπτογράφησης τα InitKS, FSKS αντίστοιχα.

```
process
        new InitKC: G;
        new InitKS: G;
        new skS: skey;
        let pkS = pk(skS) in
        out(c,pkS); ( (!Client(pkS, InitKC)) | (!Server(skS, pkS, InitKS)))
```

Αυτή είναι η διεργασία που υλοποιεί το πρωτόκολλο. Αρχικοποιούνται οι τιμές InitKC, InitKS, skS και υπολογίζεται το pkS από την συνάρτηση pk(skS),το οποίο στέλνεται στο δημόσιο κανάλι c. Στην συνέχεια εκτελούμε παράλληλα τις άπειρες αντιγραφές των διεργασιών Client και Server.

Παρατηρούμε ότι στις διεργασίες Client και Server συναντάμε διάφορα events. Τα events χρησιμοποιούνται για να ελέγξουμε ιδιότητες αυθεντικοποίησης μέσω των κατάλληλων queries, όπως είδαμε συνοπτικά στο κεφάλαιο της ProVerif. Για παράδειγμα το event InitC(CHLO) υποδηλώνει ότι ο Client πιστεύει ότι είναι έτοιμος να τρέξει το πρωτόκολλο με τον Server έχοντας χρησιμοποίηση την τιμή CHLO. Το event EndS(x1) υποδηλώνει ότι ο Server πιστεύει ότι έχει ολοκληρώσει το πρωτόκολλο χρησιμοποιώντας την x1. Το event sendReqM(ReqM) υποδηλώνει ότι ο Client πιστεύει ότι έχει στείλει το ReqM στον Server και αντίστοιχα το event acceptReqM(ReqMx) από την μεριά του Server ότι έχει λάβει το ReqM από τον Client. Παρόμοια εξήγουνται και τα υπόλοιπα γεγονότα InitS(exp(g, LPri)), EndC(x1), sendResM(ResM), acceptResM(ResMx). Δεδομένων αυτών, τα ερωτήματα που μπορούν να τεθούν για να αποδείξουν ιδιότητες αυθεντικοποίησης είναι τα παρακάτω

query x: bitstring; inj - event(acceptReqM(x)) $\implies$
inj - event(sendReqM(x))
query x: bitstring; inj - event(acceptResM(x)) $\implies$
inj - event(sendResM(x))
query x: bitstring; inj - event(EndS(x)) $\implies$ inj - event(InitC(x))
query x: bitstring; inj - event(EndC(x)) $\implies$ inj - event(InitS(x))

Τα δύο πρώτα ερωτήματα αφορούν την ιδιότητα ότι για να ληφθούν τα ReqM, ResM από τους Server, Client πρέπει πρώτα να έχουν σταλεί από τους Client,

Server αντίστοιχα. Το τρίτο και τέταρτο ερώτημα αφορούν τους τερματισμούς του κάθε συμμετέχοντα, εφόσον έχει ξεκινήσει η εκτέλεση του πρωτοκόλλου με τις κατάλληλες τιμές.

Ακόμα μπορούμε να ελέγξουμε ιδιότητες που αφορούν την forward secrecy. Στην διεργασία Server η τελευταία γραμμή περιέχει την εντολή phase 1, χωρίζοντας την διεργασία σε φάσεις. Η εντολή out(c, FSKS) που έπεται έχει ισχύ μετά από αυτό το σημείο. Με αυτόν τον τρόπο γνωστοποιούμε το FSKS στον εχθρό από ένα σημείο και μετά και θέλουμε να ελέγξουμε αν αυτή η διαρροή επιτρέπει στον εχθρό να χρησιμοποιήσει το FSKS για τους υπολογισμούς του στην προηγούμενη φάση. Ο έλεγχος γίνεται με την εντολή: query attacker(new FSKS) phase 0
Το ίδιο μπορούμε να κάνουμε και για το κλειδί FSKC.

5.3 QUIC στην Verifpal

Η υλοποίηση του πρωτοκόλλου στην Verifpal αποτελεί ενά πολύ χαρακτηριστικό παράδειγμα του βασικού στόχου της, που είναι να υλοποιείται όπως περιγράφεται σε κάποια φυσική γλώσσα. Θα διαπιστώσουμε ότι η περιγραφή που κάναμε στην αρχή του κεφαλαίου για το πρωτόκολλο QUIC έχει ελάχιστες διαφορές με τον τρόπο που αυτό γράφεται στην Verifpal. Ακολουθείται η σειριακή ανταλλαγή μηνυμάτων, όπως φαίνεται στο 5.1, με την χρήση των κατάλληλων primitives που διατίθενται για την επεξεργασία και αποστολή των μηνυμάτων.

Θα χρησιμοποιηθούν τα primitives AEAD_ENC, AEAD_DEC, SIGN, SIGNVERIF, τα οποία τα έχουμε παρουσιάσει στο προηγούμενο κεφαλαίο. Θα δούμε επίσης την χρήση των εξισώσεων στη δημιουργία νέων τιμών, κατασκευάζοντας με αυτόν τον τρόπο και τα κλειδιά Diffie - Hellman. Για λόγους ευκολίας και σύγκρισης, χρησιμοποιούνται αρκετά ίδια ονόματα με την υλοποίηση της ProVerif.

```
principal client [
        generates CHLO
        knows public c0
        ]
```

```
    client → server: [CHLO]

principal server [
          generates CertS, Token, LPri
          knows public c0
          knows private skS
          pkS = G^skS
          LPub = G^LPri
          ssign = SIGN(skS, LPub)
          ]

   server → client: [CertS], Token, LPub, ssign, [pkS]

principal client[ generates CEPri, ReqM
          _=SIGNVERIF(pkS, LPub, ssign)?
          CEPub = G^CEPri
          InitKC = LPub^CEPri
          e_ReqM = AEAD_ENC(InitKC, ReqM, c0)
          ]

   client → server: CEPub, e_ReqM

principal server [ generates SEPri, ResM
          InitKS = CEPub^LPri
          SEPub = G^SEPri
          FSKS = CEPub^SEPri
          e_ReqMx = AEAD_DEC(InitKS, e_ReqM, c0)?
          e_ResM = AEAD_ENC(FSKS, ResM, c0)
          e_SEPub = AEAD_ENC(InitKS, SEPub, c0)
          ]

   server → client: e_ResM, e_SEPub

principal client[ SEPubs = AEAD_DEC(InitKC, e_SEPub, c0)?  FSKC =
          SEPubs^CEPri _= AEAD_DEC(FSKC, e_ResM, c0)?
          ]
```

Όπως παρατηρούμε, η υλοποίηση του πρωτοκόλλου στην Verifpal είναι μια σειρά βημάτων, με κάθε βήμα να είναι είτε αποστολή μηνύματος από τον client στον server ή αντίστροφα, είτε μια σειρά "εσωτερικών" ενεργειών από κάποιον από τους δύο συμμετέχοντες που μπορεί να αφορούν την δημιουργία και τον υπολογισμό σταθερών ή δηλώσεις σχετικά με τη γνώση τους (με την

εντολή knows). Εφόσον η αρχική περιγραφή του πρωτοκόλλου ταιριάζει στην περιγραφή της συγκεκριμένης υλοποίησης και έχοντας ήδη καταγράψει τα ονόματα που χρησιμοποιούμε και τον ρόλο τους, θα επισημάνουμε μόνο ορισμένα που αφορούν τον παραπάνω κώδικα χωρίς να αναλύσουμε κάθε βήμα. Παρατηρούμε ότι συγκεκριμένες σταθερές περικλείονται από brackets κατά την αποστολή τους ([CLHO], [CertS], [pkS]). Όπως έχουμε ήδη αναφέρει, αυτές οι σταθερές ονομάζονται προστατευμένες (guarded) και μοντελοποιούν την ιδιότητα του να είναι γνωστές στον εχθρό κατά την αποστολή τους σε ένα δημόσιο κανάλι αλλά να μην μπορούν να υποστούν τροποποίηση από έναν ενεργητικό εχθρό. Έτσι ο παραλήπτης ενός τέτοιο μηνύματος αποδέχεται ότι προέρχεται από ασφαλή πηγή.
Τα primitives AEAD_ENC, AEAD_DEC, όπως έχουμε δει, πέρα από το κλειδί κρυπτογράφησης και αποκρυπτογράφησης χρησιμοποιούν κάποια επιπρόσθετα δεδομένα και το ρόλο αυτό τον παίζει η σταθερά c0, η οποία είναι γνωστή και στους δύο συμμετέχοντες. Βλέπουμε επιπλέον τη χρήση εξισώσεων για τη δημιουργία των κλειδιών Diffie - Hellman, όπου κάθε σταθερά - εξίσωση είναι το αποτέλεσμα της εκθετικής πράξης δύο άλλων σταθερών. Τέλος, σε τρεις περιπτώσεις παρατηρούμε τη χρήση του ερωτηματικού ?, παράδειγμα το _=SIGNVERIF(pkS, LPub, ssign)? και το e_ReqMx = AEAD_DEC(InitKS, e_ReqM, c0)?. Συγκεκριμένα, έχουμε επαλήθευση των κανόνων που ορίζουν τα σχετικά primitives, ελέγχοντας αν τα inputs συμφωνούν με αυτούς τους κανόνες. Σε περίπτωση αποτυχίας διακόπτεται η εκτέλεση του πρωτοκόλλου.

Τα ερωτήματα που μπορούμε να θέσουμε αφορούν τα μηνύματα ResM και ReqM.
Με την εντολή: confidentiality? ResM
ελέγχουμε αν ο εχθρός έχει στην κατοχή του το ResM.
Με την εντολή: authentication? server→client: e_ResM
ελέγχουμε αν το e_ResM, που είναι το κρυπτοκείμενο του ResM που λαμβάνει, αποκρυπτογραφεί και αυθεντικοποιεί ο Client, έχει όντως σταλεί από τον Server.
Αντίστοιχες εντολές μπορούμε να δώσουμε και για το ReqM.
Επίσης μπορούμε να ελέγξουμε και ιδιότητες forward secrecy, με την διαρροή των κλειδιών FSKS, FSKC, χωρίζοντας το πρωτόκολλο σε φάσεις και δίνοντας τις εξής εντολές:
phase[1]
principal client [leaks FSKC]
principal server [leaks FSKS]

5.4 Σύγκριση

Σε γενικές γραμμές έχουμε αναφέρει τα χαρακτηριστικά του κάθε εργαλείου. Η ProVerif χρησιμοποιεί ως γλώσσα μια διάλεκτο του Εφαρμοσμένου Π - Λογισμού, άρα η βάση της είναι μια αυστηρά θεμελιωμένη λογική με πλούσιο θεωρητικό υπόβαθρό. Η μετάφραση ενός πρωτοκόλλου από την φυσική γλώσσα ή κάποια γλώσσα προγραμματισμού στην ProVerif δεν είναι τετριμμένη, αντίθετα πρόκειται για ένα απαιτητικό εγχείρημα καθώς προφανώς τα περισσότερα πρωτόκολλα δεν είναι προσανατολισμένα στην λογική της ProVerif. Επιπλέον, ο χρήστης οφείλει να είναι εξοικειωμένος με τα χαρακτηριστικά και τις δυνατότητες του Εφαρμοσμένου Π - Λογισμού. Η ανάλυση της ProVerif είναι αποτελεσματική με αρκετές ιδιότητες ασφαλείας, ειδικά στο κομμάτι της μυστικότητας ενός όρου θα εντοπίσει όλους τους όρους που έχει στην κατοχή του ο εχθρός. Από την άλλη, οι προτάσεις Horn που δημιουργούνται συνοδεύονται με ορισμένες προσεγγίσεις που σπάνια μπορεί να οδηγήσουν σε λανθασμένα αποτελέσματα όπως false attacks.
Η Verifpal χρησιμοποιεί μια γλώσσα εξαιρετικά φιλική προς το χρήστη και αρκετά τετριμμένη στη μετάφραση ενός πρωτοκόλλου. Τα μειονεκτήματα αυτής της προσέγγισης είναι οι περιορισμοί στους ελέγχους ασφαλείας που μπορούν να γίνουν, ενώ πολλές φορές τα αποτελέσματα πρέπει να συγκριθούν με άλλα πιο αυστηρά θεμελιωμένα εργαλεία, όπως η ProVerif, για να ελεγχθεί η εγκυρότητά τους.
Στο [9] εκτελείται το πρωτόκολλο QUIC στις εκδόσεις ProVerif 2.00 και Verifpal 0.18.1 θέτοντας τα queries που είδαμε πιο πάνω, τα οποία σχετίζονται με συγκεκριμένες ιδιότητες ασφαλείας. Παρουσιάζονται οι απαντήσεις που δόθηκαν καθώς και ο χρόνος που χρειάστηκε το κάθε εργαλείο για να απαντήσει. Παρατηρούμε ότι σε όλες τις ιδιότητες οι απαντήσεις είναι κοινές, με την ProVerif όμως να είναι σημαντικά πιο γρήγορη στην πλειονότητα.

Ιδιότητα 1η: Ο Client και ο Server συμφωνούν στην εγκυρότητα του ResM μετά τον επιτυχή τερματισμό του πρωτοκόλλου
ProVerif: Σωστό - <1s, Verifpal: Σωστό - 7s
Το ResM επιβεβαιώνεται ότι μπορεί να σταλεί μόνο από τον Server στον Client.

Ιδιότητα 2η: Ο Client και ο Server συμφωνούν στην εγκυρότητα του ReqM μετά τον επιτυχή τερματισμό του πρωτοκόλλου
ProVerif: Λάθος - <1s, Verifpal: Λάθος - 1s
Δεν μπορεί να επιβεβαιωθεί ότι ο Server λαμβάνει το ReqM από τον

Client. Αυτό οφείλεται στην τιμή CEPub που στέλνεται χωρίς κάποια κρυπτογράφηση σε δημόσιο κανάλι και έτσι ο εχθρός μπορεί να την χρησιμοποιήσει για να προσποιηθεί τον Client.

Ιδιότητα 3η: Ο Client και ο Server συμφωνούν ότι επικοινωνούν μεταξύ τους μετά τον τερματισμό του πρωτοκόλλου
ProVerif: Λάθος - <1s, Verifpal: Λάθος - 1s
Αυτό το αποτέλεσμα συνεπάγεται από την αποτυχία της δεύτερης ιδιότητας.

Ιδιότητα 4η: Ο εχθρός δεν μπορεί να έχει στην κατοχή του το ReqM
ProVerif: Σωστό - <1s, Verifpal: Σωστό - 5s
Δεν εντοπίζεται κάποια επίθεση που μπορεί να δώσει στον εχθρό το ReqM.

Ιδιότητα 5η: Ο εχθρός δεν μπορεί να έχει στην κατοχή του το ResM
ProVerif: Λάθος - 1s, Verifpal: Λάθος - <1s
Εφόσον ο εχθρός έχει στην κατοχή του το CEPub, το στέλνει μαζί με το κρυπτογραφημένο ReqM για να λάβει από τον Server το ResM, καθώς ο Server θεωρεί ότι μιλάει με τον Client.

Ιδιότητα 6η: Η διαρροή των FSKC, FSKS στην επόμενη φάση δεν επηρέαζει την προηγούμενη φάση.
ProVerif: Σωστό - 1s, Verifpal: Σωστό - 7s
Διαπιστώνεται ότι ο εχθρός δεν μπορεί να χρησιμοποιήσει με κάποιον τρόπο τα FSKC, FSKS, αφού του έχουν γνωστοποιηθεί με το πέρας της πρώτης φάσης, για να υποκλέψει κάποιο αρχικό μήνυμα.

Κεφάλαιο 6

Συμπεράσματα

Η μελέτη μας αφορούσε τη χρήση τυπικών μεθόδων στην ανάλυση κρυπτογραφικών πρωτοκόλλων. Ασχοληθήκαμε με δύο εργαλεία εφαρμογής του συμβολικού μοντέλου, την ProVerif και την Verifpal και εμβαθύναμε στο θεωρητικό υπόβαθρο του ενός, μελετώντας τον Εφαρμοσμένο Π - Λογισμό. Εργαλεία που βασίζονται στο συμβολικό μοντέλο έχουν αξιοποιηθεί και αξιοποιούνται στην ανάλυση διαδεδομένων πρωτοκόλλων, με ευρεία χρήση στην καθημερινή μας ζωή. Στην βιβλιογραφία καταγράφονται πολλές περιπτώσεις πρωτοκόλλων όπου εντοπίστηκαν ελαττώματα που μπορούσαν να οδηγήσουν σε επιθέσεις, ιδιαίτερα σε πρωτόκολλα που θεωρούνταν ασφαλή. Επιπρόσθετα, είναι συχνή και η επιδιόρθωση των ελαττωμάτων και γενικά η βελτιστοποίηση των διεργασιών του πρωτοκόλλου, εξετάζοντας τον τρόπο που κάθε εργαλείο ανιχνεύει ένα ελάττωμα ή μια πιθανή επίθεση.
Ένα σημαντικό ζήτημα που περιορίζει το συμβολικό μοντέλο είναι η δυσκολία της μετάφρασης ενός πρωτοκόλλου από τη φυσική γλώσσα ή από κάποια γλώσσα προγραμματισμού σε γλώσσα που βασίζεται στο συμβολικό μοντέλο. Διαπιστώσαμε ότι όταν εργαλεία όπως η Verifpal προσπαθούν να διευκολύνουν τη διαδικασία της μετάφρασης με μια προσιτή γλώσσα τότε παρουσιάζονται προβλήματα με την ανάλυση και την εγκυρότητά τους. Ένας τρόπος να αντιμετωπιστεί αυτό είναι η γραφή των πρωτοκόλλων να κατευθύνεται από το συμβολικό μοντέλο έτσι ώστε κάθε διεργασία να μπορεί να μεταφραστεί και να αναλυθεί. Φυσικά κάτι τέτοιο προϋποθέτει την εξειδίκευση σε λογικές όπως ο Εφαρμοσμένος Π - Λογισμός και θεωρίες όπως οι Άλγεβρες Διεργασιών. Αυτό που δεν πρέπει να παραλείπουμε είναι ότι το συμβολικό μοντέλο είναι αυτό ακριβώς που δηλώνει η δεύτερη λέξη,

δηλαδή μοντέλο. Οι ορισμοί του έχουν παραδοχές και προσεγγίσεις και έτσι είναι ανέφικτη η ταύτιση ενός πρωτοκόλλου στο συμβολικό μοντέλο με ένα ρεαλιστικό πρωτόκολλο.
Στο γενικό συμπέρασμα που καταλήγουμε είναι ότι οι τυπικές μέθοδοι ανάλυσης αποτελούν μια ισχυρή θεωρία με προγραμματιστικά εργαλεία στη διάθεση της κρυπτανάλυσης, που έχουν καταγράψει σημαντικά αποτελέσματα. Ο στόχος είναι η εξέλιξή τους ώστε να αποτελούν μέθοδοι ανάλυσης για όλο και περισσότερα πρωτόκολλα.

Βιβλιογραφία

[1] Gavin Lowe. «Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR». In: *Softw. Concepts Tools* 17 (1996), pp. 93–102. URL: https://api.semanticscholar.org/CorpusID:9626081.

[2] Gérard Berry and Gérard Boudol. «The chemical abstract machine». In: *Theoretical Computer Science* 96.1 (1992), pp. 217–248. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(92)90185-I. URL: https://www.sciencedirect.com/science/article/pii/030439759290185I.

[3] Martín Abadi, Bruno Blanchet, and Cédric Fournet. «The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication». In: *Journal of the ACM* 65 (Sept. 2016). DOI: 10.1145/3127586.

[4] Bruno Blanchet and Ben Smyth. «ProVerif 1.85: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial». In: (Apr. 2011).

[5] Bruno Blanchet. «Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif». In: *Foundations and Trends in Privacy and Security* 1 (Oct. 2016), pp. 1–135. DOI: 10.1561/3300000004.

[6] Bruno Blanchet. *Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif*. Jan. 2014. ISBN: 978-3-319-10081-4. DOI: 10.1007/978-3-319-10082-1_3.

[7] Bruno Blanchet. «Using Horn Clauses for Analyzing Security Protocols». In: *Cryptology and Information Security Series* 5 (Jan. 2011). DOI: 10.3233/978-1-60750-714-7-86.

[8] Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. «Verifpal: Cryptographic Protocol Analysis for the Real World». In: Dec. 2020, pp. 151–202. ISBN: 978-3-030-65276-0. DOI: 10.1007/978-3-030-65277-7_8.

[9] Jingjing Zhang et al. «Formal Analysis of QUIC Handshake Protocol Using Symbolic Model Checking». In: *IEEE Access* 9 (2021), pp. 14836–14848. DOI: 10.1109/ACCESS.2021.3052578.

[10] Mohamud Jimale et al. «Authenticated Encryption Schemes: A Systematic Review». In: *IEEE Access* 10 (Jan. 2022), pp. 1–1. DOI: 10.1109/ACCESS.2022.3147201.

[11] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. «Automated Verification for Secure Messaging Protocols and Their Implementations: A Symbolic and Computational Approach». In: *2017 IEEE European Symposium on Security and Privacy (EuroS'n'P)*. 2017, pp. 435–450. DOI: 10.1109/EuroSP.2017.38.

[12] Bruno Blanchet, Martín Abadi, and Cédric Fournet. «Automated verification of selected equivalences for security protocols». In: *The Journal of Logic and Algebraic Programming* 75.1 (2008). Algebraic Process Calculi. The First Twenty Five Years and Beyond. III, pp. 3–51. ISSN: 1567-8326. DOI: https://doi.org/10.1016/j.jlap.2007.06.002. URL: https://www.sciencedirect.com/science/article/pii/S1567832607000549.

[13] Colin Stirling. «Bisimulation and Language Equivalence». In: (Nov. 2001). DOI: 10.1007/0-306-48088-3_7.

[14] Bruno Blanchet. «Security Protocol Verification: Symbolic and Computational Models». In: *The post*. 2012. URL: https://api.semanticscholar.org/CorpusID:2109089.

[15] Rocco De Nicola. «A gentle introduction to Process Algebras ?» In: 2013. URL: https://api.semanticscholar.org/CorpusID:18837826.

[16] Riccardo Bresciani and Andrew Butterfield. «Weakening the Dolev-Yao model through probability». In: Jan. 2009, pp. 293–297. DOI: 10.1145/1626195.1626265.

[17] Véronique Cortier and Steve Kremer. *Formal Models and Techniques for Analyzing Security Protocols: A Tutorial*. Jan. 2014. ISBN: 9781601989031. DOI: 10.1561/9781601989031.

[18] Cédric Fournet and Georges Gonthier. «A hierarchy of equivalences for asynchronous calculi». In: *J. Log. Algebraic Methods Program.* 63 (1998), pp. 131–173. URL: https://api.semanticscholar.org/CorpusID:3742.

[19] Anguraj Baskar, Ramaswamy Ramanujam, and S. Suresh. «A Dolev-Yao Model for Zero Knowledge». In: Dec. 2009, pp. 137–146. ISBN: 978-3-642-10621-7. DOI: 10.1007/978-3-642-10622-4_11.

[20] Wan Fokkink. *Introduction to Process Algebra*. Jan. 2000. ISBN: 978-3-540-66579-3.

[21] Bruno Blanchet, Vincent Cheval, and Véronique Cortier. «ProVerif with Lemmas, Induction, Fast Subsumption, and Much More». In: *2022 IEEE Symposium on Security and Privacy (SP)*. 2022, pp. 69–86. DOI: 10.1109/SP46214.2022.9833653.

[22] Bruno Blanchet. «The Security Protocol Verifier ProVerif and its Horn Clause Resolution Algorithm». In: *Electronic Proceedings in Theoretical Computer Science* 373 (Nov. 2022), pp. 14–22. DOI: 10.4204/EPTCS.373.2.

[23] Παγουρτζής Αριστείδης and Ζάχος Ευστάθιος. *Υπολογιστική κρυπτογραφία [Προπτυχιακό εγχειρίδιο]*. 2015. ISBN: 978-960-603-276-9. DOI: http://dx.doi.org/10.57713/kallipos-492.

[24] Στεφανέας Πέτρος and Κολέτσος Γεώργιος. *Εφαρμογές της λογικής στην πληροφορική [Προπτυχιακό εγχειρίδιο]*. 2015. ISBN: 978-960-603-368-1. DOI: http://dx.doi.org/10.57713/kallipos-571.

[25] Χίου Χρυσόστομος. *Ανάλυση Πρωτοκόλλων Ηλεκτρονικής Ψηφοφορίας με την ProVerif [Διπλωματική Εργασία]*. 2020. DOI: http://dx.doi.org/10.26240/heal.ntua.18816. URL: https://dspace.lib.ntua.gr/xmlui/handle/123456789/51118.