Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών

Τομέας Μαθηματικών

ΓΚΟΛΦΗ ΑΝΑΣΤΑΣΙΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ :

“Protocol Composition Logic-Εφαρμογή στο SRP”

Επιτροπή : Π.Στεφανέας (επιβλέπων)

Γ.Κολέτσος

Α.Παπαϊωάννου

ΑΘΗΝΑ 2011

Περίληψη / Abstract

Αυτή η διπλωματική αφορά μία εφαρμογή της Protocol Composition Logic (PCL) στο πρωτόκολλο Secure Remote Password (SRP) προκειμένου να καταστεί δυνατή η απόδειξη των ιδιοτήτων ασφαλείας του. Το SRP είναι ένα πρωτόκολλο τύπου password-authentication key agreement. Η PCL είναι μία τυπική λογική που έχει φτιαχτεί για να δηλώνονται και να αποδεικνύονται ιδιότητες ασφαλείας των πρωτοκόλλων δικτύου. Χρησιμοποιείται ένας συγκεκριμένος φορμαλισμός που λέγεται cord calculus και είναι ένας action calculus που περιγράφει τα βήματα των πρωτοκόλλων με δυναμικό τρόπο. Επιπλέον, έχουν καθιερωθεί μερικές ιδιότητες των πρωτοκόλλων προκειμένου να ενισχυθεί το reasoning τους. Το αποδεικτικό σύστημα της PCL αποτελείται από ένα σύνολο αξιωμάτων και κανόνων έτσι ώστε να γίνει εφικτή η σύνθεση των πρωτοκόλλων. Δύο είδη σύνθεσης είναι δυνατό να συμβούν : η παράλληλη σύνθεση και η διαδοχική σύνθεση. Επίσης, υπάρχουν επιπλέον λήμματα και θεωρήματα που υποστηρίζουν τις αποδείξεις των ιδιοτήτων και για τις δύο μεθόδους σύνθεσης.

This thesis, concerns an application of Protocol Composition Logic (PCL) to Secure Remote Password protocol (SRP) in order to prove its security properties. SRP is a password-authenticated key agreement protocol.PCL is a formal logic for stating and proving security properties of network protocols. It is used a specific formalism called cord calculus which is an action calculus describing protocol steps in a dynamic way.In addition, some protocol properties are established in order to enhance protocol reasoning. The proof system of PCL consists of a set of actioms and rules so that protocol composition to become feasible. In fact, two kinds of composition are possible through PCL:Parallel Composition and Sequential Composition. There exist additional lemmas and theorems which support the proof of properties for both methods of composition.

Θα ήθελα να ευχαριστήσω θερμά τον κ. Π.Στεφανέα τόσο για την επιστημονική επιμέλεια αυτής της διπλωματικής όσο και για τη συνολική του στήριξη καθ' όλη τη διάρκεια εκπόνησής της. Επίσης, νιώθω την ανάγκη να ευχαριστήσω τον κ. Ι.Ουρανό για την επιλογή αυτού του υπέροχου θέματος που μου κίνησε το ενδιαφέρον και για περαιτέρω έρευνα. Τέλος, θα ήθελα να εκφράσω τις ευχαριστίες μου στους κ. Γ.Κολέτσο και Α.Παπαϊωάννου για τις επιστημονικές τους επισημάνσεις.

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 4

Αποδεικτικές μέθοδοι της PCL

ΚΕΦΑΛΑΙΟ 5

Secure Remote Password Protocol (SRP)

ΚΕΦΑΛΑΙΟ 6

Εφαρμογή της PCL στο SRP

ΚΕΦΑΛΑΙΟ 7

Συμπεράσματα και Προτάσεις για περαιτέρω έρευνα

Ευρετήριο πινάκων

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Στην παρούσα διπλωματική θα αναλυθεί η Protocol Compostion Logic , μία λογική σύνθεσης πρωτοκόλλων ασφαλείας που αναπτύσσεται στο πανεπιστήμιο Stanford από το εργαστήριο «Stanford Security Lab».

Το δομικό στοιχείο δράσης της είναι ο cord calculus ο οποίος είναι εμπνεύσμένος από τον φορμαλισμό των strand spaces[15] . Η εξέλιξη του έγκειται στο ότι έπρεπε να δοθεί η δυνατότητα για μία δυναμική ανάλυση του reasoning και των υπολογισμών και όχι στατική όπως στην περίπτωση των strand spaces. Προς την κατεύθυνση αυτή λοιπόν τα strand spaces ενισχύθηκαν με λειτουργική σημασιολογία του τύπου της chemical abstract machine[1] και έτσι προέκυψε ο cord calculus, που είναι ένας σχετικά απλός process calculus.

Στο δεύτερο κεφάλαιο αναπτύσσεται ο cord calculus στο σύνολό του, δηλαδή τα δομικά του μέρη και ο τρόπος με τον οποίο περιγράφει με δυναμικό τρόπο τα πρωτόκολλα (actions κτλ). Στη συνέχεια εξηγείται πώς περιγράφει πλήρη τρόπο τα πρωτόκολλα και τις πιθανές εκτελέσεις τους και τέλος αναλύεται ο cord category και οι ιδιότητες των πρωτοκόλλων.

Στο τρίτο κεφάλαιο αναπτύσσεται το συντακτικό της PCL , η σημασιολογία της και το αποδεικτικό της σύστημα.

Στο τέταρτο κεφάλαιο αναλύονται διεξοδικά οι αποδεικτικές μέθοδοι της PCL , δηλαδή η σύνθεση τα θεωρήματά της και η abstraction-refinement μέθοδος.

Στο πέμπτο κεφάλαιο παρουσιάζεται λεπτομερώς το πρωτόκολλο Secure Remote Password (SRP), το κρυπτογραφικό του υπόβαθρο και η αντίσταση του σε πιθανές επιθέσεις.

Στο έκτο κεφάλαιο αναπτύσσεται μία εφαρμογή της PCL στο πρωτόκολλο SRP.

Στο έβδομο κεφάλαιο εκτίθενται τα συμπεράσματα αυτής της εφαρμογής καθώς και προτάσεις για περαιτέρω έρευνα.

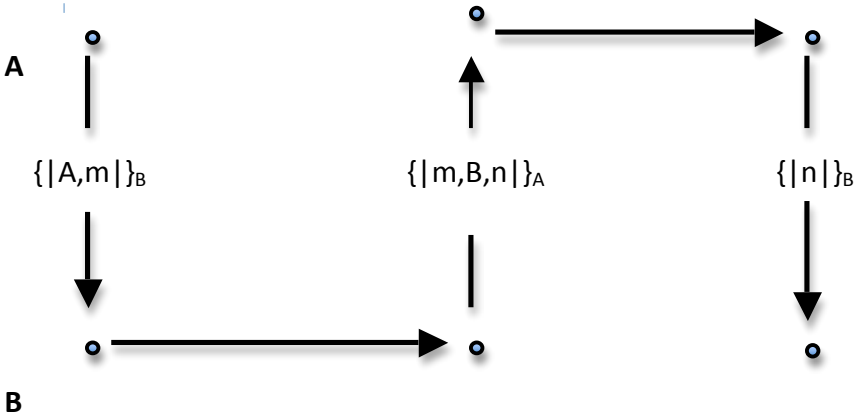
Γενικά, η PCL φαίνεται να είναι μία πολλά υποσχόμενη προσέγγιση στο πεδίο των formal proofs. Ήδη έχουν γίνει πολλές και σημαντικές εφαρμογές σε πρωτόκολλα και το “Stanford Security Lab” έχει αποσπάσει βραβεία γι’ αυτή του τη δουλειά.Είναι όμως ακόμα ένα ανοιχτό πεδίο έρευνας και διαρκώς υφίσταται βελτιώσεις και αναβαθμίσεις.

ΚΕΦΑΛΑΙΟ 2

Cord Calculus

2.1.Εισαγωγή

Τα πρωτόκολλα και τα συστατικά τους μέρη αναπαρίστανται μέσω ενός φορμαλισμού που ονομάζεται *cord calculus* [10] και είναι εμπνευσμένος από τα strand spaces [15] καθώς και από την ανεπίσημη αλλά ευρέως διαδεδομένη στο security χρήση «μηνυμάτων και βελών».Παρακάτω βλέπουμε την αναπαράσταση του NSL με μηνύματα και βέλη :



καθώς και την αναπαράσταση του με τη χρήση του cord calculus :

A = [(νm) <{|A,m|}$_B$> (u) (u/{|m,B,υ|}$_{\bar{A}}$) <{|υ|}$_B$>]

B = [(x) (x/{|Y,z|}$_{\bar{B}}$) (νn) <{|z,B,n|}$_Y$> (ω) (ω/{|n|}$_{\bar{B}}$)]

Επιπλέον, υπάρχει η δυνατότητα να περιέχονται μεταβλητές οι οποίες επιδέχονται αντικατάστασης από πολλούς τύπους τιμών. Έτσι, μαζί με τη βοήθεια ενός μηχανισμού ταιριάσματος και υποκατάστασης μπορεί να επιτευχθεί λειτουργική σημασιολογία που οδηγεί όχι μόνο σε στατική αλλά και σε δυναμική ανάλυση των πρωτοκόλλων.

2.2.Όροι, Δράσεις, Strands και Cords

Ένα από τα δομικά στοιχεία του cord calculus είναι τα *ονόματα* (N). Αυτά αποτελούνται είτε από μεταβλητές ονόματος $\hat{X}$, είτε από σταθερές ονόματος $\hat{A}$.Υπάρχουν επίσης και οι *agents* (X και A εαν πρόκειται για μεταβλητές ή σταθερές αντίστοιχα).Αυτή η διαφοροποίηση έγκειται στο εξής: ένας principal ,έστω $\hat{A}$,μπορεί να έχει ξεκινήσει ένα πρωτόκολλο με κάποιον άλλο, έστω $\hat{B}$, ενώ ταυτόχρονα να παίζει διαφορετικό ρόλο στο ίδιο πρωτόκολλο που ομώς αυτή τη φορά εχει ξεκινήσει από κάποιον τρίτο έστω $\hat{C}$. Προκειμένου, λοιπόν, να μην προκαλείται σύγχιση δίνουμε ονόματα στις διαφορετικές εκτελέσεις (threads), και χρησιμοποιούμε τα agents (εδώ A) για να σχεδιάσουμε ενα συγκεκριμένο thread που εκτελείται απο τον συγκεκριμένο principal (εδώ $\hat{\mathrm{A}}$).

Ένα άλλο συστατικό στοιχείο του cord calculus είναι τα *κλειδιά (keys)*, τα οποία αποτελούνται από τα βασικά κλειδιά K_0 και τα αντίστροφά τους $\overline{\mathrm{K}_0}$. Τα βασικά κλειδιά μπορεί να είναι είτε σταθερές κλειδιού k, είτε μεταβλητές κλειδιού y, είτε

ονόματα N όπως αυτά ορίστηκαν παραπάνω (τα τελευταία χρησιμοποιούνται όταν αναφερόμαστε στα κλειδιά του αντίστοιχου principal).

Οι *όροι (terms)*, οι οποίοι συνιστούν ένα από τα πιο σημαντικά δομικά συστατικά του cord calculus σχηματίζονται από μεταβλητές όρων x, σταθερές όρων c, ονόματα N, threads P, κλειδιά K, nonces n και κατασκευαστές p οι οποίοι περιέχουν τουλάχιστον την παράθεση? (tupling) όρων t,t , την κρυπτογράφηση δημοσίου κλειδιού $\{|t|\}_K$ (όπου ο συμβολισμός $\{|\_|\}_\_$ αναπαριστά την ασύμμετρη κρυπτογράγηση) και την υπογραφή $\{|t|\}_{\overline{K}}$. Οι όροι οι οποίοι δεν περιέχουν ελεύθερες μεταβλητές μπορούν να σταλούν ως μηνύματα.

Οι *δράσεις(actions)* είναι ο τρόπος με τον οποίο περιγράφονται οι διάφορες ενέργειες που λαμβάνουν χώρα κατά την εκτέλεση των πρωτοκόλλων και περιέχουν όρους. Υπάρχει η κενή δράση $\in$, κατά την οποία –όπως προδίδει και το όνομά της- δεν εκτελείται καμία δράση, η αποστολή ενός όρου <t> , η λήψη ενός όρου σε μία μεταβλητή (x) , ο έλεγχος συμβατότητας σε σχέση με κάποιο πρότυπο (pattern matching) (t/q(x)) και η δημιουργία μίας νέας τιμής (vx).

Το συντακτικό των προτύπων (patterns) φαίνεται στον πίνακα που ακολουθεί:

βασικοί όροι	b ::= x|c|N|K	επιτρεπόμενοι βασικοί όροι για τα πρότυπα
βασικά πρότυπα	p ::= b,…,b	πρότυπο παράθεσης(tuple pattern)
πρότυπα	q ::= p	βασικό πρότυπο
	$\{\|p\|\}_{\overline{K}}$	πρότυπο αποκρυπτογράφησης

Πίνακας 2.1. Πρότυπα

Έστω ένα πρότυπο $p(x_1,\ldots,x_n)$ με n μεταβλητές και ένας όρος $t=t_1,\ldots,t_n$. Τότε το $p(t)=[\ t_1,\ldots,t_n / x_1,\ldots,x_n\ ]p$ είναι όρος ο οποίος έχει προκύψει από την αντικατάσταση των $x_1,\ldots,x_n$ στο p από τις $t_1,\ldots,t_n$. Ενδεικτικά παρατίθενται μερικά παραδείγματα προτύπων:

$p_1(x_1,x_2) = x_1,x_2$

$p_2(x_1,x_2) = x_1,A,x_2$

$q_1(x_1,x_2) = \{| p_1(x_1,x_2)|\}_{\overline{K}} = \{|x_1,x_2|\}_{\overline{K}}$

$q_2(x_1,x_2) = \{| p_2(x_1,x_2)|\}_{\overline{K}} = \{|x_1,A,x_2|\}_{\overline{K}}$

όπου τα p_i και q_i , i=1,2 είναι πρότυπα με δύο ορίσματα. Το πρώτο είναι μια απλή περίπτωση προτύπου παράθεσης των ορισμάτων του, το δεύτερο είναι παρόμοιο με το πρώτο με μια επιπλέον σταθερά A και τα δύο τελευταία είναι πρότυπα αποκρυπτογράφησης.

Η αποκρυπτογράφηση (δημοσίου κλειδιού) αναπαρίσταται ως μια δράση τύπου pattern matching ($\{|x|\}_A$ /$\{|z|\}_{\overline{A}}$) , όπου πρώτα γίνεται ο έλεγχος συμβατότητας του $\{|x|\}_A$ με το πρώτυπο $\{|z|\}_A$ και στη συνέχεια,εάν δεν προκύψει πρόβλημα, αποκρυπτογραφείται το $\{|x|\}_A$ με χρήση του ιδιωτικού κλειδιού $\overline{A}$. Για παράδειγμα, η αποκρυπτογράφηση ($\{|x,C,y|\}_A$ /$\{|z,w|\}_{\overline{A}}$) δεν είναι εφικτή αφού το προς αποκρυπτογράφηση μήνυμα $\{|x,C,y|\}_A$ δεν είναι συμβατό με το πρότυπο $\{|z,w|\}_A$.

Τα ***strands*** είναι λίστες από δράσεις και ορίζονται συντακτικά (με αναδρομικό τρόπο) ως S ::= αS|α όπου α είναι δράσεις. Κάθε strand δηλαδή αναπαριστά μια ακολουθία δράσεων ενός thread. Για παράδειγμα, το strand $(vx)\langle x\rangle_A$ μας λέει ότι ο A δημιουργεί μία νέα τιμή στη μεταβλητή x και στη συνέχεια τη στέλνει με τη μορφή μηνύματος.

Δύο strands μπορεί να μην είναι διαφορετικά, παρόλο που έχουν τις δράσεις τους σε διαφορετική σειρά. Αυτό συμβαίνει όταν μια μεταβλητή που είναι δεσμευμένη σε μία δράση δεν εμφανίζεται ως ελεύθερη μεταβλητή σε άλλη δράση. Για παράδειγμα, το strand $(x)(y)\langle x,y\rangle$ έχει ως inputs τα x και y και στη συνέχεια στέλνει το ζεύγος x,y.Το ίδιο ισχύει και για το strand $(y)(x)\langle x,y\rangle$ (στα δίκτυα δεν ισχύει η διατήρηση της σειράς αποστολής αφού λειτουργούν με ασύγχρονο τρόπο –θα αναλυθεί σε επόμενο κεφάλαιο διεξοδικότερα-). Επομένως τα δύο προηγούμενα strands είναι ισοδύναμα.

Δύο strands S και T είναι ανεξάρτητα όταν δεν μπορούν να αλληλεπιδράσουν οι μεταβλητές τους. Συμβολίζουμε την σχέση ισοδυναμίας μεταξύ των strands με ~ .Άρα

$$ST \sim TS \Leftrightarrow \begin{cases} FV(S) \cap BV(T) = \varnothing \\ FV(T) \cap BV(S) = \varnothing \end{cases}$$

Όπου με FV και BV ενοούμε αντίστοιχα τα σύνολα των ελεύθερων και των δεσμευμένων μεταβλητών του strand που βρίσκεται στην παρένθεση.Παρατηρούμε ότι, αφού η διάδοση της τιμής μιας μεταβλητής γίνεται αντικαθιστώνας την σε κάθε εμφάνιση της μεταβλητής , τότε πράγματι οι παραπάνω συνθήκες εξασφαλίζουν την ανεξαρτησία των strands S και T.

Οι δράσεις (x) , (t/q(x)) και (vx) , όταν επενεργήσουν σε ένα strand S, δεσμεύουν τη μεταβλητή x που βρίσκεται σε αυτό. Στην περίπτωση της δράσης <t> όμως, οι μεταβλητές που περιέχονται στον όρο t προστίθενται στο σύνολο των ελεύθερων μεταβλητών του S. Πιο αναλυτικά :

$$\mathrm{FV}(<t>S) = \mathrm{FV}(t) \cup \mathrm{FV}(S)$$
$$FV((x)S) = FV(S) \setminus \{x\}$$
$$FV((t/q(x))S) = FV(t) \cup FV(S) \setminus \{x\}$$
$$FV((vx)S) = FV(S) \setminus \{x\}$$

$$BV(<t>S) = BV(S)$$
$$BV((x)S) = BV(S) \cup \{x\}$$
$$BV((t/q(x))S) = BV(S) \cup \{x\}$$
$$BV((vx)S) = BV(S) \cup \{x\}$$

Η σχέση ισοδυναμίας $\approx$ είναι η ελάχιστη ανακλαστική, μεταβατική και κλειστή ως προς την ~ σχέση. Έτσι, τα strands S και T είναι $\approx$-ισοδύναμα αν και μόνο αν το ένα μπορεί να προκύψει από το άλλο μετά από μετονομασία των δεσμευμένων μεταβλητών και μετάθεση των δράσεων στα πλαίσια των δεσμευμένων μεταβλητών του. Πρέπει να παρατηρήσουμε ότι με αυτό τον τρόπο διατηρούνται όλες οι ελεύθερες μεταβλητές των strands. Έτσι, αν ***S*** είναι το σύνολο όλων των δυνατών strands,

ορίζουμε τα *cords* ως τις κλάσεις ισοδυναμίας των strands modulo τη σχέση ισοδυναμίας $\approx$, δηλαδή

$$\boldsymbol{C} = \boldsymbol{S} / \approx$$

Το κάθε cord τα εσωκλείεται σε [ ] .Έτσι, $[S]_{\approx}=\{S' | S' \approx S\}$.Η κλάση ισοδυναμίας για το strand S του thread A γράφεται $[S]_A$, το κενό cord [] και τέλος υποθέτουμε ότι $[]=[]_X$ για κάθε thread X.

Στον ακόλουθο πίνακα συνοψίζονται οι παραπάνω ορισμοί που αφορούν τα δομικά στοιχεία του cord calculus :

ονόματα	N ::=	$\hat{X}$	μεταβλητή ονόματος
		$\hat{A}$	σταθερά ονόματος
threads	P ::=	X	μεταβλητή thread
		A	σταθερά thread
βασικά κλειδιά	K_0 ::=	k	μεταβλητή κλειδιού
		y	σταθερά κλειδιού
		N	όνομα
κλειδιά	K ::=	K_0	βασικό κλειδί
		$\overline{K_0}$	αντίστροφο κλειδί
όροι	t ::=	x	όρος μεταβλητής
		c	όρος σταθεράς
		N	όνομα
		P	thread
		K	κλειδί
		t,t	παράθεση όρων (tupling)
		$\{\|t\|\}_K$	όρος κρυπτογραφημένος με το κλειδί K
		$\{\|t\|\}_{\overline{K}}$	όρος υπογεγραμμένος με το κλειδί $\overline{K}$
δράσεις (actions)	a ::=	$\in$	κενή δράση
		<t>	αποστολή του όρου t
		(x)	λήψη όρου στη μεταβλητή x
		(vx)	δημιουργία νέου όρου
		$(t/ q(x_1,\ldots,x_n))$	έλεγχος συμβατότητας του όρου t με τον $q(x_1,\ldots,x_n)$
Strands	S ::=	αS|α	λίστα από δράσεις

Πίνακας 2.2. Όροι,Δράσεις και Strands

Όλοι οι παραπάνω ορισμοί βρίσκονται στα [2,4,10,11,12]

2.3.Cord Spaces, Reactions και Runs

Cord Space είναι ένα μη κενό πολυσύνολο από cords C={$[S_1]_{A_1}$,$[S_2]_{A_2}$,…,$[S_k]_{A_k}$ } , όπου κάθε S_i είναι ένα μη κενό strand. Ο κενός cord space είναι ο {[]} που γράφεται και ως []. Οι cord spaces αποτελούν ελεύθερα αντιμεταθετικά μονοειδή (C,⊗,[]) που έχουν δημιουργηθεί από cords και η διμελής πράξη ⊗ είναι η ένωση πολυσυνόλων και [] το μοναδιαίο στοιχείο , δηλαδή C⊗ []=[]⊗ C=C [10].

Στους cord spaces η λειτουργική σημασιολογία επιτυγχάνεται μέσω των *reaction steps*, τα οποία διαφέρουν ανάλογα με τις δράσεις τις οποίες περιέχουν. Τα βασικά reaction steps είναι τα ακόλουθα:

(1) [S(x)S'] ⊗ [T<t>T'] ⊗ C ▷▷ [SS'(t/x)] ⊗ [TT'] ⊗ C με FV(t)=∅

(2) [S(p(t)/p(x))S'] ⊗ C ▷▷ [SS'(t/x)] ⊗ C με FV(t)=∅

(3) [S({|p(t)|}$_y$/{|p(x)|}$_y$)S'] ⊗ C▷▷ [SS'(t/x)] ⊗ C με FV(t)=∅ και y δεσμευμένη

(4) [S(vx)S'] ⊗ C▷▷ [SS'(m/x)] ⊗ C με $x \notin FV(S)$ και $m \notin FV(C) \cup FV(S) \cup FV(S')$

Στο πρώτο reaction έχουμε μια *εξωτερική δράση* επειδή εφορά την αλληλεπίδραση δύο διαφορετικών cords ενώ σε όλα τα υπόλοιπα έχουμε *εσωτερικές δράσεις* μιας και όλες οι δράσεις τους λαμβάνουν χώρα μέσα στο ίδιο cord.

Πιο συγκεκτριμένα, στο (1) έχουμε μία αλληλεπίδραση αποστολής και λήψης που παριστάνει την ταυτόχρονη αποστολή του t απο το πρώτο cord με τη λήψη του t στη μεταβλητή x από το δεύτερο cord. Στο (2) έχουμε τη δράση pattern matching όπου το cord ελέγχει τη συμβατότητα του p(t) με το πρότυπο p(x) και ανικαθιστά το t στο x. Το (3) είναι μία δράση τύπου decryption με το cord να ελέγχει τη συμβαότητα του ({|p(t)|}$_y$ με το {|p(x)|}$_y$ και αντικαθιστά το x με το t. Τέλος, το (4) δείχνει μία δράση

ταιριάσματος όπου το cord δημιουργεί μια νέα τιμή η οποία δεν εμφανίζεται πουθενα αλλού στο cord space και αντικαθιστά με αυτή το x στο δεξί cord.

Ακολουθεί η παρουσίαση ενός τρεξίματος του NSL με τη μορφή cord space και στη συνέχεια τα reaction steps.

A = [(vm) <{|A,m|}$_B$ > (u)(u/{|m,B,υ|}$_{\bar{A}}$ <{|υ|}$_B$>]
B = [(x)(x/{|Y,z|}$_{\bar{B}}$) (vn) <{|z,B,n|}$_Y$> (ω) (ω/{|n|}$_{\bar{B}}$)]

Πινακας 2.3 NSL run

A ⊗ **B** = [(vx)<{|A,x|}$_B$>(u)…] ⊗ [(x)(x/{|Y,z|}$_{\bar{B}}$)…] **(a)**

▷▷ [<{|A,m|}$_B$>(u)…] ⊗ [(x)(x/{|Y,z|}$_{\bar{B}}$)(vy)<{|z,B,y|}$_Y$>…] **(b)**

▷▷ [(u)…] ⊗ [({|A,m|}$_B$)/{|Y,z|}$_{\bar{B}}$)(vy)<{|z,B,y|}$_Y$>…] **(c)**

▷▷ [(u)(u/{|m,B,υ|}$_{\bar{A}}$ ...]⊗ [(vy)<{|m,B,y|}$_A$>(ω)…] **(d)**

▷▷ [(u)(u/{|m,B,υ|}$_{\bar{A}}$ ...]⊗ [<{|m,B,n|}$_A$>(ω)…] **(e)**

▷▷ [({|m,B,n|}$_A$/{|m,B,υ|}$_{\bar{A}}$ <{|υ|}$_B$>]⊗ [(ω)…] **(f)**

▷▷ [<{|n|}$_B$>]⊗ [(ω)(ω/{|n|}$_{\bar{B}}$)] **(g)**

▷▷ []⊗ [({|n|}$_B$)/{|n|}$_{\bar{B}}$)] **(h)**

▷▷ [] **(i)**

Στα βήματα (a) και (d) χρησιμοποιήθηκε το reaction step (4), στα (b),(e) και (g) το (1), και στα (c), (f) και (h) το (3) [10,11].

2.4.Cord Category

Μια *διαδικασία (process) s* είναι ένας cord space μαζί με μία input interface που αποτλείται από μια azακολουθία διαφορετικών μεταξύ τους μεταβλητών και μία output interface που αποτελείται από μία ακολουθία όρων και γράφεται ως

$$s = (y_0...y_{m-1})S<t_0...t_{n-1}>$$

όπου $(y_0...y_{m-1})$ είναι μία input interface, S είναι ένας cord space και $<t_0...t_{n-1}>$ μία output interface [10,11] .

Αν το S αποτελείται από ένα μόνο cord και όλες οι ελεύθερες μεταβλητές του είναι δεσμευμένες από το input interface τότε η διαδικασία λέγεται κλειστό cord. Όλες οι μεταλητές που βρίσκονται στο input interface πρέπει να είναι διαφορετικές ανά δύο, ενώ κάτι τέτοιο δεν είναι απαραίτητο να ισχύει για τις μεταβλητές του output interface.

Πολλαπλότητα (arity) είναι μία λίστα από μετβλητές όπως η $(y_0...y_{m-1})$ modulo τη μετονομασία. Στην περίπτωση του $s = (y_0...y_{m-1})S<t_0...t_{n-1}>$ η πολλαπλότητα είναι $m=\{0,1,...,m-1\}$. Έτσι, η διαδικασία s μπορεί να παρασταθεί από το μορφισμό $s : m \rightarrow n$. Μπορεί επίσης να γραφεί και ως $s : A^m \rightarrow A^n$ όπου η πολλαπλότητα των m μεταβλητών παίρνει τη μορφή εκθέτη κάποιου αφηρημένου τύπου βάσης A. [10]

Το cord category επιτρέπει την παράλληλη και τη sequential σύνθεση των cords και των cord spaces (η σύνθεση θα αναπτυχθεί σε επόμενο κεφάλαιο). Έτσι, δεδομένων των μορφισμών

$r = (x_0...x_{l-1})R<u_0...u_{m-1}> : A^l \rightarrow A^m$
και
$s =(y_{0...}y_{m-1})S<t_0...t_{n-1}> : A^m \rightarrow A^n$

η sequential σύνθεσή τους ορίζεται να είναι

$(r;s) = (x_0...x_{l-1})RS'<t_0'...t_{n-1}'> : A^l \rightarrow A^n$

όπου τα S' και t_i' είναι τα στιγμιότυπα που προέκυψαν από την αντικατάσταση των y_k από τα u_k . Κατά τη μετονομασία αυτή χρειάζεται προσοχή ώστε στο r;s να μην έχει

δεσμευτεί κάποια από τις ελεύθερες μεταβλητές των S, t_j και u_κ . Επίσης, για κάθε agent X,

$RS' = \{[UV]_X \mid [U]_X \in R$ και $[V]_X \in S'\}$

Αφού $[]=[]_X \ \forall$ X η διαδικασία $id_m = (y_{0\ldots}y_{m-1})[] <y_{0\ldots}y_{m-1}> : A^m \rightarrow A^m$ είναι η ταυτοτική.

Δεδομένων των μορφισμών

$p = (z_0\ldots z_{k-1})P<u_0\ldots u_{l-1}> : A^k \rightarrow A^l$

και

$s =(y_{0\ldots}y_{m-1})S<t_0\ldots t_{n-1}> : A^m \rightarrow A^n$

η παράλληλη σύνθεση του p με το s $p \otimes s : A^{k+m} \rightarrow A^{l+n}$ ορόζεται ως

$p \otimes s = (\vec{z}\ \vec{y})P \otimes S<\vec{v}\ \vec{t}>$

με κατάλληλη μετονομασία (renaming) των δεσμευμένων μεταβλητών των p και s ώστε στην πράθεση (concatenation) $\vec{z}\ \vec{y}$ οι μεταβλητές του input interface να είναι διαφορετικές ανά δύο.

2.5.Πρωτόκολλα

Ρόλος είναι ένα strand που έχει input και output interfaces που χρησιμοποιούνται στη sequential composition. Όλες οι μεταβλητές που βρίσκονται σ'ένα ρόλο πρέπει να είναι δεσμευμένες είτε από το input interface είτε από δράσεις που βρίσκονται σε αυτόν. Οι ρόλοι είναι cords στα οποία αναγράφεται κάτω και δεξιά των brankets ο principal που τα εκτελεί και φυσικά πρέπει να υπάρχει συμφωνία του principal με το ιδιωτικό κλειδί. [2,10]

Πρωτόκολλο Q είναι ένα σύνολο από ρόλους $\{\rho_1,\rho_2,\ldots,\rho_\kappa\}$ που ο καθένας έχει εκτελεστεί από 0 ή περισσότερους honest principals σε οποιαδήποτε εκτέλεση του Q. Μερικοί ρόλοι που είναι διαισθητικά κατανοητοί είναι ο initiator, ο responder, ο server κτλ.

Ένας συμμετέχων σε κάποιο πρωτόκολλο λέγεται *principal* και συμβολίζεται με $\hat{A},\hat{B},...$ Το στιγμιότυπο ενός συγκεκριμένου ρόλου που εκτελείται από κάποιον principal λέγεται *thread*. Όλα τα threads κάποιου principal μοιράζονται τα ίδια στατικά δεδομένα. Για να αναφερθούμε σε έναν principal θα χρησιμοποιούμε το συμβολισμό $\hat{X}$ ενώ για να αναφερθούμε σε ένα thread το X.

Το *ιδιωτικό κλειδί* του X συμβολίζεται με $\overline{X}$ και είναι το κλειδί αποκρυπτογράφησης ενός κρυπτοσυστήματος δημσίου κλειδιού. Βρίσκεται μόνο στα διάφορα threads του ίδιου principal και πιο συγκεκριμένα στα decryption patterns και στην κατασκευή υπογραφών, με αποτέλεσμα να μην είναι επιτρεπτή η αποστολή του σε μηνύματα και να μπορούμε να δεχθούμε τη μυστικότητα των ιδιωτικών κλειδιών ως αξίωμα.

Επίθεση (attack) είναι μία διαδικασία η οποία έχει προκύψει από τη σύνθεση ενός πρωτοκόλλου με μία άλλη διαδικασία με τέτοιο τρόπο ώστε τα runs που προκύπτουν προβεβλημένα στους ρόλους του πρωτοκόλλου να μην ικανοποιούν τις προδιαγραφές του πρωτοκόλλου.[2,10] Ένας *εισβολέας (attacker/intruder)* είναι ένα σύνολο από threads που μοιράζονται όλα τα δεδομένα σε μία επίθεση και παίζουν ρόλους σε μία ή περισσότερες sessions του πρωτοκόλλου. Έστω ότι C είναι ένα cord και Dkeys(C) είναι το σύνολο όλων των ονομάτων των οποίων τα ιδιωτικά κλειδιά εμφανίζονται στο C, δηλαδή

Dkeys = { X | $\overline{X}$ εμφανίζεται στο C }

Έτσι λοιπόν, ορίζουμε ως *intruder role* με κλειδιά $K_1,\ldots,K_n$ ένα πολυσύνολο από cords I τέτοιο ώστε το Dkeys(I) να περιέχει μόνο τα $K_1,\ldots,K_n$ και η χρήση του ιδιωτικου κλειδιού να γίνεται αποκλειστικά και μόνο στις θέσεις της δράσης decryption pattern match. Με αυτό τον τρόπο ο intruder αποτρέπεται από το να χρησιμοποιεί κλειδιά που δεν έχει καθώς και να κάνει υπολογισμούς ιδιωτικού από δημόσιο κλειδί.

Ο *αρχικός σχηματισμός (initial configuration)* ενός πρωτοκόλλου Q καθορίζεται από i)ένα σύνολο από principals μερκοί από τους οποίους θεωρούνται honest, ii)ένα cord space που έχει κατασκευαστεί από τους ρόλους που εκτελούν οι honest principals (ο

κάθε honest principal μπορεί να έχει και πάνω από έναν ρόλους) και iii)έναν intruder cord που μπορεί να χρησιμοποιεί ιδιωτικά κλειδιά μόνο από dishonest principals. [2,10]

Ένα παράδειγμα initial configuration για το NSL είναι η παρακάτω :

$Init(A\ B) \otimes\ Init(C\ A) \otimes\ Resp(A) \otimes Resp(B) \otimes\ =$

$[(vx)<\{|A,x|\}_B>\ldots]_A \otimes [(vx)<\{|C,x|\}_A>\ldots]_C \otimes [(x)(x/\{|Y,z|\}_{\bar{A}})\ldots]_A\ \otimes\ [(x)(x/\{|Y,z|\}_{\bar{B}})\ldots]_B$

Μπορούμε να χρησιμοποιήσουμε εναλλακτικά το cord σ=()[]<A B C A A B> και να γράψουμε την παραπάνω initial configuration ως τη σύνθεση :

σ ; (Init ⊗ Init ⊗ Resp ⊗ Resp)

όπου το σ συνέδεσε μέσω του output interface του όλα τα threads του πρωτοκόλλου.

Σε κάθε initial configuration έχουμε έναν intruder role, ο οποίος μπορεί να επλεγεί από το σύνολο όλων των δυνατών intruder roles. Εάν $P_C=\{U_1,\ldots,U_l\}$ είναι ένα σύνολο από principals και $HONEST(C) =\{U_1,\ldots,U_k\} \subseteq P_C$ είναι ένα σύνολο από honest principals, θέτουμε I_C να είναι οποιοσδήποτε intruder role του οποίου το ιδιωτικό κλειδί να ανληκει στο σύνολο $\{U_{k+1},\ldots,U_l\}$. Για να έχουμε μία initial configuration με n ρόλους Init, m ρόλους Resp και ρόλο intruder I_C, ορίζουμε το

$\sigma = (\ )\ [\ ] <X_1\ Z_1\ X_2\ Z_2 \ldots X_n\ Z_n\ Y_1\ Y_2 \ldots Y_m\ U_1\ U_2 \ldots U_f>$

Έτσι προκύπτει η initial configuration για το NSL :

C = $Init(X_1\ Z_1) \otimes\ Init(X_2\ Z_2) \otimes \ldots \otimes Init(X_n\ Z_n) \otimes$

$Resp(Y_1) \otimes\ Resp(Y_2) \otimes \ldots \otimes Resp(Y_m) \otimes$

$I_C(U_1\ U_2 \ldots U_l)$

Όπου $X_1,X_2,\ldots,X_n,Y_1,Y_2,\ldots,Y_n \in HONEST(C)$ και $U_1,U_2,\ldots,U_l \in P_C \backslash HONEST(C)$.

Τρέξιμο (run) ενός πρωτοκόλλου είναι μία ακολουθία από reaction steps μίας initial configuration.

Γεγονός (event) είναι ένα στιγμιότυπο αντικατάστασης μίας δράσης, δηλαδή μία δράση στην οποία όλες οι μεταβλητές έχουν αντικατασταθεί από όρους που περιέχουν μόνο

σταθερές. Τα events αναπαριστούν τα αποτελέσματα των reaction steps που επενεργουν σε ένα cord.[10]

Τα events συμβολίζονται ως :

EVENT (R, X, P, $\vec{n}$, $\vec{x}$) ≡ (([SPS']$_X$ ⊗ C ▷▷ [SS'($\vec{n}$ / $\vec{x}$)]$_X$ ⊗ C') ∈ R)

Που σημαίνει ότι στη run R ο principal X εκτελεί τις δράσεις P δεχόμενος δεδομένα $\vec{n}$ στις μεταβλητές $\vec{x}$, όπου $\vec{n}$ και $\vec{x}$ έχουν το ίδιο μήκος.

Εάν τα $\vec{n}$, $\vec{x}$ είναι μη κενά διανύσματα τότε το κατηγόρμα είναι αληθές μόνο αν το P είναι δράση τύπου receive, new ή pattern match. Παραδείγματα από κατηγορήματα που είναι true είναι τα :

EVENT(R, A, <{|A,m|}$_E$>, ∅,∅)

EVENT(R, A, <{|n|}$_E$>, ∅,∅)

EVENT(R, B, ({|A,m|}$_B$/{|Y,z|}$_{\vec{B}}$) , {|A,m|} , {|Y,z|})

EVENT(R, A, (vx), m, x)

<u>Ίχνος (trace)</u> είναι μία λίστα από events κάποιου thread μέσα σε ένα run. Συμβολίζουμε με R|$_X$ τα events που διαδραματίστηκαν από το thread X στη run R. Λέμε ότι *<u>το P ταιριάζει στο</u>* R|$_X$ (όπου P ακολουθία από actions του thread X στη run R του πρωτοκόλλου Q και R|$_X$ είναι τα events που συμβαίνουν στο run R από τον principal X) εάν το R|$_X$ καταλήγει ακριβώς στο σP –όπου σ είναι μία αντικατάσταση τιμών στις μεταβλητές- και τότε το σ ονομάζεται *<u>υποκατάσταση ταιριάσματος (matching substitution)</u>* [10]

<u>Λήμμα:</u> Για κάθε configuration του πρωτοκόλλου Q και για κάθε run R, εάν ο principal X ∈ HONEST(C) το R|$_X$ είναι ένα interleaving των traces των ρόλων του Q που έχουν εκτελεσθεί από τον X. [2,10,11]

Απόδειξη: Η απόδειξη είναι άμεση από τον ορισμό του initial configuration που έχει κατασκευασθεί αναθέτοντας έναν ή περισσότερους ρόλους του Q σε κάθε honest principal. □

2.6.Ιδιότητες των πρωτοκόλλων

Λήμμα 2.1: (της μη τηλεπάθειας- no telephathy-) Έστω ότι Q είναι ένα πρωτόκολλο, R μία τυχαία run και X ένα thread και $R|_X$ αποτελείται από αρχικά τμήματα των traces $\rho_1,\rho_2,...,\rho_k$, όπου κάθε ρ_i είναι ένας ρόλος του Q. Έστω ότι m είναι ένα μήνυμα που έχει σταλεί από τον X ως τμήμα του ρόλου ρ_i . Τότε κάθε σύμβολο στον όρο m είτε έχει δημιουργηθεί στο ρ_i , είτε έχει ληφθεί στο ρ_i είτε βρισκόταν στο στατικό interface του ρ_i . [2,10,11]

Απόδειξη: Προκύπτει από τον ορισμό των cords που χρησιμοποιούμε για να αναπαραστήσουμε τους ρόλους. Κάθε ρόλος είναι ένα κλειστό cord και έτσι όλες οι τιμές πρέπει να είναι δεσμευμένες. ΄Ολα τα σύμβολα δεσμεύονται είτε στο στατικό interface, είτε με τις δράσεις ν, receive και pattern match. □

Τα δίκτυα λειτουργούν με ασύγχρονο τρόπο, δηλαδή παρέχουν έναν προσωρινό αποθηκευτικό χώρο (buffer) όπου τα μηνύματα μπορούν να αποθηκευτούν ώσπου ο παραλήπτης τους να είναι σε θέση να τα λάβει. Προκειμένου να μοντελοποιηθεί αυτό στον cord calculus εισάγουμε την έννοια του *buffer cord* [(x)<x>] ,ενός cord που λαμβάνει και στη συνέχεια στέλνει τα μηνύματα. Θεωρούμε ότι όλοι οι principals και ο intruder στέλνουν και λαμβάνουν τα μηνύματά τους μέσω του buffer cord και ότι σε κάθε πρωτόκολλο υπάρχουν ακετά στιγμιότυπα του buffer cord ώστε να εξασφαλίζεται η διανόμή όλων των μηνυμάτων. Υποθέοτυμε ότι τα buffer cords εκτελούνται από ανώνυμους agents και ουσιαστικά δεν αποτελούν μέρος του πρωτοκόλλου.

Λήμμα 2.2: (της ασύγχρονης επικοινωνίας) Σε κάθε run, κάθε thread που θέλει να στείλει ένα μήνυμα μπορεί πάντοτε να το στείλει. Επισης, υπάρχει μία αυστηρή γραμμική διάταξη σε όλες τις εξωτερικές πράξεις. [2,10,11]

Απόδειξη : Εξ ορισμού υπάρχουν αρκετά buffer cords στην initial configuration ώστε να προσφέρουν μία λήψη για κάθε δράση αποστολής από ένα non-buffer cord. Εφόσον η εξωτερική πράξη αναφέρεται σε μία λήψη ή αποστολή από ένα non-buffer thread συνεπάγεται από τον ορισμό του run ότι δε μπορούν να συμβούν δύο εξωτερικές πράξεις στο ίδιο βήμα του run. □

Λήμμα 2.3:Για κάθε δράση τύπου receive υπάρχει μία αντίστοιχη δράση τύπου send. Πιο τυπικά, εάν στη run R ο thread X εκτέλεσε μία δράση τύπου receive δέχομενος δεδομένα m στη μεταβλητή x , τότε υπάρχει ένας thread Y τέτοιος ώστε στην ίδια run R εκτέλεσε τη δράση <m>. [10]

Απόδειξη :Προκύπτει άμεσα από τον ορισμό των reaction steps και των buffer cords. □

Λήμμα 2.4: Για κάθε initial configuration C ενός πρωτοκόλλου Q και κάθε run R, αν ο principal $\hat{X} \in$ HONEST(C) τότε για κάθε thread X που έχει εκτελεστεί από τον principal $\hat{X}$, το $R|_X$ είναι ένα ίχνος ενός μοναδικού ρόλου του Q που έχει εκτελεστεί απο τον $\hat{X}$. [10]

Απόδειξη : Προκύπτει άμεσα από τον ορισμό της initial configuration που έχει κατασκευασθεί αναθέτοντας ρόλους στα threads των honest principals. □

Στην κατασκευή των μοντέλων έχει γίνει η σύμβαση να μην είναι εφικτή η αποκρυπτογράφηση ενός μηνυματος το οποίο έχει κρυπτογραφηθεί με δημόσιο κλειδί ενός honest principal από οποιονδήποτε άλλο εκτός από τον συγκεκριμένο principal. Αυτό συμβαίνει επειδή, αφενός μεν, δεν επιτρέπουμε στον intruder cord να περιέχει δράσεις αποκρυπτογραφησης που χρησιμοποιούν κλειδιά από honest principals (και υποθέτουμε ότι δε μπορεί να τα μαντέψει), αφεταίρου δε, κάνουμε reasoning μόνο για πρωτόκολλα όπου τα ιδιωτικά κλειδιά εμφανίζονται στη δράση τύπου decrypt pattern

match, δηλαδή σε πρωτόκολλα των οποίων οι ρόλοι δεν επιτρέπεται να στείλουν ιδωτικά κλειδιά σε μηνύματα.

Λήμμα 2.5: (secrecy) Για κάθε initial configuration C του πρωτοκόλλου Q και για κάθε run R

$$\text{EVENT}\ (\ R, X, (\{|t|\}_Y / \{|x|\}_{\overline{Y}}\ , t, x\ ) \wedge Y \in \text{HONEST}(C) \supset X = Y$$

Απόδειξη : Έστω ότι ρ είναι ο ρόλος στον οποίο εκτελέστηκε το event της run R. Έστω ότι σ_c είναι η υποκατάσταση που χρησιμοποιήθηκε στην initial configuration C. Διακρίνουμε δύο περιπτώσεις :

1η περίπτωση : Υποθέτουμε ότι $Y \in FV(\rho)$. Εάν το ρ είναι ο ρόλος του intruder τότε το ρ δεν μπορεί να περιέχει αυτή τη δράση επειδή $DKeys(\sigma_c ; \rho) \cap HONEST(C) = \varnothing$ Εάν το ρ είναι ένας ρόλος πρωτοκόλλου τότε $DKeys(\sigma_c ; \rho) = \{X\}$ λόγω του περιορισμού για τα decryption keys των καλώς ορισμένων ρόλων των πρωτοκόλλων . Άρα το Y δεν μπορεί να ανήκει στο $DKeys(\sigma_c ; \rho)$ παρά μόνο αν Y = X.

2η περίπτωση : Υποθέτουμε ότι $Y \notin FV(\rho)$. Τότε, θα πρέπει με κάποιο τρόπο το $\overline{Y}$ να έχει ληφθεί από το ρ μέσα σε κάποιο μήνυμα. Όμως κάτι τέτοιο αντίκειται στον περιορισμό ο οποίος δεν επιτρέπει στα ιδωτικά κλειδιά των καλώς ορισμένων ρόλων των πρωτοκόλλων να εμφανίζονται παρά μόνο στη δράση τύπου decrypt pattern match. [10]

ΚΕΦΑΛΑΙΟ 3

Protocol Logic

3.1.Εισαγωγή

Στο προηγούμενο κεφάλαιο αναλύσαμε τον Cord Calculus, δηλαδή τον process calculus γύρω από τον οποίο είναι δομημένη η PCL. Είδαμε το συντακτικό του καθώς και τον τρόπο με τον οποίο τα reaction steps περιγράφουν την εκτέλεση των πρωτοκόλλων. Παρακάτω θα αναπτυχθεί το συντακτικό της protocol logic, η σημασιολογία της και οι αποδεικτικές μέθοδοι που μπορούν να εφαρμοστούν στα security protocols.

3.2.Συντακτικό της Protocol Logic

Όπως έχουμε δει μέχρι τώρα ένα thread είναι μία ακολουθία από actions όπου ένας principal εκτελεί ένα στιγμιότυπο κάποιου ρόλου και με $\hat{X}$ συμβολίζουμε τον principal που εκτελεί το thread X. Τα μηνύματα είναι δομημένα έτσι ώστε να περιλαμβάνουν την πηγή (source), τον προορισμό (destination), τον «προσδιοριστή» πρωτοκόλλου (protocol identifier) και φυσικά τα περιεχόμενα (contents). Το πεδίο της πηγής μπορεί να μην καθορίζει ακριβώς τον αποστολέα μιας και ένας intruder μπορεί να αλλοιώσει το περιεχόμενό του. Το ίδιο ισχύει και για τον προοροσμό αφού μπορεί να μην

καταλήξει το μήνυμα στο σωστό προορισμό λόγω πιθανής παρεμπόδισης από κάποιον intruder. Ωστόσο, τα δύο τελευταία πεδία είναι χρήσιμα για απόδειξη ιδιοτήτων authentication.

Στον πίνακα που ακολουθεί δίνεται το συντακτικό της λογικής :

<u>Φόρμουλες δράσης (action formulas)</u>
a ::= Send(P,m) | Receive(P,m) | New(P,t) | Encrypt(P,t) | Decrypt(P,t) | Sign(P,t) | Verify(P,t)
<u>Φόρμουλες</u>
ϕ ::= a | $a_1<a_2$ | Has(P,t) | Fresh(P,t) |Gen(P,t) | FirstSend(P,t,t') | Honest(N) | $t_1=t_2$ | Contains(t_1,t_2) | $\varphi \wedge \varphi$ | $\neg\varphi$ | $\exists x.\varphi$ | <->ϕ | (-)ϕ | Start(P) | Source(n,X,t,K) | CSend(X, $\{\|m\|\}_K$) | HasAlone(X,t)
<u>Τροπικές Φόρμουλες (Modal formulas)</u>
ϕSϕ

Πίνακας 3.1.Συντακτικό της protocol logic

Στον παραπάνω πίνακα έχουμε συμβολίσει με t τους όρους, P τα threads, S κάποιο Strand, ϕ και ψ τις κατηγορηματικές φόρμουλες (predicate formulas) και m τα μηνύματα.

Οι <u>*action formulas*</u> [2,4,6,10,11,13] αναφέρονται στις δράσεις που εκτελούνται από τα ατνίστοιχα threads. Η Send(X,m) σημαίνει ότι ο principal $\hat{X}$ στέλνει το μήνυμα m στο thread X. Τα Receive, New, Encrypt, Decrypt, Sign και Verify έχουν τις αντίστοιχες σημασίες των δράσεων των οποίων φέρουν το όνομα. [2,4,10,11]

Η φόρμουλα $a_1<a_2$ σημαίνει ότι οι δράσεις a_1 και a_2 έχουν συμβεί στο run και ότι, επιπλέον, η δράση a_2 έγινε μετά την a_1. Μπορεί όμως αυτές οι δράσεις να μην είναι μοναδικές. Σε αυτή την περίπτωση αυτός ο τελεστής χρονικής διάταξης μας

πληροφορεί απλώς ότι δύο οποιεσδήποτε δράσεις a_1 και a_2 έχουν συμβεί με αυτή τη σειρά.

Η φόρμουλα Has(X,x) σημαίνει ότι ο principal $\hat{X}$ κατέχει την πληροφορία x στο thread X. Το «κατέχειν» εδώ έχει την έννοια ότι είτε ο $\hat{X}$ δημιούργησε το x είτε το δέχθηκε κρυπτογραφημένο (με την προυπόθεση να του είναι γνωστό το κλειδί) ή μη. Χρησιμοποιείται κυρίως για τις ιδιότητες secrecy, όπως για παράδιεγμα αν υποθέσουμε ότι ο όρος t αποτελεί μυστικό μεταξύ των threads X και Y, τότε μπορούμε να το περιγράψουμε με το $\forall Z.Has(Z,t) \supset (Z = X \vee Z = Y)$.

Η φόρμουλα CSend(X, $\{|m|\}_K$) , η οποία αποτελεί συντομογραφία του “Created and Sent”, σημαίνει ότι ο Thread X ξέρει το m (δεν είναι απαραίτητο να το έχει δημιουργήσει ο X) , ξέρει το κλειδί K και έχει στείλει το μήνυμα $\{|m|\}_K$.

Το HasAlone(X,t) σημαίνει ότι ο μοδικός thread που έχει τον όρο t είναι ο X και ορίζεται ως HasAlone(X,t) $\equiv$ Has(X,t) $\wedge$ Has(Y,t) $\supset$ X=Y.

Η Fresh(X,t) σημαίνει ότι ο όρος t που δημιουργήθηκε στο X είναι «φρέσκος», δηλαδή μέχρι τώρα δεν τον έχει δει κανείς ξανά ούτε ως υποόρο (συνήθως χρησιμοποιείται για nonces). Η φόρμουλα Gen(X,t) σημαίνει οτι ο όρος t προέρχεται από το thread X και κάποτε ήταν «φρέσκος» στο X. Η FirstSend(X,t,t’) σημαίνει ότι ο thread X έχει στείλει τον όρο t (ενδεχομένως και ως υποόρο) και ότι το πρώτο τέτοιο συμβάν ήταν μία δράση όπου ο X έστειλε το μήνυμα t’.

Η *φόρμουλα* Honest(N) σημαίνει ότι οι δράσεις του principal $\hat{X}$ στην τρέχουσα run είναι ακριβώς το interleaving των αρχικών τμημάτων των traces ενός συνόλου ρόλων του πρωτοκόλλου. Δηλαδή, κάθε thread X εκτελεί ακριβώς εκείνες τις δράσεις που προβλέπονται από το ρόλο που έχει στο πρωτόκολλο.

Η φόρμουλα Contains(t_1,t_2) σημαίνει ότι ο όρος t_1 περιέχει τον όρο t_2 σαν υποόρο. Η φόρμουλα Start(X) σημαίνει ότι ο thread X δεν έχει εκτελέσει άλλες δράσεις στο παρελθόν.

Η <-> σημαίνει ότι σε κάποια κατάσταση στο παρελθόν ισχύει η φ ενώ η (-) σημαίνει ότι η φ ισχύει στην προηγούμενη κατάσταση.

Τέλος, η φόρμουλα Source(n,X,t,K) σημαίνει ότι ο μόνος τρόπος για έναν principal $\hat{Y}$ να γνωρίζει το n είναι να το έχει μάθει από ένα μήνυμα $\{|t|\}_K$ (ενδεχομένως και από κάποια έμμεση οδό). Το Source(n,X,t,K) μπορεί να είναι true εάν το n έχει δημιουργηθεί από μία δράση τύπου (vn) και αν δεν έχει σταλεί ποτέ σε μη κρυπτογραφημένη μορφή από τον X. Χρησιμοποιείται λοιπόν για το reasoning της πηγής μιας πληροφορίας, συνήθως για nonces.

Πρέπει να σημειώσουμε ότι οι φόρμουλες Start, $a_1<a_2$ και FirstSend κάνουν χρονική διάταξη.

Οι *modal formulas* εφαρμόζουν preconditions και postconditions στα προγράμματα. Μία φόρμουλα της μορφής $\theta[P]_X\phi$ σημαίνει ότι αφού επενεργήσουν οι δράσεις P στο thread X, ξεκινώντας από μία κατάσταση στην οποία η φόρμουλα θ είναι true, η φόρμουλα φ είναι true για την κατάσταση στην οποία έχει επέλθει το X. [2]

3.3.Σημασιολογία (Semantics)

Μία φόρμουλα μπορεί να είναι αληθής ή ψευδής (true ή false) σε μια run του πρωτοκόλλου. Συγκεκριμένα, λέμε ότι «η φόρμουλα φ ισχύει για τη run R του πρωτοκόλλου Q» αν Q,R |= φ . Εδώ το R μπορεί είτε να έχει ολοκληρωθεί είτε να εκκρεμεί η έλευση κάποιων μηνυμάτων σε threads. Με $\overline{Q}$ συμβολίζουμε το σύνολο όλων των initial configurations του πρωτοκόλλου Q, καθεμία από τις οποίες περιλαμβάνει ένα πιθανό intruder cord. Runs(Q) είναι το σύνολο όλων των runs του πρωτοκόλλου Q που περιέχουν ρόλο intruder και κάθε run ξεκινάει από μία initial configuration που ανήκει στο σύνολο $\overline{Q}$. Εάν το φ περιέχει ελεύθερες μεταβλητές, τότε (ισχύει το) Q,R |= φ εάν Q,R |= σφ για κάθε υποκατάσταση σ που δεσμεύει όλες τις ελεύθερες μεταβλητές του φ. Τέλος, γράφουμε Q |= φ αν Q,R |= φ $\forall R \in Runs(Q)$.

Επειδή το run ενός πρωτοκόλλου είναι η ακολουθία reaction steps είναι εφικτό να αποφανθούμε για το αν μια συγκεκριμένη δράση έλαβε χώρα στο run καθώς επίσης και για τη χρονική διάταξη των δράσεων. Αν δούμε τα runs ως γραμμικές ακολουθίες καταστάσεων και λαμβάνοντας υπόψιν ότι η μετάβαση από τη μία κατάσταση στην επόμενη επηρρεάζεται από μία δράση κάποιου thread σε ένα ρόλο και συσχετίζοντας τη δράση με την κατάσταση στην οποία καταλήγει ένα σύστημα, συμπεράινουμε ότι μία action formula a σε ένα run είναι true εάν είναι η τελευταία action αυτού του run (αφού μία φόρμουλα είναι true σε κάποιο run εάν είναι true στην τελευταία κτάσταση αυτού του run). Μία past formula <->a είναι true εάν στο παρελθόν η action formula a ήταν true σε κάποια κατάσταση.

Το κατηγόρημα μιας σγκεκριμένης δράσης ισχύει στη run εάν έχει συμβεί η αντίστοιχη δράση (στην ίδια run) με τους ίδιους όρους για παραμέτρους. Πιο συγκεκριμένα : [2,4,10,11]

Q,R |= Send(A,m) εάν στη run R ο thread A εκτέλεσε τη δράση <m>, δηλαδή αν EVENT(R,A,<m>,∅,∅).

Q,R |= Rceive(A,m) εάν υπάρχει μεταβλητή x τέτοια ώστε στο run R ο thread A να έχει εκτελέσει τη δράση (x) δεχόμενος δεδομένα m στη μεταβλητή x, δηλαδή αν EVENT(R,A,(x),m,x).

Q,R |= New(A,m) εάν υπάρχει μία μεταβλητή x τέτοια ώστε στο run R ο thread A να έχει εκτελέσει τη δράση (νx) βάζοντας δεδομένα m στη μεταβλητή x, δηλαδή αν EVENT(R,A,(νx),m,x).

Q,R |= Encrypt(A, $\{|m|\}_K$) εάν υπαρχει μεταβλητή x τέτοια ώστε στο run R ο thread A έχει εκτελέσει τη δράση x:=$\{|m|\}_K$ βάζοντας τα δεδομένα $\{|m|\}_K$ στη μεταβλητή x, δηλαδή αν EVENT(R,A, x:=$\{|m|\}_K$, $\{|m|\}_K$, x).

Q,R |= Decrypt(A,$\{|m|\}_K$) εάν υπάρχει μία μεταβλητή x τέτοια ώστε στη run R ο thread A να έχει εκτελέσει τη δράση $\{|m|\}_K/\{|x|\}_{\overline{K}}$ δηλαδή αν EVENT(R,A, $\{|m|\}_K/\{|x|\}_{\overline{K}}$, m, x).

Q,R |= Sign(A,$\{|m|\}_{\overline{K}}$) εάν υπα΄ρχει μεταβλητή x τέτοια ώστε στο run R ο thread A να έχει εκτελέσει τη δράση x:=$\{|m|\}_{\overline{K}}$ βάζοντας τα δεδομένα $\{|m|\}_{\overline{K}}$ στη μεταβλητή x, δηλαδή αν EVENT(R,A, x:=$\{|m|\}_{\overline{K}}$, $\{|m|\}_{\overline{K}}$, x).

Q,R |= Verify(A, $\{|m|\}_{\overline{K}}$) εάν υπάρχει μεταβλητή x τέτοια ώστε στο run R ο thread A να έχει εκτελέσει τη δράση signature verification $\{|m|\}_{\overline{K}}/\{|m|\}_K$, δηλαδή αν EVENT(R,A, $\{|m|\}_{\overline{K}}/\{|m|\}_K$, $\varnothing,\varnothing$).

Το κατηγόρημα Has χρησιμοποιείται για να μοντελοποιήσει κατά κάποιο τρόπο αυτό που διαισθητικά αντιλαμβανόμαστε ως γνώση. Το Has ισχύει για όρους οι οποίοι είναι γνωστοί είτε άμεσα (με τη μορφή ελεύθερης μεταβλητής ή ώς αποτέλεσμα των δράσεων λήψης/δημιουργίας) είτε έμμεσα (μέσω των δράσεων pattern matching/decryption/encryption/tupling).

Επομένως :

Q,R |= Has(A,m) εάν υπάρχει i τέτοιο ώστε να ισχύει το $Has_i(A,m)$, όπου το Has_i ορίζεται επαγωγικά ως εξής :

$Has_0(A,m)$ εάν $(m \in FV(R|_A))$

$\vee$ EVENT(R, A, (vx), m, x)

$\vee$ EVENT(R, A, (x), m, x)

και $Has_{i+1}(A,m)$ εάν $Has_i(A,m)$

$\vee$ ($Has_i(A,m')$

$\vee$ ($Has_i(A,m') \wedge Has_i(A,m'') \wedge ((m=m',m'') \vee (m=m'',m'))$)

$\vee$ ($Has_i(A,m') \wedge Has_i(A,K) \wedge m=\{|m'|\}_K$)

$\vee$ ($Has_i(A,\alpha) \wedge Has_i(A, g^b) \wedge m=g^{\alpha b}$)

$\vee$ ($Has_i(A, g^{\alpha b}) \wedge m=g^{b\alpha}$)

Q,R |= Fresh(A,t) εάν η Q,R |= New(A,t) ισχύει και επιπλέον για κάθε m τέτοιο ώστε Q,R |= Send(A,m) να ισχύει ότι το t δεν είναι υποόρος του m.

Q,R |= Start(A) εάν $R|_A$ είναι κενό, δηλαδή ότι ο A δεν έχει εκτελέσει καθόλου πραξεις στο παρελθόν.

Q,R |= Gen(A,t) εάν υπάρχει πρόθεμα R' του R τέτοιο ώστε να ισχύει η Q,R' |= Fresh(A,t).

Q,R |= FirstSend(A,t,t') εάν το t είναι υποόρος του t', ισχύει η Q,R |= Send(A,t') και για κάθε πρόθεμα R' του R και για κάθε όρο t'' τέτοιο ώστε $t \sqsubseteq t''$ και Q,R' |= Send(A,t'') πρέπει να ισχύει η Q,R' |= Send(A,t').

Q,R |= HONEST($\hat{A}$) εάν $\hat{A} \in$ HONEST(C) στην initial configuration C για το R και όλα τα threads του $\hat{A}$ είναι σε κατάσταση pausing στο R. Συγκεκριμένα, το $R|_{\hat{A}}$ είναι ένα interleaving των βασικών ακολουθιών των ρόλων στο Q.

Q,R |= Contains(t_1,t_2) εάν $t_1 \sqsubseteq t_2$, όπου η $\sqsubseteq$ είναι η σχέση υποόρων μεταξύ των όρων.

Q,R |= <->ϕ εάν Q,R' |= ϕ όπου R' είναι ένα πρόθεμα (όχι απαραιτήτως γνήσιο) του R. Αυτό σημαίνει ότι η φόρμουλα ϕ είναι true σε κάποια κατάσταση στο παρελθόν.

Q,R |= (-)ϕ εάν Q,R' |= ϕ , όπου R=r'e για κάποιο event e, δηλαδή η (-)ϕ είναι true σε μία κατάσταση εάν είναι true στην προηγούμενη κατάσταση.

Q,R |= ($\phi_1 \wedge \phi_2$) εάν Q,R |= ϕ_1 και Q,R |= ϕ_2.

Q,R |= $\neg\phi$ εάν Q,R |$\neq \phi$.

Q,R |= $\exists x \varphi$ εάν Q,R |= (d/x)ϕ για κάποιο d, όπου (d/x)ϕ συμβολίζει τη φόρμουλα που προέκυψε από την αντικατάσταση του d στο x της ϕ.

Q,R |= Source(m,A,t,K) εάν EVENT(R,A,(vx),m,x) $\wedge$ Q,R |= $\forall Z.((Z \neq A \wedge Has(Z,m))$ $\supset$(Decrypts(Z,{|t|}$_K$) $\vee$ ($\exists$Y.Decrypts(Y,{|t|}$_K$) $\wedge$ Send(Y,t’))) όπου m $\subseteq$ t’.

Q,R |= $\phi_1[P]\phi_2$ εάν R=$R_0R_1R_2$ για κάποια R_0,R_1 και R_2 και είτε το P δεν ταιριάζει (matches) το $R_1|_A$ είτε το P ταιριάζει το $R_1|_A$ και Q,R_0 |= σϕ_1 συνεπάγεται Q,R_0R_1 |= σϕ_2 όπου το σ είναι η υποκατάσταση που ταιριάζει το P στο $R_1|_A$. [2, 10]

3.4.Αποδεικτικό Σύστημα

Υπάρχουν αξιώματα για τις δράσεις που διενεργούνται στα πρωτόκολλα και που ισχύουν ως τα αποτελέσματα της εκτέλεσης ή μη των δράσεων αυτών. Παρακάτω παρατίθενται μερικά από τα αξιώματα αυτά. Χρησιμοποιείται το α για το συμβολισμό των δράσεων, ενώ το a για τα αντίστοιχα κατηγορήματα (predicates). Το T συμβολίζει τη λογική τιμή true (boolean value).

<u>Αξιώματα που αφορούν δράσεις πρωτοκόλλων :</u> [2,4,10,11]

AA1 : $\phi[\alpha]_X$ <->a

Έστω ένα πρωτόκολλο Q και R=$R_0R_1R_2$ ένα run τέτοιο ώστε η $R_1|_X$ να ταιριάζει (matches) το α υπό την υποκατάσταση σ και Q,R_0 |= <->σϕ. Αρκεί να δείξουμε ότι Q,R_0R_1 |= <-> σa. Αφού το $R_1|_X$ ταιριάζει το α υπό την υποκατάσταση σ, το R_1 πρέπει να περιέχει τη δράση σα άρα, λόγω της σημασιολογίας του χρονικού τελεστή <-> συμπεραίνουμε ότι θα πρέπει Q, R_0R_1 |= <->a. Από τον ορισμό των modal formulas έχουμε ότι Q |= $\phi[\alpha]_X$ <->a.

Με άλλα λόγια, το AA1 λέει ότι αν ένας principal έχει εκτελέσει μία δράση α σε κάποιο ρόλο τότε το αντίστοιχο κατηγόρημα (predicate) που λέει ότι αυτή η δράση έλαβε χώρα στο παρελθόν είναι true.

AA2 : Fresh(X,t) $[\alpha]_X$ <-> (α $\wedge$ (-)Fresh(X,t))

Έστω ένα πρωτόκολλο Q και $R=R_0R_1R_2$ μία run τέτοια ώστε η $R_1|_X$ να ταιριάζει τη δράση α υπό την υποκατάσταση σ και $Q,R_0 \models \sigma Fresh(X,t)$. Αρκεί να δείξουμε ότι $Q,R_0R_1 \models \sigma <->(a \wedge (-)Fresh(X,t))$. Αφού η R_1 ταιριάζει την α υπό τη σ έπεται ότι η R_1 περιέχει τη σα. Έστω R_1' ένα πρόθεμα της R_1 που περιέχει όλες τις δράσεις που προηγούνται της σα. Τότε εύκολα συνάγεται ότι $Q,R_0R_1'\sigma a \models \sigma a$. Από την άλλη, $Q,R_0 \models \sigma Fresh(X,t)$. Το R_1 δεν περιέχει καθόλου δράσεις του thread X, άρα, από τη σημασιολογία του predicate Fresh, η εγκυρότητα της φόρμουλας Fresh(X,t) εξαρτάται μόνο από τις δράσεις που εκτέλεσε ο X στο παρελθόν. Επομένως, $Q,R_0R_1 \models \sigma Fresh(X,t)$ και έτσι $Q,R_0R_1\sigma a \models (-)\sigma Fresh(X,t)$ (από τη σημασιολογία του τελεστή (-)). Τέλος, λόγω της σημασιολογίας του τελεστή <->, έπεται ότι θα πρέπει $Q,R_0R_1 \models \sigma <->(a \wedge (-)Fresh(X,t))$.

Διαισθητικά, το παραπάνω αξίωμα λέει ότι εάν ένας όρος t σε μία κατάσταση είναι fresh, παραμαίνει fresh τουλάχιστον μέχρι ο συγκεκριμένος thread να εκτελέσει κάποια δράση.

Ένας ισοδύναμος τρόπος γραφής του αξιώματος AA2 είναι ο $Start(X)[]_X \neg a(X)$ου λέει ότι κατά την έναρξη ενός thread οποιαδήποτε σημασιολογία δράσης (action predicate) που εφαρμόζεται σε αυτό το thread είναι false.

AA3 : $\neg$Send(X,t) $[b]_X \neg$Send(X,t) εάν σSend(X,t) $\neq$ σb $\forall$ υποκατάσταση σ

Το αξίωμα αυτό μας λέει ότι το κατηγόρημα που ισχυρίζεται ότι ο thread X έχει στείλει τον όρο t παραμένει false μετά την εκτέλεση οποιασδήποτε πράξης που δε στέλνει όρο που περιέχει τον t.

AA4 : $T[\alpha;\ldots;b]_X\ a < b$

Το αξίωμα αυτό μας λέει ότι αφού ο thread X εκτελέσει τις δράσεις α,...,b στη σειρά, τα κατηγορήματα (predicates) a και b, που αντιστοιχούν στις δράσεις α και b έχουν διαταχθεί χρονικά με την ίδια σειρά.

Τα παρακάτω αξιώματα έχουν να κάνουν με τις ιδιότητες των «φρέσκα» (freshly) δημιουργημένων nonces :

AN1 : $New(X,x) \wedge New(Y,x) \supset Y=X$

Το αξίωμα αυτό μας λέει ότι κάθε συγκεκριμένο nonce έχει δημιουργηθεί από έναν και μόνο thread.

AN2 : $\phi[(vn)]_X\ Has(Y,n) \supset (Y=X)$

Το αξίωμα AN2 μας λέει ότι ο thread X δημιουργήσει μία νέα τιμή m και δεν εκτελέσει άλλες δράσεις, τότε μόνο ο X μπορεί να ξέρει το m. Το AN2 αναφέρεται στη μυστικότητα (secrecy) των «φρέσκων» (fresh) nonces.

AN3 : $\phi[(vn)]_X\ Fresh(X,n)$

Το αξίωμα αυτό μας λέει ότι μία τιμή (value) που μόλις δημιουργήθηκε είναι Fresh αμέσως μετά τη δημιουργία της.

AN4 : $Fresh(X,x) \supset Gen(X,x)$

Το παραπάνω αξίωμα μας λέει ότι ο X είναι ο thread από τον οποίο προέρχεται το nonce n.

ARP : <->Receive(X, p(x))$[q(x)/q(t)]_X$ <->Receive(X, p(t))

Έστω Q ένα πρωτόκολλο και $R=R_0R_1R_2$ ένα run τέτοιο ώστε το $R_1|_X$ να ταιριάζει το (q(x)/q(t)) υπό την αντικατάσταση σ και $Q,R_0 \models \sigma$<->Receive(X, p(x)). Αρκεί να δείξουμε ότι $Q,R_0R_1 \models \sigma$<->Receive(X, p(t)). Εφόσον $R_1|_X$ ταιριάζει το (q(x)/q(t)) υπό τη σ και τα events της R_1 περιέχουν μόνο ground terms, θε πρέπει η σx να είναι ίδια με τη σt και έτσι $Q,R_0 \models$ <->Receive(X, p(t)). Προφανώς οι φόρμουλες του τύπου <->a ισχύουν αφού νέες δράσεις εκτελέστηκαν κι έτσι $Q,R_0R_1 \models \sigma$<->Receive(X, p(t)).

Αξιώματα κατοχής (possession axioms) : [2,4,10,11]

Τα αξιώματα κατοχής σχετίζονται με το πώς ένας thread μπορεί να συσσωρεύσει γνώση και να ανακτήσει την πληροφορία για κάποιον όρο εάν έχει όλα τα τμήματα που είναι απαραίτητα για την κατασκευή του.

Τα αξιώματα κατοχής είναι τα ακόλουθα :

TUP : Has(X,x) $\wedge$ Has(X,y) $\supset$ Has(X, (x,y))

ENC : Has(X,x) $\wedge$ Has(X,K) $\supset$ Has(X, $\{|x|\}_K$)

Τα δύο παραπάνω αξιώματα κατοχής δίνουν τη δυνατότητα κατασκευής tuples και κρυπτογραφημένων όρων όταν τα απαραίτητα για αυτές τις δράσεις μέρη είναι γνωστά.

PROJ : Has(X, (x,y)) $\supset$ Has(X,x) $\wedge$ Has(X,y)

DEC : Has(X, $\{|x|\}_K$) $\wedge$ Has(X,K) $\supset$ Has(X,x)

Τα αξιώματα PROJκαι DEC επιτρέπουν την αποσύνθεση tuples στα συστατικά τους μέρη και την αποκρυπτογράφηση ενός κρυπτογραφημένου όρου εάν το κλειδί είναι γνωστό.

DEC1 : Decrypts(X, $\{|m|\}_K$) $\supset$ Has(X, $\{|m|\}_K$)

DEC2 : Decrypts(X, $\{|m|\}_K$) $\supset$ Has(X,m)

Τα δύο προηγούμενα αξιώματα λένε ότι εάν ένας thread X αποκρυπτογραφήσει ένα μήνυμα $\{|m|\}_K$, αυτό συνεπάγεται ότι κατέχει τη γνώση τόσο του $\{|m|\}_K$ όσο και του m.

SRC : Source(m, X, t, K) $\wedge$ Has(Z, m) $\wedge$ $Z \neq X \supset \exists$Y.Decrypts(Y, $\{|t|\}_K$)

Το παραπάνω αξίωμα μας λέει ότι εάν ο thread X έχει δημιουργήσει μία τιμή m και αυτή έχει σταλεί μόνο σαν μέρος του μηνύματος $\{|t|\}_K$ και κάποιος άλλος thread Z έμαθε το m, θα πρέπει να υπάρχει thread Y ο οποίος να έχει αποκρυπτογραφήσει το $\{|t|\}_K$.

CSEND : CSend(X, $\{|m|\}_K$) $\supset$ Has(X,m) $\wedge$ Has(X,K) $\wedge$Send (X, $\{|m|\}_K$)

Το παραπάνω αξίωμα που σημαίνει “Created and Sent” μας λέει ότι ο thread X ξέρει το m (δεν είναι απαραίτητο να το έχει δημιουργήσει αυτός), ξέρει το κλειδί K και έστειλε το μήνυμα $\{|m|\}_K$.

ORIG : <->New(X,n) $\supset$ Has(X,n)

REC : <->Receive(X,x) $\supset$ Has(X,x)

Τα δύο τελευταία αξιώματα συνδέουν τη γνώση που έχει κάποιος thread με τις δράσεις που έχουν εκτελεστεί το παρελθόν. Συγκεκριμένα, λένε ότι ένας thread κατέχει έναν όρο εάν είτε τον δημιούργησε (freshly-nonce-) είτε τον έλαβε σε κάποιο μήνυμα.

Τα αξιώματα που ακολουθούν (AR1, AR2 και AR3) επιτρέπουν την ένθεση των κατάλληλων υποκαταστάσεων οι οποίες έχουν προκύψει μετά τις δράσεις τύπου

pattern matching, signature verification και decryption, οι οποίες έχουν επενεργήσει στο κατηγόρημα δράσης (action predicate) a.

AR1 : $a(x)\ [q(x)/q(t)]_X\ a(t)$

Έστω ότι Q είναι ένα πρωτόκολλο και $R=R_0R_1R_2$ ένα run του Q, τέτοιο ώστε το $R_1|_X$ να ταιριάζει το (q(x)/q(t)) υπό την υποκατάσταση σ και $Q,R_0 \models \sigma a(x)$. Πρέπει να δείξουμε ότι $Q,R_0R_1 \models \sigma a(t)$. Αφού το $R_1|_X$ ταιριάζει το (q(x)/q(t)) υπό τη σ και τα events του R_1 περιέχουν μόνο ground όρους θα πρέπει η σx να ταυτίζεται με τη σt και επομένως $Q,R_0 \models a(t)$. Προφανώς, οι φόρμουλες του a(t) παραμένουν true καθώς εκτελούνται καινούριες πράξεις, άρα $Q,R_0R_1 \models \sigma a(t)$.

AR2 : $a(x)\ [(\{|t|\}_{\bar{K}}/\{|t|\}_K)]_X\ a(Sign(A,\{|t|\}_{\bar{K}}))$

AR3 : $a(x)\ [\{|x|\}_K/\{|y|\}_{\bar{K}}]_X\ a(\{|y|\}_K)$

<u>Αξιώματα για Encryption και Signature :</u>

SEC : $Honest(\hat{X}) \wedge Decrypt(Y, \{|n|\}_X) \supset (\hat{Y} = \hat{X})$

Το αξίωμα αυτό μας λέει ότι αν ένας principal $\hat{X}$ είναι honest και ένας thread Y αποκρυπτογράφησε το μήνυμα$\{|n|\}_X$ (που είναι κρυπτογραφημένο με το δημόσιο κλειδί του $\hat{X}$) τότε ο $\hat{Y}$ πρέπει να ταυτίζεται με τον $\hat{X}$. Δηλαδή, αν ένας principal είναι honest, τότε μόνο αυτός μπορεί να αποκρυπτογραφήσει τα μηνύματα που είναι κρυπτογραφημένα με το δημόσιο κλειδί του.

VER : $Honest(\hat{X}) \wedge Verify(Y, \{|n|\}_{\bar{X}}) \wedge \hat{Y} \neq \hat{X} \supset \exists X \exists m.$(<->Send(X,m) $\wedge$ Contains(m, $\{|n|\}_{\bar{X}}$))

Έστω Q ένα πρωτόκολλο και C μία initial configuration τέτοια ώστε $\hat{X} \in$ HONEST(C). Έστω R ένα run του Q που ξεκινάει από την C έτσι ώστε Q,R |= <-> Verify(Y, $\{|n|\}_{\bar{X}}$). Όταν $\hat{X} \in$ HONEST(C) μόνο τα threads του $\hat{X}$ μπορούν να δημιουργήσουν ψηφιακές υπογραφές με το ιδιωτικό κλειδί του $\hat{X}$. Επειδή $\hat{Y} \neq \hat{X}$, θα πρέπει ο thread Y να έχει λάβει τον όρο $\{|n|\}_{\bar{X}}$ ως τμήμα κάποιου μηνύματος m' (δηλαδή $\exists m'$ τέτοιο ώστε EVENT(R, Y, (x), m', x) και $\{|n|\}_{\bar{X}} \subseteq m'$). Από το λήμμα 2.3 υπάρχει μία δράση send που αντιστοιχεί σε κάθε δράση receive κι έτσι θα υπάρχει ένας thread Z τέτοιος ώστε το EVENT(R, Z, <m>, $\varnothing,\varnothing$) είναι true. Άρα, υπάρχει τουλάχιστον μία δράση στην R όπου το $\{|n|\}_{\bar{X}}$ έχει σταλεί ως τμήμα κάποιου μηνύματος. Έστω R' το ελάχιστο πρόθεμα του R για το οποίο για κάποιο thread Z και για κάποιον όρο m για τον οποίο $\{|n|\}_{\bar{X}} \subseteq$ m, να είναι true το EVENT(R', Z, <m>, $\varnothing,\varnothing$). Από το λήμμα 2.1 (no telepathy), το $\{|n|\}_{\bar{X}}$ θα πρέπει είτε να έχει ληφθεί είτε δημιουργηθεί από τον thread Z και, αφού το R' είναι το ελάχιστο run στο οποίο έχει σταλεί το $\{|n|\}_{\bar{X}}$ ως τμήμα κάποιου μηνύματος, θα πρέπει ο Z να έχει δημιουργήσει το $\{|n|\}_{\bar{X}}$. Από τον ορισμό της εκτέλεσης μοντέλου (excecution model) και το γεγονός ότι ο $\hat{X}$ είναι honest, έπεται ότι το Z είναι ένα thread του $\hat{X}$. Τέλος, από τη σημασιολογία των χρονικών τελεστων και το λήμμα της ασύγχρονης επικοινωνίας συνεπάγεται ότι Q,R |= <->Send(Z,m) $\wedge$ Contains(m, $\{|n|\}_{\bar{Z}}$).

Διαισθητικά, το παραπάνω αξίωμα μας λέει ότι εάν ο principal $\hat{X}$ είναι honest και κάποιος thread Y έχει κάνει verification σε ένα μήνυμα που έχει υπογραφεί με το ιδιωτικό κλειδί του $\hat{X}$, θα πρέπει ο $\hat{X}$ να έχει στείλει την υπογραφή του σε κάποιο thread X ως τμήμα ενός μηνύματος, δηλαδή, εάν ο $\hat{X}$ είναι honest τότε μόνο αυτός μπορεί να υπογράψει μηνύματα με το δημόσιο κλειδί του.

Αξιώματα που αφορούν τη μοναδικότητα των nonces [2,4,10,11,19]

N1 : <->New(X,n) $\wedge$ <->New(Y,n) $\supset$ (Y=X)

Το αξίωμα αυτό μας λέει ότι εάν δύο threads έχουν δημιουργήσει κατά το παρελθόν το ίδιο nonce τότε θα πρέπει αυτά να ταυτίζονται.

N2 : After(New(X,n_1), New(X,n_2)) $\supset$ ($n_1 \neq n_2$)

Το παραπάνω αξίωμα μας λέει ότι εάν δύο nonces δημιουργηθούν μέσα στο ίδιο thread σε διαφορετικές χρονικές στιγμές (κατά τη διάρκεια δύο διαφορετικών δράσεων), τότε πρέπει να είναι διαφορετικά.

F1 : <->Fresh(X,n) $\wedge$ Fresh(Y,n) $\supset$ (X=Y)

Το αξίωμα αυτό μας λέει ότι αν ένα nonce n είναι fresh σε δύο διαφορεικά threads τότε θα πρέπει αυτά να ταυτίζονται.

Αξίωμα που αφοά τη σχέση υποόρων : [4]

CON : Contains((x,y), x) $\wedge$ Contains((x,y), y)

Δηλαδή κάθε tuple περιέχει τα τμήματά του.

Αξιώματα διατήρησης (modal axioms) : [4]

P1 : Persist(X,t) $[\alpha]_X$ Persist(X,t) όπου Persist $\in$ {<->, a, FirstSend, Gen, Has}

Αυτό το αξίωμα μας λέει ότι τα κατηγορήματα που ανήκουν στο {<->, FirstSend, Gen, a, Has} διατηρούνται ως true και μετά την εκτέλεση επιπλέον πράξεων.

P2 : Fresh(X,t) $[\alpha]_X$ Fresh(X,t) όπου $t \not\sqsubset \neq \alpha$

Δηλαδή η freshness ενός όρου t διατηρείται μετά την τέλεση δράσεων από κάποιο thread εφόσον ο t δεν είναι υποόρος κάποιου όρου που χρησιμοποιείται στη δράση του X κι έτσι αν η δράση περιλαμβάνει αποστολή να μην υπάρχει κίνδυνος να σταλεί και ο t.

P3 : HasAlone(X,n) $[\alpha]_X$ HasAlone(X,n) όπου $n \neq \not\sqsubset_{\cup} \alpha$ ή $\alpha \neq$ <m>

Όπου το κατηγόρημα (predicate) HasAlone ορίζεται ως εξής :

HasAlone(X,t) $\equiv$ Has(X,t) $\wedge$ Has(Y,t) $\supset$ X=Y

Σύμφωνα με το παραπάνω αξίωμα ένα nonce παραμένει μυστικό εφόσον δε σταλεί ως τμήμα κάποιου μηνύματος m.

F : ϕ $[<m>]_X$ $\neg$ Fresh(X,t) όπου $t \sqsubseteq m$

Δηλαδή η freshness χάνεται όταν ένας όρος στέλνεται μέσω ενός μηνύματος που τον περιέχει.

<u>Αξιώματα και κανόνες για τη χρονική διάταξη :</u> [2,4]

FS1 : Fresh(X,t) $[<t'>]_X$ FirstSend(X,t,t') όπου $t \sqsubseteq t'$

Το αξίωμα αυτό μας λέει ότι αν ο thread X δημιούργησε έναν fresh όρο t και στη συνέχεια τον έστειλε με έναν όρο t' ο οποίος τον περιέχει τότε αυτό ήταν το πρώτο fresh event τύπου send.

FS2 : $\text{FirstSend}(X,t,t') \wedge a(Y,t'') \supset \text{Send}(X,t') < a(Y,t'')$ όπου $X \neq Y$ και $t \subseteq t''$

Το αξίωμα αυτό μας λέει ότι αν ένας thread Y εκτελέσει κάποια δράση με έναν όρο t'' που περιέχει τον t (ο οποίος έχει πρώτα σταλεί μέσα στον όρο t' από τον thread X) τότε αυτή η δράση τύπου send θα πρέπει να έχει προηγηθεί της a.

AF0 : $\text{Start}(X)\ []_X \neg <->a(X,t)$

Δηλαδή, πριν ο thread X αρχίσει να εκτελεί δράσεις είναι true ότι δεν είχε εκτελέσει άλλες δράσεις στο παρελθόν.

AF1 : $\theta[\alpha_1...\alpha_n]_X\ \text{After}(a_1,a_2) \wedge \ldots \wedge \text{After}(a_{n-1},a_n)$

Αυτό το αξίωμα λέει ότι αν ένας thread σκτελέσει τις δράσεις $\alpha_1...\alpha_n$ με αυτή τη σειρά, τότε το κατηγόρημα $\text{After}(a_i,a_{i+1})$ είναι αληθές για κάθε i=1,…,n-1.

AF2 : $(<->b_1(X,t_1) \wedge (-)\text{Fresh}(X,t)) \wedge <->b_2(Y,t_2) \supset \text{After}(b_1(X,t_1),b_2(Y,t_2))$
όπου $t \subseteq t_2$ και $X \neq Y$

Αυτό το αξίωμα λέει ότι όλες οι δράσεις που εμπλέκουν τον όρο t -ο οποίος ήταν fresh σε κάποια στιγμή- θα πρέπει να έχουν συμβεί μετά την πρώτη φορά που ο t στάλθηκε.

<u>Αξιώματα που αφορούν το πρωτόκολλο ανταλλαγής κλειδιού Diffie-Hellman :</u> [4,5,7]

Ορίζουμε πρώτα ένα κατηγόρημα (predicate) που είναι χρήσιμο :

$\text{Computes}(X,g^{\alpha b}) \equiv (\ (\ \text{Has}(X,\alpha) \wedge \text{Has}(X,g^b)\ )\ \vee\ (\ \text{Has}(X,b) \wedge \text{Has}(X,g^{\alpha})\ )\ )$

Το οποίο έχει την πρωφανή ερμηνεία σε σχέση με το Diffie-Hellman, δηλαδή ότι για να υπολογίσει κανείς το $g^{\alpha b}$ θα πρέπει να γνωρίζει είτε τα α και g^b είτε τα b και g^{α} .

DH1 : Computes(X,$g^{\alpha b}$) $\supset$ Has(X,$g^{\alpha b}$)

Το παραπάνω αξίωμα μας λέει ότι αν ένας thread X έχει όλες τις απαραίτητες πληροφορίες (όπως αυτές ορίζονται απο το συγκεκριμένο πρωτόκολλο) για να υπολογίσει το $g^{\alpha b}$, τότε επίσης "έχει" δηλαδή γνωρίζει το $g^{\alpha b}$.

DH2 : Has(X,$g^{\alpha b}$) $\supset$ (Computes(X,$g^{\alpha b}$) $\vee$ $\exists$m.(<->Receive(X,m) $\wedge$ Contains(m, $g^{\alpha b}$)))

Δηλαδή ο μόνος τρόπος για να έχει κάποιος ένα μυστικό Diffie-Hellman είναι είτε να το έχει υπολογίσει από τον εκθέτη και το εκθετικό είτε να το έχει λάβει σε κάποιο μήνυμα στο παρελθόν.

DH3 : (<->Receive(X,m) $\wedge$ Contains(m, $g^{\alpha b}$)) $\supset$ $\exists$Y,m'.(Computes(Y, $g^{\alpha b}$) $\wedge$ <->Send(Y,m') $\wedge$ Contains(m', $g^{\alpha b}$)

Έστω R ένα run στο οποίο ο thread X δέχεται ένα μήνυμα m που περιέχει το $g^{\alpha b}$. Από το λήμμα 2.3 έχουμε ότι στη run R υπάρχει κάποιος που έστειλε το μήνυμα m που περιέχει το $g^{\alpha b}$. Έστω R' το μικρότερο πρόθεμα του R, στο οποίο κάποιος thread Y στέλνει κάποιο μήνυμα m' που περιέχει το $g^{\alpha b}$. Αφού το R' είναι το μικρότερο τέτοιο πρόθεμα σημαίνει ότι ο Y δεν θα μπορούσε να έχει ήδη δεχθεί ένα άλλο μήνυμα m'' που να περιέχει το $g^{\alpha b}$. Άρα, από το DH2 προκύπτει ότι ο Y θα πρέπει να έχει υπολογίσει ο ίδιος το $g^{\alpha b}$.

DH4 : Fresh(X,α) $\supset$ Fresh(X, g^{α})

Δηλαδή αν ένας εκθέτης του D-H είναι fresh θα πρέπει και το αντίστοιχο εκθετικό να είναι fresh.

Καθολικοί κανόνες (generic rules) : [2,4,12]]

Οι γενικοί αυτοί κανόνες χρησιμοποιούνται για reasoning που αφορά προ-συνθήκες (pre-conditions) και μετασυνθήκες (post-conditions) προγραμμάτων.

$$\frac{\theta[\mathrm{P}]_{\mathrm{X}}\varphi \qquad \theta[\mathrm{P}]_{\mathrm{X}}\psi}{\theta[P]_{X}\varphi \wedge \psi}$$ **G1**

$$\frac{\theta[\mathrm{P}]_{\mathrm{X}}\psi \qquad \varphi[\mathrm{P}]_{\mathrm{X}}\psi}{\theta \vee \varphi[P]_{X}\psi}$$ **G2**

$$\frac{\theta' \supset \theta \qquad \theta[\mathrm{P}]_{\mathrm{X}}\varphi \qquad \phi \supset \varphi'}{\theta'[P]_{\mathrm{X}}\varphi'}$$ **G3**

$$\frac{\varphi}{\theta[P]_{\mathrm{X}}\varphi}$$ **G4**

Seqeuncing Rule :

$$\frac{\varphi_1[\mathrm{P}]_{\mathrm{X}}\varphi_2 \qquad \varphi_2[\mathrm{P}']_{\mathrm{X}}\varphi_3}{\varphi_1[\mathrm{PP}']_{\mathrm{X}}\varphi_3}$$ **S1**

Ο κανόνας αυτός μας δίνει τη δυνατότητα να εφαρμόσουμε sequential composition σε δύο cords P και P’ όταν η μετασυνθήκη του P ταιριάζει με την προσυνθήκη του P’.

Απόδειξη : Έστω Q ένα πρωτόκολλο και R ένα run του Q τέτοιο ώστε Q,R $|= \phi_1[P]_A\phi_2$ και Q,R $|= \phi_2[P']_A\phi_3$. Πρέπει να δείξουμε ότι Q,R $|= \phi_1[PP']_A\phi_3$. Έστω $R=R_0R_1R_2$. Υποθέτουμε ότι το $R_1|_A$ ταιριάζει (matches) το PP’ υπό την υποκατάσταση σ και ότι $Q,R_0 |= \sigma\phi_1$. Η run R μπορεί να γραφεί και ως $R=R_0R_1'R_1''R_2$ όπου $R_1'|_A$ ταιριάζει την P υπό

τη σ και $R_1''|_A$ ταιριάζει την P' υπό τη σ. Άρα, $Q,R_0R_1' \models \sigma\phi_2$ και επομένως $Q,R_0R_1'R_1'' \models \sigma\phi_3$.

Preservation Rules (κανόνες διατήρησης) :

Οι διατήρηση μερικών ιδιοτήτων έχει ήδη αναφερθεί και στα αξιώματα διατήρησης. Ωστόσο, μία αναδιατύπωση με τη μορφή κανόνων διατήρησης είναι αρκετά διαφωτιστική. Επίσης περιέχονται κανόνες για τη διατήρηση του κατηγορήματος Source.

Παρακάτω, όπου Persist $\in$ {Has, Send, Decrypts}

$$\frac{[\mathrm{P}]_A\, Persist(X,t)}{[P < m >]_A\, Persist(X,t)}$$ P1

$$\frac{[\mathrm{P}]_A\, Persist(X,t)}{[P(x)]_A\, Persist(X,t)}$$ P2

$$\frac{[\mathrm{P}]_A\, Persist(X,t)}{[P(vx)]_A\, Persist(X,t)}$$ P3

$$\frac{[\mathrm{P}]_A\, Persist(X,t)}{[P(u / q(x))]_A\, Persist(X,t)}$$ P4

$$\frac{[\mathrm{P}]_A\, Source(m,Y,t,K)}{[P < t' >]_A\, Source(m,Y,t,K)}$$, (m $\not\subset\neq$ t') P5

$$\frac{[\mathrm{P}]_A\, Source(m,Y,t,K)}{[P(x)]_A\, Source(m,Y,t,K)}$$ P6

$$\frac{[\mathrm{P}]_A\, Source(m,Y,t,K)}{[P(vx)]_A\, Source(m,Y,t,K)}$$ P7

$$\frac{[\mathrm{P}]_A\, Source(m,Y,t,K)}{[P(u / q(x))]_A\, Source(m,Y,t,K)}$$ P8

Honesty rule :

Είδαμε στο κεφάλαιο 1 ότι ένα πρωτόκολλο είναι ένα σύνολο $Q=\{\rho_1,...,\rho_k\}$ που αποτελείται από ρόλους ρ_i, i=1,…k, καθένας από τους οποίους εκτελείται από μηδέν ή περισσότερους honest principals σε κάθε run του Q.

Μία ακολουθία P από δράσεις είναι μία βασική ακολουθία (basic sequence) του ρόλου ρ και γράφεται $P \in BS(\rho)$, εάν η P είναι μία υπακολουθία του ρ τέτοια ώστε :
i)είτε ξεκινάει από την αρχή του ρ και καταλήγει στην τελευταία δράση πριν από το πρώτο receive
ii)είτε ξεκινάει από ένα receive και συνεχίζει μέχρι την τελευταία δράση πριν από το επόμενο receive
iii)είτε ξεκιναέι από το τελευταίο receive και φθάνει μέχρι το τέλος του ρόλου.

Με βάση τα παραπάνω και για $\rho \in Q$ μπορούμε να διατυπώσουμε τον honesty rule :

$$\frac{\mathrm{Start}(X)[]_X\varphi \qquad \forall \rho \in Q.\forall P \in BS(\rho).\phi[P]_X\varphi}{Honest(\hat{X}) \supset \varphi} \quad \mathbf{HON_Q}$$

όπου δεν υπάρχουν ελεύθερες μεταβλητές στο φ εκτός του X που είναι δεσμευμένο στο $[P]_X$.

Δηλαδή, εάν η φ ισχύει για κάθε ρόλο του πρωτοκόλλου Q και διατηρείται για κάθε basic sequence του κάθε ρόλου, τότε κάθε honest principal που εκτελεί το πρωτόκολλο θα πρέπει να ικανοποιεί τη φ. Ο περιορισμός για τις ελεύθερες μεταβλητές αποτρέπει οποιαδηποτε δέσμευση τους στο συμπέρασμα Honest($\hat{X}$) $\supset$ φ . Με άλλα λόγια, ο honesty rule λέει ότι εάν μία ιδιότητα ισχύει πριν την έναρξη κάθε ρόλου και διατηρείται μετά από οποιασδήποτε ακολουθία δράσεων η οποία μπορεί να εκτελεσθεί από honest principal, τότε η ιδιότητα αυτή ισχύει για κάθε honest principal. [2,10]

Απόδειξη : Έστω ότι Q είναι ένα πρωτόκολλο και R ένα run του Q τέτοιο ώστε Q,R |= Start(X)[]$_X$ ϕ και Q,R |= ϕ[P]$_X$ϕ $\forall \rho \in Q$ και $\forall P \in BS(\rho)$. Πρέπει να δείξουμε ότι Q,R |= Honest($\hat{X}$)$\supset$ϕ . Έστω Q,R |= Honest($\hat{X}$) . Από τη σημασιολογία του κατηγορήματος honest και το λήμμα A.3.4 θα πρέπει R|$_X$ να είναι trace κάποιου ρόλου του Q που έχει εκτελεσθεί από τον X και, επιπλέον, ο thread X να βρίσκεται σε κατάσταση παύσης στο τέλος του R. Επομένως, το R|$_X$ είναι παράθεση (concatenation) από basic sequences του Q. Άρα, από την ορθότητα του sequencing rule S1 συνεπάγεται ότι Q,R |= ϕ . □

<u>Το θεώρημα της ορθότητας (soundness theorem) :</u> [2,3,4,10,11]

Όταν γράφουμε Γ |- γ εννοούμε ότι το γ μπορεί να αποδειχθεί από τις φόρμουλες στο γ και οποιοδήποτε αξίωμα ή inference rule του συστήματος αποδείξεων εκτ;ο απο τον honesty rule.

Όταν γράφουμε Γ |-$_Q$ γ εννοούμε ότι το γ μπορεί να αποδειχθεί με τη χρήση φορμουλών του Γ, βασικών αξιωμάτων και inference rules του αποδεικτικού συστήματος και του honesty rule για το πρωτόκολλο Q.

Στα παραπάνω όπου γ μπορεί να είναι modal formula ή basic formula.

<u>Θεώρημα :</u> Εάν Γ |-$_Q$ γ τότε Γ |=$_Q$ γ . Επιπλέον, εάν Γ|- γ τότε Γ |= γ.

ΚΕΦΑΛΑΙΟ 4

Αποδεικτικές Μέθοδοι της PCL

4.1.Εισαγωγή

Στο προηγούμενο κεφάλαιο ασχοληθήκαμε με τη σημασιολογία καθώς και με τα βασικά αξιώματα και κανόνες που στοιχειοθετούν το αποδεικτικό σύστημα της PCL. Επίσης, είδαμε το θεώρημα της ορθότητας και παραθέσαμε ενδεικτικά κάποιες αποδείξεις ορθότητας κανόνων και αξιωμάτων. Στο παρόν κεφάλαιο θα αναλύσουμε τις δύο αποδεικτικές μεθόδους στην PCL, οι οποίες είναι η συνθετική αποδεικτική μέθοδος (compositional proof method) και η abstraction-refinment αποδεικτικ

ή μέθοδος.

4.2.Η σύνθεση ως αποδεικτική μέθοδος

Η σημερινή πραγματικότητα απαιτεί από τα πρωτόκολλα προηγμένες λειτουργίες, με αποτέλεσμα να είναι απαραίτητη η χρήση σύνθετων πρωτοκόλλων. Ως εκ τούτου, προκειμένου να επιτευχθεί το reasoning τους, μία εύστοχη μέθοδος είναι να επιστρατευθούμε το reasoning των επι μέρους απλούστερων πρωτοκόλλων που τα συνιστούν.

Στο compositional security πρέπει να εξασφαλίσουμε δύο βασικές προϋποθέσεις :
i)τον προσθετικό συνδυασμό (additive combination), όπου θέλουμε να συνδυάσουμε τμήματα πρωτοκόλλων με τέτοιο τρόπο ώστε να συσσωρεύουν security ιδιότητες και
ii)το μη καταστρεπτικό συνδυασμό (nondestructive combination), όπου θέλουμε να διασφαλίσουμε ότι κατά τον συνδυασμό διαφορετικών τμημάτων δε θα υποβαθμιστούν οι επί μέρους security ιδιότητες. [2]

Στον additive combination χρησιμοποιούμε before-after ισχυρισμούς για να κάνουμε reasoning στα βήματα της εκτέλεσης ενός πρωτοκόλλου. Έτσι, μπορούμε να συνδυάσουμε ισχυρισμούς διαφορετικών βημάτων και να πάρουμε ιδιότητες σε μια ακολουθία από βήματα. Για παράδειγμα, αν $\phi[P]_A\psi$ και $\psi[P']_A\theta$ τότε $\phi[PP']_A\theta$.

Ο nondestructive combination εξασφαλίζεται με τη χρήση ισχυρισμών σταθερότητας (invariance assertions). Ο κεντρικός ισχυρισμός στο reasoning system που χρησιμοοποιούμε είναι ο $\Gamma \vdash \phi[P]_A\psi$ που λέει ότι σε κάθε πρωτόκολλο που κανοποιεί τη σταθερά (invariant) Γ, ο before-after ισχυρισμός $\phi[P]_A\psi$ ισχύε σε κάθε run. Γενικά,οι σταθερές μας είναι δηλώσεις για principals που ακολουθούν τους κανόνες ενός πρωτοκόλλου, όπως για παράδειγμα τα τελικά συμπεράσματα. Με άλλα λόγια, ο nondestructive combination συμβαίνει όταν συνδυάζονται δύο πρωτόκολλα και κανένα δεν παραβιάζει τις invariants του άλλου.

Θα μπορούσε να πει κανείς ότι όταν έχουμε additive combination, αφού έχουμε συσσώρευση ιδιοτήτων θα έχουμε κατά κάποιο τρόπο και nondestructive combination, δηλαδή ότι οι δύο έννοιες, κατά κάποιο τρόπο αλληλοεπικαλύπτονται. Στην PCL όμως, όταν θα λέμε additive combination θα εννοούμε την προσθήκη επιπλέον βημάτων σε ένα πρωτόκολλο υπό τη θεώρηση κάποιων συγκεκριμένων invariants, ενώ όταν θα λέμε nondestructive combination θα εννοούμε ότι το σύνολο αυτών των επιπλέον βημάτων διατηρεί το σύνολο των invariants.

4.3.Θεωρήματα Σύνθεσης (Composition Theorems)

Στην PCL υπάρχουν δύο είδη σύνθεσης πρωτοκόλλων : η sequetial composition και η parallel composition. Πρόκειται για λειτουργίες που λαμβάνουν χώρα στα cords, τα οποία συνιστούν τα πρωτόκολλα ως ρόλοι τους.

Παράλληλη σύνθεση (parallel composition) $Q_1|Q_2$ των πρωτοκόλλων Q_1 και Q_2 είναι η ένωση των συνόλων των cords των Q_1 και Q_2 . [2,4]

Σε αυτή την περίπτωση ελλοχεύει ο κίνδυνος κάποια ιδιότητα που έχει αποδειχθεί ξεχωριστά για κάθε ένα από τα πρωτόκολλα να μην ισχύει κατά το παράλληλο τρέξιμό τους, γιατί κάποια πληροφορία που προέκυψε από την εκτέλεση του ενός να χρησιμοποιηθεί από έναν intruder για να κάνει επίθεση στο άλλο. Αυτό στη λο/γική ανάλυση του πρωτοκόλλου που βάλλεται σημαίνει ότι πλέον δεν ισχύει κάποια ιδιότητα και επειδή έχει γίνει το reasoning πριν τη σύνθεση, θα πρέπει να ισχύουν όλα τα αξιώματα και οι κανόνες εκτός από αυτούς που αποδείχθηκαν κάνοντας χρήση του honesty rule (π.χ. οι invariants του πρωτοκόλλου).

Για να επιτευχθεί λοιπόν η διατήρηση των επι μέρους ιδιοτήτων των πρωτοκόλλων κατά την παράλληλη σύνθεση πρέπει να διασφαλιστεί ότι κάθε πρωτόκολλο σέβεται τις invariants του άλλου. Κάτι τέτοιο επιτυγχάνεται ακολουθώντας τη μεθοδολογία των τεσσάρων βημάτων που ακολουθεί :

Βημα 1 : Αποδεικνύουμε ξεχωριστά τις security ιδιότητες καθενός από τα πρωτόκολλα Q_1 και Q_2 .

$$|\text{-}_{Q_1}\ \Psi_1 \quad \text{και} \quad |\text{-}_{Q_2}\ \Psi_2$$

Βήμα 2 : Προσδιορίζουμε τα σύνολα των invariants που έχουν χρησιμοποιηθεί στις δύο αποδείξεις Γ_1 και Γ_2 . Οι φόρμουλες που περιέχονται σε αυτά τα σύνολα τυπικά θα είναι οι φόρμουλες στις δύο αποδείξεις οι οποίες αποδείχθηκαν χρησιμοποιώντας τον honesty rule. Οι αποδείξεις από το προηγούμενο βήμα μπορούν να αποσυντεθούν σε

δύο μέρη : το πρώτο μέρος να αποδεικνύει τις protocol invariants χρησιμοποιώντας τον honesty rule, ενώ το δεύτερο μέρος να αποδεικνύει την ιδιότητα χρησιμοποιώντας τις invariants ως υποθέσεις αλλά χωρίς τη χρήση του honesty rule. Δηλαδή :

$$|\text{-}_{Q_1} \Gamma_1 \text{ και } \Gamma_1|\text{-}\Psi_1 \text{ και } |\text{-}_{Q_2} \Gamma_2 \text{ και } \Gamma_2|\text{-}\Psi_2$$

Βήμα 3 : Είναι δυνατόν να εξασθενίσουμε την υπόθεση σε $\Gamma_1 \cup \Gamma_2$. Η απόδιεξη των ιδιοτήτων των πρωτοκόλλων προφανώς διατηρείται κάτω από ένα μεγαλύτερο σύνολο υποθέσεων.

$$\Gamma_1 \cup \Gamma_2 |\text{-}\ \Psi_1 \text{ και } \Gamma_1 \cup \Gamma_2 |\text{-}\ \Psi_2$$

Βήμα 4 : Αποδεικνύουμε ότι οι invariants $\Gamma_1 \cup \Gamma_2$ ισχύουν και για τα δύο πρωτόκολλα. Αυτό το βήμα χρησιμοποιεί τη μεταβατική ιδιότητα στη λογική : εάν $|\text{-}_Q \Gamma$ και $\Gamma|\text{-}\gamma$ τότε $|\text{-}_Q \gamma$. Αφού το $|\text{-}_{Q_1} \Gamma_1$ έχει ήδη αποδειχθεί στο πρώτο βήμα, στο παρόν βήμα ακεί να δείξουμε ότι $|\text{-}_{Q_1} \Gamma_2$ και αντίστοιχα $|\text{-}_{Q_2} \Gamma_1$. Από το λήμμα 3.1 που ακολουθεί συνεπάγεται ότι $|\text{-}_{Q_1 \| Q_2} \Gamma_1 \cup \Gamma_2$. Από αυτό και τις φόρμουλες του δεύτερου βήματος μπορούμε να συμπεράνουμε ότι οι security ιδιότητες των πρωτοκόλλων Q_1 και Q_2 διατηρούνται υπό την παράλληλη σύνθεσή τους.

$$|\text{-}_{Q_1 \| Q_2} \Psi_1 \text{ και } |\text{-}_{Q_1 \| Q_2} \Psi_2$$

Λήμμα 4.1 : Εάν $|\text{-}_{Q_1} \phi$ και $|\text{-}_{Q_2} \phi$ τότε $|\text{-}_{Q_1 \| Q_2} \phi$, όπου το τελευταίο βήμα στην απόδειξη του φ και στο Q_1 και στο Q_2 χρησιμοποιεί τον honesty rule και κανένα άλλο προηγούμενο βήμα δε χρησιμοποιεί τον honesty rule. [2]

Απόδειξη : Έστω ότι η φόρμουλα Honest($\hat{X}$) $\supset \phi$ μπορεί να αποδειχθεί και στο Q_1 και στο Q_2 χρησιμοποιώντας τον honesty rule. Από τον ορισμό του honesty rule, θα πρέπει να ισχύει η $|\text{-Start(X)}[]_x \phi$ και η $\forall \rho \in Q_1 \cup Q_2 . \forall P \in BS(\rho) | \text{-} \varphi[\mathrm{P}]_{\mathrm{X}} \varphi$. Κάθε basic

sequence P ενός ρόλου στο $Q_1|Q_2$ είναι μία basic sequence ενός ρόλου στο Q_1 ή μία basic sequence ενός ρόλου στο Q_2 . Επομένως $|-\phi[P]_X\phi$, άρα με εφαρμογή του honest rule $|-_{Q_1|Q_2}$ Honest($\hat{X}$) $\supset \phi$.

Θεώρημα 4.1 : Εάν $|-_{Q_1}\Gamma$ και $\Gamma|-\Psi$ και $|-_{Q_2}\Gamma$, τότε $|-_{Q_1|Q_2}\Psi$. [2]

Sequential Composition : Ένα πρωτόκολλο Q είναι η sequential composition δύο πρωτοκόλλων Q_1 και Q_2 , εάν κάθε ρόλος του Q έχει προκύψει από τη sequential composition ενός cord του Q_1 με ένα cord του Q_2 . [2]

Υπενθυμίζουμε ότι εάν έχουμε δύο κλειστά cords $r=(x_0\ldots x_{l-1})[R]_X<u_0\ldots u_{m-1}>$ και $s=(y_0\ldots y_{m-1})[S]_Y<t_0\ldots t_{n-1}>$, η sequential σύνθεσή τους ορίζεται ως

$$r;s=(x_0\ldots x_{l-1})[RS']_X<t_0'\ldots t_{n-1}'>$$

όπου τα S' και t_i' είναι τα στιγμιότυπα υποκατάστασης των S και t αντίστοιχα, έτσι ώστε κάθε μεταβλητή y_k να έχει αντικατασταθεί από τον όρο u_k . Επιπλέον, υπό αυτή την υποκατάσταση, το Y αντιστοιχίζεται στο X. Οι μεταβλητές μετονομάζονται, έτσι ώστε οι ελεύθερες μεταβλητές των S, t_j και u_k να μη δεσμευθούν στο r;s. Το RS' είναι το strand το οποίο προκύπτει από παράθεση (concatenation) των δράσεων του R με αυτές του S'.

Με τη sequential composition μπορεί να προκύψουν διαφορετικά πρωτόκολλα, ανάλογα με τους ρόλους που θα διαλέξουμε για να κάνουμε τη σύνθεση. Γι΄αυτό και, συνήθως, δεν αναφερόμαστε στη sequential composition δύο πρωτοκόλλων, αλλά σε μία sequential composition των δύο πρωτοκόλλων. Όπως και στην περίπτωση της παράλληλης σύνθεσης, θα πρέπει τα πρωτόκολλα που συντίθενται να σέβονται το ένα τις invariants του άλλου. Επίσης, εάν έχουμε να συνθέσουμε sequentially δύο ρόλους με δράσεις P και P' θα πρέπει η post-condition του P να ταιριάζει με την pre-condition του P'. Έτσι, όπως και στην παράλληλη σύνθεση, και για την περίπτωση της sequential composition έχουμε μία μεθοδολογία πέντε βημάτων που εξασφαλίζουν την ασφαλή απόδειξη ιδιοτήτων :

Βήμα 1 : Αποδεικνύουμε ξεχωριστά τις security ιδιότητες τνω πρωτοκόλλων Q_1 και Q_2 .

$$|\text{-}_{Q_1} \Psi_1 \quad \text{και} \quad |\text{-}_{Q_2} \Psi_2$$

Βήμα 2 : Προσδιορίζουμε το σύνολο των invariants που έχουν χρησιμοποιηθεί στις δύο αποδείξεις, Γ_1 και Γ_2 . Οι φόρμουλες που περιλαμβάνονται σε αυτά τα σύνολα τυπικά θα είναι οι φόρμουλες των δύο αποδείξεων που έχουν αποδειχθεί χρησιμοποιώντας τον honesty rule. Οι αποδείξεις του προηγούμενου βήματος μπορούν να αποσυντεθούν σε δύο τμήματα : το πρώτο τμήμα αποδεικνύει τις invariants του πρωτοκόλλου χρησιμοποιώντας τον honesty rule για το πρωτόκολλο, ενώ το δεύτερο αποδεικνύει την ιδιότητα του πρωτοκόλλου χρησιμοποιώντας τις invariants ως υποθέσεις χωρίς όμως τη χρήση του honesty rule. Δηλαδή,

$$|\text{-}_{Q_1} \Gamma_1 \text{ και } \Gamma_1|\text{-}\Psi_1 \quad \text{και} \quad |\text{-}_{Q_2} \Gamma_2 \text{ και } \Gamma_2|\text{-}\Psi_2$$

Βήμα 3 : Εξασθενίζουμε την υπόθεση στο $\Gamma_1 \cup \Gamma_2$. Η απόδειξη των ιδιοτήτων του πρωτοκόλλου προφανώς διατηρείται κάτω από ένα μεγαλύτερο σύνολο υποθέσεων.

$$\Gamma_1 \cup \Gamma_2 |\text{-}\Psi_1 \quad \text{και} \quad \Gamma_1 \cup \Gamma_2 |\text{-}\Psi_2$$

Βήμα 4 : Εάν η post-condition της modal formula Ψ_1 ταιριάζει με την pre-condition της Ψ_2' , τότε αυτές μπορούν να συντεθούν sequentially εφαρμόζοντας τον sequencing rule S1. Εδώ, η Ψ_2' έχει προκύψει από την Ψ_2 μετά από αντικατάσταση των ελεύθερων μεταβλητών της όπως καθορίζει η sequential composition των αντίστοιχων cords. Αυτό διατηρεί τις φόρμουλες που έχουν αποδειχθεί σε προηγούμενα βήματα, αφού αυτές οι

φόρμουλες είναι true υπό όλες τις αντικαταστάσεις των ελεύθερων μεταβλητών. Υποθέτοντας ότι οι Ψ_1 και Ψ_2 είναι αντίστοιχα $\theta[P_1]\phi$ και $\phi[P_2]\psi$ έχουμε :

$$\Gamma_1 \cup \Gamma_2 \mid\!- \theta[P_1P_2]\psi$$

Βήμα 5 : Αποδεικνύουμε ότι οι invariants που έχουν χρησιμοποιηθεί για να αποδείξουμε τις ιδιότητες των πρωτοκόλλων, $\Gamma_1 \cup \Gamma_2'$, ισχύουν και για τα δύο πρωτόκολλα. Επειδή το $\mid\!-_{Q_1} \Gamma_1$ έχει ήδη αποδειχθεί στο βήμα 1, σε αυτό το βήμα αρκεί να δέιξουμε ότι $\mid\!-_{Q_1} \Gamma_2'$ και ομοίως ότι $\mid\!-_{Q_2} \Gamma_1$. Από το λήμμα 4.2 (που παρατίθεται παρακάτω) έχουμε ότι $\mid\!-_{Q_3} \Gamma_1 \cup \Gamma_2'$, όπου το Q_3 είναι η sequential σύνθεσή τους. Από αυτό και τις φόρμουλες των βημάτων 3 και 4 μπορούμε να συμπερένουμε ότι οι security ιδιότητες των Q_1 και Q_2 έχουν διατηρηθεί κάτω από τη sequential composition τους και επιπλέον ότι η φόρμουλα που ακολουθεί μπορεί να αποδειχθεί :

$$\mid\!-_{Q_3} \theta[P_1P_2]\psi$$

Λήμμα 4.2 : Εάν $\mid\!-_{Q_1} \phi$ και $\mid\!-_{Q_2} \phi$ τότε $\mid\!-_{Q_3} \phi$, όπου το Q_3 είναι μία sequential composition των Q_1 και Q_2 και το τελευταίο βήμα στην απόδειξη του φ και στο Q_1 και στο Q_2 χρησιμοποιεί τον honesty rule και κανένα από τα προηγούμενα βήματα δε χρησιμοποιεί τον honesty rule. [2,4]

Απόδειξη : Έστω ότι η φόρμουλα Honest($\hat{X}$) $\supset \phi$ μπορεί να αποδειχθεί και στο Q_1 και στο Q_2 χρησιμοοιώντας τον honesty rule. Από τον ορισμό του honesty rule θα πρέπει $\mid\!-\text{Start}(X)[]_x\phi$ και $\forall \rho \in Q_1 \cup Q_2 . \forall P \in BS(\rho) \mid\!- \varphi[P]_X \varphi$. Έστων ότι Q έιναι ένα πρωτόκολλο που έχει προκύψει από τη seuential composition των Q_1 και Q_2 . Κάθε basic sequence P ενός ρόλου στο Q θα πρέπει να είναι είτε μία basic sequence ενός ρόλου στο Q_1 είτε μία basic sequence ενός ρόλου στο Q_2 είτε μία παράθεση (concatenation) μίας basic sequence ενός ρόλου στο Q_1 και μίας basic sequence ενός

ρόλου στο Q_2 . Στις δύο πρώτες περιπτώσεις η |- $\phi[P]_X\phi$ ισχύει τετριμμένα ενώ στη τρίτη περίπτωση προκύπτει από μία εφαρμογή του sequencing rule S1. Έτσι, από την εφαρμογή του honesty rule |- ϱ Honest($\hat{X}$) $\supset \phi$.

Θεώρημα 4.2 : Εάν $\vdash_{\varrho_1} \Gamma_1$, Γ_1|-$\theta[P_1]\phi$;|-$_{\varrho_2}$ Γ_2 , Γ_2|-$\phi[P_2]\psi$; Γ_2 και |-$_{\varrho_1}$ Γ_2 , |-$_{\varrho_2}$ Γ_1 τότε |-$_{\varrho_3}$ $\theta[P_1P_2]\psi$, όπου Q_3 είναι μία sequential composition των Q_1 και Q_2 . [2,4]

Στην πράξη δε συναντάμε πάντα περιπτώσεις sequential composition όπου τα invariants είναι ανεξάρτητα από τη σειρά των ρόλων, αλλά πολλές φορές η συγκεκριμένη σειρά τους μπορεί να μπορεί να διευκολύνει τις αποδείξεις. Έστω για παράδειγμα ότι Q είναι η seuential composition των πρωτοκόλλων Q_1 και Q_2 και ότι $r_1;r_2$ είναι ένας ρόλος του Q όπου r_1 και r_2 είναι ρόλοι των Q_1 και Q_2 αντίστοιχα. Εάν θέλουμε να αποδείξουμε την invariance μίας φόρμουλας φ πάνω στον r_2 θα χρησιμοποιήσουμε μερικές πληροφορίες που έχουμε από την προηγούμενη εκτέλεση του r_1 .

Στο παρακάτω θεώρημα οι pre-conditions θ_{r_i} αναπαριστούν αυτές τις προηγούμενες πληροφορίες (history information). Η invariant induction είναι η συνηθισμένη επαγωγή για τον honesty rule ενισχυμένη από τις preconditions ενώ η pecondition induction μας διασφαλίζει ότι οι preconditions ισχύουν στη συγκεκριμένη κατάσταση της εκτέλεσης του πρωτοκόλλου.

Θεώρημα 4.3 : Εάν Q είναι μία sequential composition των πρωτοκόλλων Q_1 και Q_2 τότε μπορούμε να συμπεράνουμε ότι |-$_Q$ Honest($\hat{X}$) $\supset \phi$ εάν οι παρακάτων συνθήκες ισχύουν για κάθε $r_1;r_2$ στο Q, όπου $r_1 \in Q_1$ και $r_2 \in Q_2$:

(i) (Invariant induction)

$$\forall \mathrm{P} \in BS(r_1).\vdash\!-\theta_{r_1} \wedge \varphi[\mathrm{P}]_{\mathrm{X}}\varphi \quad \text{και} \quad \forall \mathrm{P} \in BS(r_2).\vdash\!-\theta_{r_2} \wedge \varphi[\mathrm{P}]_{\mathrm{X}}\varphi$$

(ii) (Precondition induction)

$\vdash \text{Start}(X)[]_X\theta_{r_1}$ και $\vdash \theta_{r_1} r_1 \theta_{r_2}$

$\forall P \in BS(r_1).\vdash \theta_{r_1}[P]_X\theta_{r_1}$ και $\forall P \in BS(r_2).\vdash \theta_{r_2}[P]_X\theta_{r_2}$

Με βάση το προηγούμενο θεώρημα προκύπτει η παρακάτω αποδεικτική μεθοδολογία :

Βήμα 1 : Αποδεικνύουμε ξεχωριστά τις security ιδιότητες Ψ_1 και Ψ_2 υποθέτοντας τα σύνολα των invariants Γ_1 και Γ_2 αντίστοιχα. Δηλαδή,

$$\Gamma_1 \vdash \Psi_1 \quad \text{και} \quad \Gamma_2 \vdash \Psi_2$$

Βήμα 2 : Εξασθενίζουμε την υπόθεση σε $\Gamma_1 \cup \Gamma_2$. Η απόδειξη των ιδιοτήτων του πρωτοκόλλου διατηρείται υπό ένα μεγαλύτερο σύνολο υποθέσεων.

$$\Gamma_1 \cup \Gamma_2 \vdash \Psi_1 \quad \text{και} \quad \Gamma_1 \cup \Gamma_2 \vdash \Psi_2$$

Βήμα 3 : Εάν η postcondition της modal φόρμουλας Ψ_1 ταιριάζει στην precondition της Ψ_2' τότε αυτές οι δύο μπορούν να συντεθούν sequentially εφαρμόζοντας τον sequencing rule S1. Εδώ η Ψ_2' έχει προκύψει από την Ψ_2 μετά από μία αντικατάσταση των ελεύθερων μεταβλητών όπως αυτή έχει καθοριστεί από τη sequential composition των δύο αντίστοιχων cords. Αυτό διατηρεί τις φόρμουλες που έχουν αποδειχθεί σε προηγούμενα βήματα αφού αυτές οι φόρμουλες είναι true κάτω από όλες τις δυνατές αντιαταστάσεις των ελεύθερων μεταβλητών. Υποθέτοντας ότι η Ψ_1 και η Ψ_2' είναι αντίστοιχα οι $\theta[P_1]_X\phi$ και $\phi[P_2]_X\psi$ έχουμε

$$\Gamma_1 \cup \Gamma_2' \vdash \theta\,[P_1P_2]_X\,\psi$$

Βήμα 5 : Αποδεικνύουμε ότι οι invariants που έχουν χρησιμοποιηθεί για να αποδείξουμε τις ιδιότητες των πρωτοκόλλων , το $\Gamma_1 \cup \Gamma_2$ ισχύει για το Q χρησιμοποιώντας το θεώρημα 4.3 . Από αυτό και από το βήμα 3 συμπεράινουμε ότι

$$\vdash_Q \theta[P_1P_2]_X\,\psi$$

4.4.Αποδεικτική μέθοδος abstraction-refinement

Εκτός από τη σύνθεση, μία άλλη μέθοδος για να αποκτήσουμε επιθυμητές ιδιότητες σε πρωτόκολλα είναι τα refinements. Ένα refinement επενεργεί στα περιεχόμενα μηνύματα ενός μόνο πρωτοκόλλου. Για παράδειγμα η αντικατάσταση ενός plaintext nonce απο ένα encrypted nonce είναι ένα refinement. Πρέπει να σημειώσουμε ότι ένα refinement δεν αλλάζει ούτε τον αριθμό των μηνυμάτων ούτε τη βασική δομή του πρωτοκόλλου στο οποίο λαμβάνει χώρα.

Επειδή τα refinements μπορεί να είναι η αντικατάσταση μίας έκφρασης ή μίας συνάρτησης από μία άλλη, παραδείγματος χάριν η αντικτάσταση μίας symmetric encryption function από μία keyed hash function, η πρόταση για να είναι εφικτό το reasoning σε αυτές τις περιπτώσεις είναι να κάνουμε reasoning για μία πιο αφηρημένη μορφή πρωτοκόλλων, τα protocol templates.

Ένα protocol template είναι ένα πρωτόκολλο που χρησιμοποιεί μεταβλητές συναρτήσεων (function variables). [2]

Στο παράδειγμα που αναφέρθηκε προηγουμένως θα μπορούσαμε να αντικαταστήσουμε την encryption function από μία μεταβλητή συνάρτησης (function variable) κι έτσι η απόδειξη του πρωτοκόλλου θα μπορούσε να χρησιμοποιηθεί για να παραχθεί μία απόδειξη για το protocol template (που περιέχει function variables). Από τη στιγμή, λοιπόν, που θα έχουμε την απόδειξη για το protocol template μπορούμε να αντικαταστήσουμε τη μεταβλητή συνάρτησης από την keyed hash function. Εάν η keyed hash function έχει τις ιδιότητες της symmetric encryption που χρησιμοποιήσαμε στην αρχική απόδειξη, μπορούμε να χρησιμοποιησουμε αποδείξεις αυτών των ιδιοτήτων της keyed hash στη θέση των υποθέσεων για τη function variable. Θα πρέπει

να προσέξουμε σε τέτοιες πριπτώσεις η ιδιότητα του πρωτοκόλλου που θέλουμε να αποδείξουμε να διατηρείται εάν αντικαταστήσουμε μία συνάρτηση από μία άλλη.

Έτσι λοιπόν έχουμε cords τα οποία περιέχουν μεταβλητές συναρτήσεων, δηλαδή μη ορισμένες συναρτήσεις έτσι ώστε εάν στη θέση τους τοποθετηθεί ένας συνδυασμός από ορισμένες συναρτήσεις να καθοριστεί ένα σύνολο από πιθανά runs. Προς αυτή την κατεύθυνση επεκτείνουμε το σντακτικό με επιπλέον ονόματα σναρτήσεων.

Παρακάτω δίνεται το παράδειγμα ενός template challenge-response πρωτοκόλλου:

$$A \rightarrow B : m$$

$$B \rightarrow A : n , F(B, A, n, m)$$

$$A \rightarrow B : G(A, B, m, n)$$

Όπου m και n έιναι fresh nonces και F και G είναι function variables. Αν αντικαταστήσουμε τις F και G με συγκεκριμένες κρυπτογραφικές συναρτήσεις προκύπτουν και διαφορετικά πρωτόκολλα. Για παράδειγμα, αν βάλουμε signatures τότε παίρνουμε το signature-based challenge-response πρωτόκολλο της οικογένειας ISO-9798-3, ενώ αν βάλουμε keyed-hash functions παίρνουμε το πρωτόκολλο SKID3.

Παρακάτω, όπου Q είναι ένα πρωτόκολλο που περιέχει function variables, P είναι ένας ρόλος ή ένα αρχικό τμήμα ενός ρόλου του πρωτοκόλλου και Γ το σύνολο των θεωρούμενων ιδιοτήτων και invariants (οι φόρμουλες του Γ μπορεί να περιέχουν function variables).

$$Q , \Gamma \vdash \phi_1[P]_A\phi_2$$

Αυτό σημαίνει ότι για κάθε αντικατάσταση που εξαλοίφει όλες τις μεταβλητές συναρτήσεων, κάθε εκτέλεση του πρωτοκόλλου Q’ που προκύπτει η οποία σέβεται τις invariants Γ′ που προκύπτουν ικανοποιεί τη φόρμουλα που προκύπτει $\phi_1'[P']_A\phi_2'$.

Soundness theorem : Η Protocol Composition Logic είναι sound για πρωτόκολλα και ισχυρισμούς που περιέχουν μεταβλητές συναρτήσεων. Επιπλέον, η αντικατάσταση διατηρεί τη σημασιολογική συνεπαγωγή και την αγκυρότητα των φορμουλών. [2,4]

Χαρακτηρίζοντας έννοιες πρωτοκόλλων :

Ο formal χαρακτηρισμός εννοιών που αφορούν τον σχεδιασμό των πρωτοκόλλων διευκολύνονται με τη βοήθεια των protocol templates.

Όπως και στη περίπτωση του composition, έτσι κι εδώ έχουμε μία μεθοδολογία δύο βημάτων για formal proofs.

Βήμα 1 : Θεωρώντας ιδιότητες των function variables και μερικές invariants αποδεικνύουμε ιδιότητες των protocol templates. Δηλαδή,

$$Q , \Gamma \vdash \phi_1[P]_A\phi_2$$

Όπου Q είναι ένα αφηρημένο πρωτόκολλο και P είναι ένα πρόγραμμα για ένα ρόλο του πρωτοκόλλου. Το Γ είναι το σύνολο των ιδιοτήτων που έχουμε υποθέσει και των invariants.

Βήμα 2 : Δημιουργούμε στιγμιότυπο των function variables από κρυπογραφικές συναρτήσεις και αποδεικνύουμε ότι οι ιδιότητες που είχαμε υποθέσει καθώς και οι invariants ικανοποιούνται από το πρωτόκολλο που προέκυψε. Έτσι, συμπεραίνουμε ότι αυτό το πρωτόκολλο έχει τη security ιδιότητα που έχει χαρακτηρισθεί από το protocol template.

$$\text{Εάν } Q' \vdash \Gamma' \text{ τότε } Q' \vdash \phi_1'[P']_A\phi 2'$$

Εδώ, οι πρωταρχικές εκδοχές του πρωτοκόλλου, των υποθέσεων κτλ έχουν προκύψει εφαρμόζοντας την υποκατάσταση σ που έχει χρησιμοποιηθεί στη δημιουργία στιγμιοτύπου.

Συνδυάζοντας protocol templates :

Εάν ένα συγκεκριμένο πρωτόκολλο είναι στιγμιότυπο δύο διαφορετικών protocol templates και σέβεται τις invariants που συσχετίζονται με τα αντίστοιχα templates τότε το πρωτόκολλο θα έχει τις security ιδιότητες και των δύο templates. [2]

Ακολουθεί η μεθοδολογία τριών βημάτων :

Βήμα 1 : Προσδιορίζουμε δύο protocol templates οι οποίες εγγυώνται μερικές security ιδιότητες κάτω από κάποιες υποθέσεις.

$$Q_1 , \Gamma_1 \ |- \ \phi_{11}[P_1]_A\phi_{21} \ \text{και} \ Q_2 , \Gamma_2 \ |- \ \phi_{12}[P_2]_A\phi_{22}$$

Εδώ, τα Q_1 και Q_2 είναι protocol templates, τα P_1 και P_2 είναι τα προγράμματα που αντιστοιχούν σε έναν συγκεκριμένο ρόλο και τα Γ_1 και Γ_2 είναι τα σύνολα των ιδιοτήτων που έχουμε υποθέσει ότι ισχύουν και των invariants.

Βήμα 2 : Βρίσκουμε υποκαταστάσεις σ_1 και σ_2 τέτοιες ώστε τα δύο στιγμιότυπα των πρωτοκόλλων και οι ρόλοι να ταυτίζονται, δηλ

$$\sigma_1Q_1 = \sigma_2Q_2 = Q' \ \text{και} \ \sigma_1P_1 = \sigma_2P_2 = P'$$

Βήμα 3 : Αποδεικνύουμε ότι τα στιγμιότυπο πρωτοκόλλου (instantiated protocol) ικανοποιούν τις υποθέσεις και των δύο protocol templates. Έτσι, συμπεραίνουμε ότι κληρονομεί τις securty ιδιότητες και των δύο.

Εάν Q' |- $\Gamma_1' \cup \Gamma_2'$ τότε Q' |- $(\phi_{11}' \wedge \phi_{12}')[P']_A(\phi_{21}' \wedge \phi_{22}')$

Εδώ, οι πρωταρχικές εκδοχές του πρωτοκόλλου, των υποθέσεων κτλ έχουν προκύψει εφαρμόζοντας τις υποκαταστάσεις σ_1 και σ_2 που έχουν χρησιμοποιηθεί στη δημιουργία στιγμιοτύπων.

Deductions με τον $\hat{A}$ να εκτελεί το ρόλο $Init_{CR}$: [2]

AA1, T1, P1 $[Init_{CR}]_A$ <->Receive(A, msg) ^ Contains(msg, F($\hat{B}$, $\hat{A}$,n , m)

CP3, (1) $[Init_{CR}]_A$ $\exists X \exists msg'.$(Computes(X, F($\hat{B}$, $\hat{A}$,n , m)) ^ <->Send(X, msg') ^

Contains(msg', F($\hat{B}$, $\hat{A}$,n , m)))

Γ_{CR} Computes(X, F($\hat{B}$, $\hat{A}$,n , m)) $\supset$ ($\exists A'.X = A'$) $\vee$ ($\exists B'.X = B'$)

HON Honest($\hat{Y}$) ^ <->Send(Y, msg') $\supset \exists X \exists x \exists y.$(<->Fresh(Y,y) ^

(msg'={$\hat{Y},\hat{X},y$} $\vee$ msg'={$\hat{Y},\hat{X},y$, F($\hat{Y},\hat{X},y,x$)} $\vee$

msg'={$\hat{Y},\hat{X}$, G($\hat{Y},\hat{X},y,x$)}

Γ_{CR} $\neg$Contains({$\hat{A},\hat{X},m'$}, F($\hat{B},\hat{A},n,m$))

Γ_{CR} Contains ({$\hat{A},\hat{X}$,G($\hat{A},\hat{X}$,m',x)}, F($\hat{B},\hat{A},n,m$)) $\supset \hat{A} = \hat{B}$

Γ_{CR} Contains ({$\hat{A},\hat{X}$,m',F($\hat{A},\hat{X}$,m',x)}, F($\hat{B},\hat{A},n,m$)) $\supset \hat{A}=\hat{B}$

(4-7) Honest($\hat{A}$) $\wedge$ <->Send(A',msg') $\wedge$ Contains(msg', F($\hat{B},\hat{A},n,m$)) $\supset \hat{A}=\hat{B}$

(2-3),(8) $[Init_{CR}]_A$ Honest($\hat{A}$) $\supset \exists B \exists$ msg'.(Computes(B, F($\hat{B}$, $\hat{A}$,n , m)) $\wedge$

<->Send(B, msg') $\wedge$ Contains(msg', F($\hat{B},\hat{A},n,m$))

Γ_{CR} $\neg$Contains({$\hat{B},\hat{X},n'$ }, F($\hat{B},\hat{A},n,m$)

Γ_{CR} Contains ({$\hat{B},\hat{X}$,G($\hat{B},\hat{X}$,n',x)}, F($\hat{B},\hat{A},n,m$)) $\supset$ m=n'

Γ_{CR} Contains ({$\hat{B},\hat{X}$,n',F($\hat{B},\hat{X}$,n',x)}, F($\hat{B},\hat{A},n,m$)) $\supset \hat{A}=\hat{B} \wedge$ n=n' $\wedge$ x=m

(4),(9-12) $[Init_{CR}]_A$ Honest($\hat{A}$) $\wedge$ Honest($\hat{B}$) $\supset$

$\exists B.$ (<->Send(B, ({$\hat{B},\hat{X}$,G($\hat{B},\hat{X}$,n',x)} $\wedge$ <->Fresh(B,n') $\wedge$ n'=m) $\vee$

(<->Send(B, ({$\hat{B},\hat{A}$,n,F($\hat{B},\hat{A}$,n,m)}))

AN3,P1 $[Init_{CR}]_A$<->Fresh(A,m)

(13-14) $[Init_{CR}]_A$ Honest($\hat{A}$) $\wedge$ Honest($\hat{B}$) $\supset$ (<->Send(B, ({$\hat{B},\hat{A}$,n,F($\hat{B},\hat{A}$,n,m)})

(15),HON $[Init_{CR}]_A$ Honest($\hat{A}$) $\wedge$ Honest($\hat{B}$) $\supset$ ActionsInOrder(

Receive(B,{ $\hat{A}$, $\hat{B}$,n}),Send(B,{ $\hat{A}$, $\hat{B}$,n, F($\hat{B},\hat{A}$,n,m)}))

F,AF3 $[Init_{CR}]_A After(Send(A, \{\hat{A}, \hat{B}, n\}), Receive(B, \{\hat{A}, \hat{B}, n\}))$

F,AF3,HON $[Init_{CR}]_A\ Honest(\hat{B}) \supset After(Send(B, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}), Receive(A, \{\hat{A}, \hat{B}, n, F(\hat{B}, \hat{A}, n, m)\}))$

AF1,AF2 $[Init_{CR}]_A\ Honest(\hat{A}) \wedge Honest(\hat{B}) \supset \phi_{auth}$

<u>Τα αξιώματα Computes : [2]</u>

CP1 $Computes(X,t) \supset Has(X,t)$

CP2 $Has(X,t) \supset (Computes(X,t) \vee \exists m.(<->Receive(X,m) \wedge Contains(m,t)))$

CP3 $(<->Receive(X,m) \wedge Contains(m,t)) \supset \exists Y.\exists m'.(Computes(Y,t) \wedge <->Send(Y,m') \wedge Contains(m',t))$

$Computes(X,t) \equiv (t=g^{\alpha b} \wedge Computes_{DH}(X,g^{\alpha b})) \vee (t=H(\alpha) \wedge Computes_{HASH}(X,H(\alpha))) \vee (t=E_{\alpha}(b) \wedge Computes_{ENC}(X,E_{\alpha}(b)))$

$Computes_{DH}(X,g^{\alpha b}) \equiv ((Has(X,\alpha) \wedge Has(X,g^{b})) \vee (Has(X,b) \wedge Has(X,g^{\alpha})))$

$Computes_{ENC}(X,E_{\alpha}(b)) \equiv Has(X,\alpha) \wedge Has(X,b)$

$Computes_{HASH}(X,H(\alpha)) \equiv Has(X,\alpha)$

ΚΕΦΑΛΑΙΟ 5

Secure Remote Password Protocol (SRP)

5.1 Εισαγωγή

Στο κεφάλαιο αυτό θα μελετήσουμε διεξοδικά τη δομή και τη λειτουργία του πρωτοκόλλου Secure Remote Password (SRP). Πρόκειται για ένα πρωτόκολλο τύπου password-authenticated key agreement. Αυτός ο τύπος πρωτοκόλλων θα εξηγηθεί παρακάτω. Επίσης , θα δούμε στοιχεία που οδήγησαν το SRP στο να εξελιχθεί στην τρέχουσα μορφή του, το κρυπτογραφικό του υπόβαθρο, την αντίστασή του σε πιθανές επιθέσεις καθώς και επιχειρήματα που στοιχειοθετούν την ασφάλεια του.

5.2 Authentication βασισμένο σε password (password-based authentication)

'Ολες οι μέθοδοι authentication εμπίπτουν σε τρεις κατηγορίες :

- Κάτι που ο χρήστης (user) *είναι* (π.χ. voiceprint identification, retinal scanners κτλ)
- Κάτι που ο χρήστης (user) *έχει* (π.χ. ID cards, smartcards κτλ)
- Κάτι που ο χρήστης (user) *ξέρει* (π.χ. passwords, pins κτλ)

Κάθε password authentication protocol έγκειται στο ότι η μία πλευρά προσπαθεί να αποδείξει στην άλλη ότι γνωρίζει κάποιο -συνήθως προκαθορισμένο- password.

Το *password* ως μέσο για την επίτευξη authentication έγινε πολύ δημοφιλές σε υπολογιστικά συστήματα απομακρυσμένης πρόσβασης λόγω της ικανότητας του ανθρώπινου εγκεφάλου να το απομνημονεύει εύκολα χάρη στο μικρό –συνήθως- μέγεθός του.

Εάν ένας χρήστης (user) U θελήσει να χρησιμοποιήσει μία υπηρεσία ενός host H θα πρέπει πρώτα να αρχικοποιηθεί από τον H και να εκδόσει ένα password. Ο H διατηρεί ένα αρχείο με τα passwords όλων των χρηστών, με κάθε εισαγωγή να είναι ένα ζεύγος (ID_U , P_U) όπου ID_U είναι η ταυτότητα του U και P_U το password του U.

Ακολουθεί το διάγραμμα ενός απλού password- based πρωτοκόλλου κατά το οποίο ο U προσπαθεί να αποκτήσει πρόσβαση στο H :

1. U → H : ID_U ;
2. H → U : "Password" ;
3. U → H : P_U ;
4. Ο H βρίσκει την είσοδο (ID_U , P_U) στο αρχείο του ;

Η πρόσβαση επιτρέπεται στον U εάν το P_U που δέχθηκε ταιριάζει με αυτό του αρχείου.

Το παραπάνω πρωτόκολλο που δεν εκτελεί καμία κρυπτογραφική λειτουργία δεν μπορεί να χρησιμοποιηθει στη πράξη, καθότι παρουσιάζει δύο βασικές αδυναμίες.

Η πρώτη έχει να κάνει με το ότι το αρχείο που περιέχει τα (ID_U , P_U) φυλάσσεται από τον Host σε μη κρυπτογραφημένη μορφή. Με αυτό τον τρόπο θα μπορούσε κάποιος ο οποίος κατάφερε να αποκτήσει πρόσβαση σε αυτό το αρχείο (για παράδειγμα ακόμη και ο system administrator) να χρησιμοποιήσει όλα τα δικαιώματα των χρηστών σύμφωνα με την επιθυμία του προσποιούμενος οποιονδήποτε χρήστη από τους εγγεγραμένους στο αρχείο.

Η δεύτερη αδυναμία του παραπάνω πρωτοκόλλου είναι ότι ο U στέλνει στον H το password μη κρυπτογραφημένο και θα μπορούσε να υποκλαπεί από κάποιον κατά τη διάρκεια της αποστολής του. Αυτή η επίθεση λέγεται *online password eavesdropping.*[16]

Παρακάτω, όπου αναφέρονται οι όροι *client* και *server* θα εννοούμε ό,τι εννούσαμε αντίστοιχα με τους όρους *user* και *host* στην περίπτωση των άμεσων authentication protocols. Παρομοίως, οι όροι *password* και *verifier* αντιστοιχούν στα *ιδιωτικά* και *δημόσια κλειδιά* με τη διαφορά όμως ότι ο verifier δε γίνεται δημόσια γνωστός αλλά διατηρείται μυστικός από τον server.

5.3 Παλαιότερες τεχνικές για authentication σε πρωτόκολλα

Ας υποθέσουμε ότι η Carol είναι ο user/client και ο Steve ο host/server. Όπως και στο πρωτόκολλο που περιγράφηκε στη προηγούμενη παράγραφο, η Carol στέλνει το password της σε μη κρυπτογραφημένη μορφή (plaintext password) στον Steve και αυτός με τη σειρά του το επαληθεύει είτε συγκρίνοντας το απευθείας με τη βάση δεδομένων που διατηρεί, είτε εφαρμόζοντας πρώτα μια hash function και ελέγχοντας μετά τη βάση δεδομένων που περιέχει αποθηκευμένα τα passwords σε καθένα από τα οποία έχει εφαρμοσθεί hash function. [hash function : 14]

Το γεγονός ότι η Carol έστειλε το password της ως plaintext το καθιστά αμέσως ευάλωτο σε eavesdropping attack (όπως αναλύσαμε στην προηγούμενη παράγραφο), επομένως είναι ακατάλληλο για χρήση σε δίκτυα όπου είναι εφικτές τέτοιου είδους επιθέσεις.

Προκειμένου να ξεπεραστεί αυτο το πρόβλημα υιοθετούν ένα πρωτόκολλο τύπου chellenge-response, δηλαδή ένα πρωτόκολλο που έχει την παρακάτω μορφή :

1. Η Carol στέλνει την ταυτότητα της στον Steve μαζί με κάποιο τυχαίο μήνυμα.

2. Ο Steve στέλνει στην Carol ένα τυχαίο μήνυμα που λέγεται *challenge*.

3. Η Carol εκτελεί κάποιους υπολογισμούς που βασίζονται στο challenge, στο πρωτο τυχαίο μήνυμα και στο password της. Στέλνει αυτή την απάντηση (*response*) στον Steve, ο οποίος εκτελεί τους ίδιους υπολογισμούς και επαληθεύει την ορθότητα της response της Carol.

Με το να στέλνεται διαφορετικό challenge από την Carol σε κάθε εκτέλεση του πρωτοκόλλου αποφεύγονται επιθέσεις επανάληψης (replay attacks). Παρόλ΄αυτά είναι εφικτές *dictionairy attacks* (επιθέσεις λεξικού). Για παράδειγμα, θα μπορούσε κάποιος να υποκλέψει το challenge και τη response από μία επιτυχή προσπάθεια authentication και να αρχίσει να μαντεύει passwords μέχρι να βρει εκείνο που δημιουργεί μία response που ταιριάζει με αυτήν που είχε κλέψει.

Προκειμένου, λοιπόν, να αποφευχθούν τέτοιου είδους επιθέσεις οι σχεδιαστές πρωτοκόλλων κινήθηκαν είτε προς την κατεύθυνση του να αυξήσουν το μήκος του κλειδιού είτε προσπάθησαν να βελτιώσουν το φυσικό στρώμα των καναλιών μεταφοράς ώστε να είναι δύσκολη η υποκλοπή μέσα από αυτό.

Αργότερα, το πρωτόκολλο EKE (Encrypted Key Exchange) κατάφερε να αντισταθεί σε dictionary attacks χρησιμοποιώντας ένα συνδυασμό από συμμετρική και δημοσίου κλειδιού κρυπτογραφία. Από το EKE προέκυψαν τα DH-EKE και SPEKE που παρείχαν επιπλέον *forward secrecy*, μία ιδιότητα κατά την οποία ακόμα και αν ένας attacker είχε καταφέρει να αποκήσει το password δε μπορεί να βρει τα session keys από παλαιότερες δράσεις. Όλα τα πρωτόκολλα αυτής της οικογένειας όμως, παροσιάζουν plaintext-equivalence αφού πρέπει και ο client και ο host να έχουν πρόσβαση στο ίδιο μυστικό password ή hash. Υπενθυμίζουμε ότι plaitext-equivalence έχουμε όταν πρέπει ο server να αποθηκεύσει το password του user ή το ιδιωτικό κλειδί.

5.4 Πρωτόκολλο AKE (Asymmetric Key Exchange)

Μέχρι τώρα είδαμε το EKE καθώς και κάποιες παραλλαγές του. Στην παράγραφο αυτή θα μελετήσουμε το πρωτόκολλο AKE (Asymmetric Key Exchange protocol) το οποίο

ουσιαστικά αποτελεί τη γενική μορφή μίας κλάσης πρωτοκόλλων που βασίζονται στη χρήση verifier (verified-based protocols). [20]

Το πρώτο στάδιο κατά την εκτέλεση του AKE είναι η ανταλλαγή κλειδιών μεταξύ των δύο συνιστωσών του πρωτοκόλλου, δηλαδή του client και του server, ώστε στη συνέχεια να χρησιμοποιηθούν για να επαληθευτεί ότι πράγματι οι δύο πλευρές γνωρίζουν τα passwords. Χρησιμοποιεί προκαθορισμένες μαθηματικές σχέσεις για να συνδυάσει τις «εφήμερες τιμές» (ephemeral values) –θα εξηγηθεί παρακάτω- που έχουν ανταλλαγεί με καθορισμένες παραμέτρους passwords.

Το AKE ακολουθεί μία προσέγγιση «ανταλλαγής μυστικού» (swapped secret approach), κατά την οποία κάθε πλευρά υπολογίζει ένα μυστικό και στη συνέχεια του εφαρμόζει μία μονής κατεύθυνσης συνάρτηση για να δημιουργήσει έναν επαληθευτή (verifier) ο οποίος θα σταλεί στην άλλη πλευρά. Παρόλο που είναι σημαντικό να φυλάξουμε τον verifier για να αποτρέψουμε μία επίθεση λεξικού, ο verifier από μόνος του δεν παρέχει αρκετή πληροφορία για να προσδιοριστεί ο χρήστης (user), αλλά χρειάζεται και το μυστικό password.

Μία τέτοια τεχνική είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου ο host είναι ένα σύστημα πολλών χρηστών και αποθηκεύει πολλούς verifiers. Έτσι, κατά τη διάσκεια της εγκατάστασης ή αλλαγής του password ενός χρήστη, δε χρειάζεται να φύγει από τον host το μυστικό του χρήστη αλλά μόνο ο verifier, αυξάνοντας με αυτό τον τρόπο την ασφάλεια του συστήματος.

Στον παρακάτω πίνακα συνοψίζονται οι συμβολισμοί που θα χρησιμοποιηθούν στην περιγραφή του πρωτοκόλλου.

w, x, y, z	τυχαίες παράμετροι
P(x)	μονής κατεύθυνσης συνάρτηση δημιυργίας επαληθευτή
Q(x, y), R(x, y)	συναρτήσεις «ανακατέματος» για ιδιωτικές και δημόσιες παμέτρους
S(x, y)	συνάστηση δημιουργίας session key
K	session key

Πίνακας 5.1 Συμβολισμοί του AKE

Η εξίσωση που ακολουθεί πρέπει να ικανοποιείται ώστε να λειτουργεί σωστά το AKE (οι τύποι των δεδομένων και τα πεδία ορισμού και τιμών για τις συναρτήσεις θα προσδιοριστούν αργότερα) :

$(\forall w, x, y, z)\ S(R(P(w), P(x)), Q(y, z)) = S(R((P(y), P(z)), Q(w, x))$ (1)

Στην παραπάνω εξίσωση οι επιλογές των συναρτήσεων P( ), Q( ), R( ) και S( ) είναι που θα καθορίσουν την ασφάλεια του πρωτοκόλλου. Για παράδειγμα, η P( ) πρέπει να είναι μονής κατεύθυνσης, δηλαδή δεδομένης της P(x) να μην μπορεί να βρεθεί η x.

Το πρωτόκολλο AKE ξεκινάει ως εξής:

Η Carol διαλέγει μία παράμετρο x και ο Steve μία παράμετρο z. Αυτά θα αποτελέσουν τα passwords του πρωτοκόλλου. Στη συνέχεια, η Carol υπολογίζει το P(x) και το στέλνει στον Steve ενώ ο τελευταίος υπολογίζει το P(z) και το στέλνει στην Carol. Τώρα πια και οι δύο είναι έτοιμοι να χρησιμοποιήσουν το AKE για να ανταλλάξουν κλειδιά ακολουθώντας τα παρακάτω βήματα :

Carol		**Steve**
Δημιουργεί ένα τυχαίο w	→P(w)	K=S(R(P(w),P(x)),Q(y,z))
K=S(R(P(y),P(z)),Q(w,x))	←P(y)	Δημιουργεί ένα τυχαίο y

Πίνακας 5.2 AKE

Εάν οι τιμές των x και z που χρησιμοποιήθηκαν για να υπολογιστεί το session key αντιστοιχούν στις προσυμφωνηθείσες τιμές των P(x) και P(z) , τότε, από την εξίσωση (1) οι δύο τιμές των K θα συμπίπτουν. Στο τέλος της διαδικασίας, η Carol και ο Steve μπορούν να χρησιμοποιήσουν κάποια μέθοδο την οποία εχουν συμφωνήσει από πριν για να επαληθέσουν ότι τα κλειδιά τους συμπίπτουν.

Παραπάνω, τα μυστικά x και z είναι μεγάλοι όροι ενώ τα w και y είναι *εφήμερες παράμετροι (ephemeral parameters)* που έχουν δημιουργηθεί από την κάθε πλευρά για να διασφαλιστεί ότι τα session keys είναι διαφορετικά σε κάθε session. [20]

Η P( ) πρέπει αφενός μεν να μη μπορεί να αντιστραφεί, αφετέρου δε το output της να δίνει ελάχιστες έως καθόλου πληροφορίες για το input της. Η S( ) πρέπει να επιλεγεί έτσι ώστε να προστατεύει το δεύτερο όρισμά της από διαρροή. Επίσης, πρέπει να είναι αδύνατο να υπολογιστεί η τιμή του K γνωρίζοντας μόνο τα P(w) , P(x), P(y)και P(z).

5.5 Secure Remote Password Protocol (SRP)

Το πρωτόκολλο Secure Remote Password (SRP) [20] δεν είναι τίποτα άλλο παρά μία πιθανή εφαρμογή του AKE η οποία θεωρείται απλή, γρήγορη και ασφαλής.

Στο SRP όλοι οι υπολογισμοί γίνονται στο GF(n) , επομένως όλα τα inputs και τα outputs των P( ), Q( ), R( ) και S( )ανήκουν στο {0,...,n-1}.

Η συνάρτηση μονής κατεύθυνσης P( ) που δημιουργεί τον verifier είναι η

$$P(x)=g^x \qquad (2)$$

Όπου g είναι ένας γεννήτορας στο GF(n).

Οι συναρτήσεις Q( ), R( ) και S( ) ορίζονται ως εξής :

$$Q(w,x) = w + ux \qquad (3)$$

$$R(w,x) = wx^u \qquad (4)$$

$$S(w,x) = w^x \qquad (5)$$

Όπου το u είναι συνάρτηση των w και x δηλαδή u = f(w,x). Οι εξισώσεις (2)-(5) ικανοποιούν την (1).

Ο παρακάτω πίνακας αξηγεί τους συμβολισμούς που θα χρησιμοποιηθούν στην περιγραφή των βημάτων του SRP.

n	Ένας μεγάλος πρώτος αριθμός.Όοι οι υπολογισμοί γίνονται modulo n.
g	Πρωταρχική ρίζα modulo n (γεννήτορας)
s	Ένα τυχαίο string που χρησιμοποιείται ως salt του χρήστη
P	Το password του χρήστη
x	Ένα ιδιωτικό κλειδί που έχει προκύψει από το password και το salt
υ	Ο password verifier του host
u	Τυχαία παράμετρος κρυπτογράφησης δημόσια γνωστή
α,b	Εφήμερα ιδιωτικά κλειδιά που έχουν δημιουργηθεί τυχαία
A,B	Αντίστοιχα δημόσια κλειδιά
H( )	Hash Function μονής κατεύθυνσης
m, n	Τα δύο strings m και n παραθεμένα (concatenated)
K	Session Key

Πίνακας 5.3 Συμβολισμοί του SRP

Οι τιμές των n και g είναι γνωστές από πριν. *Salt* είναι μία τυχαία ακολουθία από bits και αποτελεί το ένα input της συνάρτησης μονής κατεύθυνσης. Το άλλο input είναι συνήθως ένα password ή ένα passphrase. Το output αποθηκεύεται αντί για το password και χρησιμοποιείται για authentication. Η συνάρτηση μονής κατεύθυνσης είναι συνήθως μία κρυπτογραφική hash function. [18]

Η Carol, προκειμένου να εγκαταστήσει το password P με τον Steve διαλέγει ένα τυχαίο salt s και υπολογίζει τα

$$x = H(s, P)$$

και

$$\upsilon = g^x$$

Ο Steve αποθηκεύει τα υ και s ως τον password verifier και το salt της Carol.

Στο AKE ο Steve έχει ένα password z και ένα δημόσιο κλειδί g^z το οποίο κατέχει η Carol. Στο SRP z=0 επομένως $g^z=1$ κι έτσι ο Steve πρέπει να διατηρήσει τον verifier υ κρυφό για να διασφαλιστεί αμοιβαίο authentication. Με αυτό τον τρόπο η Carol απαλλάσσεται από το να θυμάται το δημόσιο κλειδί του Steve κι έτσι το πρωτόκολλο απλοποιείται.

Το πρωτόκολλο SRP περιγράφεται στον παρακάτω πίνακα και στη συνέχεια αναλύονται τα επιμέρους βήματα.

	Carol		**Steve**
1.		→C	αναζήτηση s,v
2.	x=H(s,P)	←s	
3.	$A=g^{\alpha}$	→A	
4.		←B,u	$B=\upsilon+g^b$
5.	$S=(B-g^x)^{\alpha+ux}$		$S=(A\upsilon^u)^b$
6.	K=H(S)		K=H(s)
7.	$M_1=H(A,B,K)$	→M_1	verify M_1
8.	verify M_2	←M_2	$M_2=H(A,M_1,K)$

Πίνακας 5.4 SRP

Στο πρώτο βήμα η Carol στέλνει στον Steve το username της (π.χ. Carol).

Στο δεύτερο βήμα ο Steve ψάχνει το password entry της Carol και φέρνει το password verifier της και το salt της. Στέλνει το s στην Carol. Αυτή υπολογίζει το x (long term private key) χρησιμοποιώντας το s και το πραγματικό της password P.

Στο τρίτο βήμα η Carol δημιουργεί έναν τυχαίο αριθμό α με 1<α<n , υπολογίζει το εφήμερο δημόσιο κλειδί της $A=g^{\alpha}$ και το στέλνει στον Steve.

Στο τέταρτο βήμα ο Steve δημιουργεί τον δικό του τυχαίο αριθμό b με 1<b<n , υπολογίζει το δημόσιο εφήμερο κλειδί του $B=v+g^{b}$ και το στέλνει στην Carol μαζί με την τυχαία παράμετρο u.

Στο πέμπτο βήμα η Carol και ο Steve υπολογίζουν την κοινή εκθετική τιμή $S=g^{\alpha b+bux}$ χρησιμοποιώντας τις τιμές που έχουν στη διάθεσή τους. Εάν το password P του βήματος δύο της Carol ταιριάζει με αυτό που είχε αρχικά χρησιμοποιήσει για τη δημιουργία του υ τότε και οι δύο τιμές του s θα ταιριάξουν.

Στο έκτο βήμα και οι δύο κάνουν hash στο S και δημιουργούν ένα κρυπτογραφικά ισχυρό session key.

Στο έβδομο βήμα η Carol στέλνει στον Steve το M_1 ως απόδειξη ότι κατέχει το σωστό session key. Ο Steve υπολογίζει το M_1 μόνος του και επαληθεύει ότι ταιριάζει με αυτό που του έστειλε η Carol.

Στο όγδοο βήμα ο Steve στέλνει στην Carol το M_2 ως απόδειξη ότι κατέχει το σωστό session key. Η Carol υπολογίζει το M_2 μόνη της και το επαληθεύει μόνο εάν ταιριάζει με αυτό που της έστειλε ο Steve.

Ακολουθεί ενδεικτικά μία υλοποιήση του SRP σε Python : [17]

```
# An example SRP-6a authentication
# WARNING: Do not use for real cryptographic purposes beyond testing.
# based on http://srp.stanford.edu/design.html
import hashlib
import random

def global_print(*names):
   x = lambda s: ["%s", "0x%x"][isinstance(s, long)] % s
   print "".join("%s = %s\n" % (name, x(globals()[name])) for name in names)

def H(*a):  # a one-way hash function
   return int(hashlib.sha256(str(a)).hexdigest(), 16) % N

def cryptrand(n=1024):
   return random.SystemRandom().getrandbits(n) % N

# A large safe prime (N = 2q+1, where q is prime)
# All arithmetic is done modulo N
# (generated using "openssl dhparam -text 1024")
```

```
N = '''00:c0:37:c3:75:88:b4:32:98:87:e6:1c:2d:a3:32:
     4b:1b:a4:b8:1a:63:f9:74:8f:ed:2d:8a:41:0c:2f:
     c2:1b:12:32:f0:d3:bf:a0:24:27:6c:fd:88:44:81:
     97:aa:e4:86:a6:3b:fc:a7:b8:bf:77:54:df:b3:27:
     c7:20:1f:6f:d1:7f:d7:fd:74:15:8b:d3:1c:e7:72:
     c9:f5:f8:ab:58:45:48:a9:9a:75:9b:5a:2c:05:32:
     16:2b:7b:62:18:e8:f1:42:bc:e2:c3:0d:77:84:68:
     9a:48:3e:09:5e:70:16:18:43:79:13:a8:c3:9c:3d:
     d0:d4:ca:3c:50:0b:88:5f:e3'''
N = int(''.join(N.split()).replace(':', ''), 16)
g = 2       # A generator modulo N

k = H(N, g)  # Multiplier parameter (k=3 in legacy SRP-6)

print "#. H, N, g, and k are known beforehand to both client and server:"
global_print("H", "N", "g", "k")

print "0. server stores (I, s, v) in its password database"

# the server must first generate the password verifier
I = "person"        # Username
p = "password1234"   # Password
s = cryptrand(64)    # Salt for the user
x = H(s, p)         # Private key
v = pow(g, x, N)     # Password verifier
global_print("I", "p", "s", "x", "v")

print "1. client sends username I and public ephemeral value A to the server"
a = cryptrand()
A = pow(g, a, N)
global_print("a", "A")  # client->server (I, A)

print "2. server sends user's salt s and public ephemeral value B to client"
b = cryptrand()
B = (k * v + pow(g, b, N)) % N
global_print("b", "B")  # server->client (s, B)

print "3. client and server calculate the random scrambling parameter"
u = H(A, B)  # Random scrambling parameter
global_print("u")

print "4. client computes session key"
x = H(s, p)
```

```
S_c = pow(B - k * pow(g, x, N), a + u * x, N)
K_c = H(S_c)
global_print("S_c", "K_c")

print "5. server computes session key"
S_s = pow(A * pow(v, u, N), b, N)
K_s = H(S_s)
global_print("S_s", "K_s")

print "6. client sends proof of session key to server"
M_c = H(H(N) ^ H(g), H(I), s, A, B, K_c)
global_print("M_c")
# client->server (M_c) ; server verifies M_c

print "7. server sends proof of session key to client"
M_s = H(A, M_c, K_s)
global_print("M_s")
# server->client (M_s) ;  client verifies M_s
```

Πίνακας 5.5 Υλοποίηση του SRP σε Python

5.6 Υπολογισμός του B και ο ρόλος του u

Στο βήμα 4 του πρωτοκόλλου SRP που περιγράφηκε στην προηγούμενη παράγραφο παρατηρούμε ότι στον υπολογισμό του B υπεισέρχεται και μία ποσότητα u την οποία είχε στείλει προηγουμένως η Carol στον Steve και περιέχει το salt, του οποίου η έννοια εξηγήθηκε παραπάνω.

Ο λόγος για τον οποίο το B δεν ισούται απλά με g^x αλλά με u+ g^x είναι ότι θέλουμε να αποφευχθεί μία επίθεση λεξικού, κατά την οποία θα μπορούσε κάποιος να προσποιηθεί ότι είναι ο host και να πείσει την Carol να κάνει προσπάθεια για authentication.

Αναλυτικότερα [20]:

1. Η Carol στέλνει στη Sue (attacker) το username της.
2. Η Sue στέλνει στην Carol το salt s που είχε υποκλέψει προηγουμένως.
3. Η Carol στέλνει στη Sue το A.
4. Η Sue διαλέγει τα δικά της τυχαία b και u, υπολογίζει το B και στέλνει στην Carol τα B και u.
5. Η Carol υπολογίζει το session key $S=B^{\alpha+ux}$, στη συνέχεια το K από το S και στέλνει το M_1 στη Sue για να της αποδείξει ότι έχει το σωστό session key.

6. Η Sue προσομοιώνει αποτυχία δικτύου ή υποδεικνύει στην Carol ότι το password δεν ήταν σωστό.

Με αυτό τον τρόπο η Sue ξέρει το A και το b και επίσης έχει την απόδειξη του K από την Carol. Μπορεί τώρα να μαντέψει ένα password p' , να υπολογίσει το x' και στη συνέχεια το υ' , να κατασκευάσει το S' [$S'=(Av'^{u})^{b}$] και το K'=H(S') και τελικά να το ελέγξει σε σχέση με το πραγματικό K που της έχει στείλει η Carol. Εάν ταιριάζουν τότε το password που μάντεψε στην αρχή είναι το σωστό.

Προκειμένου, λοιπόν, να αποφευχθεί μία τέτοιου τύπου επίθεση είναι απαραίτητο να χρησιμοποιηθεί από τον host μία τιμή για το υ όπως περιγράφηκε στο βήμα 4. Ωστόσο, η επιλογή του τρόπου με τον οποίο θα συνδυαστεί το g^b με το υ είναι κρίσιμη για την ασφάλεια του πρωτοκόλλου. Εάν υποθέσουμε ότι η μεταξύ τους σχέση εκφράζεται με μία συνάρτηση $B=f(υ,g^b)$ τότε πρέπει να προσέξουμε ώστε να μη χρησιμοποιούμε συναρτήσεις f οι οποίες έχουν την ιδιότητα $f(g^x,g^y)=g^{h(x,y)}$ όπου h() μία συνάρτηση που μπορεί να υπολογιστεί εύκολα, όπως για παράδειγμα η f(x,y)=xy .

Επιπλέον, στόχος είναι από την τιμή του B να απορρέουν όσο το δυνατόν λιγότερες πληροφορίες σχετικά με το υ . Έτσι, για παράδειγμα δε μπορούμε να χρησιμοποιήσουμε συναρτήσεις όπως η $f(x,y)=x \oplus y$ ή $f(x,y)=E_K()$ με το E_K να είναι ένας αλγόριθμος συμμετρικής κρυπτογράφησης γιατί ελλωχεύει ο κίνδυνος μίας partition attack.

Partition attack είναι μία επίθεση η οποία δουλεύει διαιρώντας τα σύνολα των δυνατών plaintexts και ciphertexts σε ικανοποιητικά υπολογίσιμες διαμερίσεις έτσι ώστε η κατανομή των ciphertexts να είναι σημαντικά όχι ομοιόμορφη όταν τα plaintexts έχουν επιλεγεί ομοιόμορφα από ένα δεδομένο block της διαμέρισης.

Στην περίπτωση του SRP, εάν χρησιμποιούσαμε για το B τη συνάρτηση $B=υ \oplus g^b$ ένας εισβολέας θα μπορούσε να αποκτήσει το B και να μαντέψει έναν verifier υ' για κάθε pasword που έχει υποθέσει (guessed password). Εάν το $B \oplus υ' > n$ τότε το συγκεκριμένο password απορρίπτεται ως αδύνατο. Εάν μία τέτοια διαδικασία επαναληφθεί για έναν αριθμό από sessions ο attacker θα είναι σε θέση να μειώσει σημαντικά τον αριθμό των πιθανών passwords και να καταφέρει να εντοπίσει το σωστό password εξαντλητικά (brute force guessing).

Η σχέση μεταξύ των g^b και υ η οποία είναι η πιο απλή και ταυτόχρονα ασφαλής είναι η πρόσθεση modulo, όπου το g είναι πρωταρχική ρίζα του GF(n) προκειμένου όλες οι τιμές του B να είναι ισοπίθανες για κάθε υ. Από αυτή τη σχέση δεν απορρέι πληροφορία για το υ και επιπλέον δεν επιτρέπει dictionary και partition attacks.

Μία άλλη παράμετρος η οποία παίζει σπουδαίο ρόλο στο SRP είναι το u. Ας υποθέσουμε ότι ένας εισβολέας, έστω ο Chris, ο οποίος έχει καταφέρει να υποκλέψει το υ προσποιείται τον client προσπαθώντας να αποκτήσει πρόσβαση στον host. Έστω, επίσης, ότι ο Chris έχει ανακαλύψει με κάποιο τρόπο την τιμή του u. Τότε είναι σε θέση να αποκτήσει πρόσβαση στον host ακολουθώντας τα παρακάτω βήματα :

1. Ο Chris στέλνει το username της Carol στον Steve.
2. Ο Steve στέλνει το salt της Carol s στον Chris.
3. Ο Chris υπολογίζει το
$$A = g^{\alpha}\upsilon^{-u}$$
και το στέλνει στον Steve αντί να χρησιμοποιήσει τον κανονικό τύπο για το A.
4. Ο Steve στέλνει στον Chris το $B=\upsilon+g^{b}$
5. Ο Chris υπολογίζει το session key K :
$$K=H((B-\upsilon)^{\alpha} \bmod n)$$
6. Ο Chris στέλνει στον Steve μία απόδειξη του K και καταφέρνει να κάνει login ως "Carol".

Η επίθεση αυτή έχει αποτέλεσμα επειδή ο Steve υπολογίζει το session key του ως :

$$S=(A\upsilon^{u})^{b}=(g^{\alpha}\upsilon^{-u}\upsilon^{u})^{b}=g^{\alpha b}$$

Πρέπει να σημειωθεί ότι αυτή η τιμή είναι ανεξάρτητη των μεγάλου μήκους κλειδιών και μπορεί εύκολα να υπολογιστεί από τον Chris. Ο τελευταίος, αφού έχει το ίδιο session key με τον Steve μπορεί να τον κάνει να πιστέψει ότι είναι η Carol. Μία τέτοιου είδους επίθεση θα μπορούσε να γίνει και αν το u ήταν μία τιμή δημόσια γνωστή.

Για να αποτραπεί ο Chris από το να μπορεί να απαλείψει το υ με τον τρόπο που είδαμε προηγουμένως, θα πρέπει ο Steve να μην αποκαλύψει την τιμή του u πριν λάβει το A από τον user. Για παράδειγμα, και οι δύο πλευρές μπορούν αν υπολογίσουν το u ως μία απλή συνάρτηση του B και ο Steve να περιμένει την Carol να του στείλει πρώτα το A και μετά αυτός να στείλει το B και να αποκαλύψει το u. Για τους ίδιους λόγους θα πρέπει να αποφευχθεί η ανάθεση της τιμής 0 στο u.

5.7 Ανάλυση της ασφάλειας του SRP

Η ανάλυση ασφαλείας του SRP έγκειται στο ότι πρέπει να αποδείξουμε ότι οποιοσδήποτε εισβολέας, δηλαδή κάποιος τρίτος που παρεισφρύει στην επικοινωνία μταξύ του Steve και της Carol, πρέπει να μην καταφέρει να αποκτήσει πρόσβαση στον host . Πιο συγκεκριμένα, θα πρέπει :

1. Καμία χρήσιμη πληροφορία συσχετιζόμενη με το password P ή το ιδιωτικό κλειδί x να αποκαλυφθεί κατά τη διάρκεια ενός επιτυχους τρεξίματος του πρωτοκόλλου. Πρέπει να αποτραπεί κάθε εισβολέας από το να μαντέψει και να επαληθεύσει passwords που βασίζονται σε κλειδιά που έχουν ανταλλαχθεί.

2. Δεν πρέπει να αποκαλυφθεί καμία χρήσιμη πληροφορία για το session key K σε κάποιον υποκλοπέα. Υποθέτουμε ότι αφού το K είναι ένα κρυπτογραφικά ισχυρό κλειδί δεν ασχολούμαστε με επιθέσεις κατά τις οποίες ο intruder θα μπορούσε απλά να το υποθέσει.

3. Ακόμη και αν ο εισβολέας μπορεί να αλλάξει τα μηνύματα ή να δημιουργήσει δικά του και να τα κάνει να φαίνονται ότι είναι του Steve ή της Carol, το πρωτόκολλο πρέπει να μην του επιτρέπει να αποκτήσει πρόσβαση στον host ή να μάθει οποιαδήποτε πληροφορία που να αφορά passwords ή session keys. Για την ακρίβεια, ο intruder θα πρέπει να μπορεί αν προκαλεί μόνο αποτυχία στο authentication μεταξύ των δύο πλευρών.

4. Ακόμη και αν υποπέσει στον intruder το αρχείο με τα passwords και καταφέρει να μάθει το υ, θα πρέπει να είναι πολύ δαπανηρό γι' αυτόν να εκτελέσει μία dictionary search.

5. Σε περίπτωση που διαρρεύσει το session key κάποιας προηγούμενης session να μην παρέχει καμία χρήσιμη πληροφορία που θα βοηθήσει τον intruder να συμπεράνει το password του χρήστη.

6. Εάν διαρρεύσει το password του χρήστη να μην είναι αρκετό για να προσδιορίσει ο intruder το session key K προηγούμων sessions, με αποτέλεσμα να μη μπορεί να τις αποκρυπτογραφήσει. Τέλος, οι τρέχουσες sessions θα πρέπει να είναι προστατευμένες από passive eavesdropping.

Με αυτές τις προϋποθέσεις [20] ένα πρωτόκολλο χαρακτηρίζεται ως ισχυρό (robust), δηλαδή αντιστέκεται στον κίνδυνο ακόμη και αν οι συμμετέχοντες σε αυτό δεν είναι απολύτως αξιόπιστοι ή ασφαλείς.

Για παράδειγμα, εάν ένας attacker καταφέρει να αποκτήσει το password ενός χρήστη, η ισχύς του να προκαλέσει ζημιά θα πρέπει να πεύσει να υφίσταται αμέσως μόλις ο χρήστης αλλάξει password.

Ως προς την ασφάλειά του , το SRP ανάγεται στο πρόβλημα Diffie-Hellman (DH), το οποίο θεωρείται ότι είναι υπολογιστικά αδύνατο να επιλυθεί όταν έχει αρκετά μεγάλες παραμέτρους. Πράγματι, μπορεί να κατασκευαστεί μία απόδειξη που να συνδέει τη δυσκολία του DH με αυτή του SRP.

Υποθέτουμε ότι υπάρχει αλγόριθμος ο οποίος παράγει το session key του SRP σε πολυωνυμικό χρόνο με δεδομένες όλες τις πληροφορίες που είναι δημόσια γνωστές ή έχουν μεταδοθεί κατά το τρέξιμο μίας νόμιμης και επιτυχημένης διεξαγωγής του πρωτοκόλλου καθώς επίσης και το password του χρήστη.

Ένας τέτοιος αλγόριθμος μπορεί να αναπαρασταθεί σαν ένα μαντείο (oracle) Q το οποίο δέχεται τις τιμές A, B, u, g, n, και x και υπολογίζει το session key $S=g^{\alpha b+bux}$, δηλαδή

$$Q(g^{\alpha}, g^{b}+g^{x}, u, g, n, x)=g^{\alpha b+bux}$$

Σύμφωνα με το DH είναι δύσκολο να υπολογισθεί το $g^{\alpha b}$ στο GF(n) δεδομένων των g^{α} και g^{b} . Θέτοντας u=2 και x=(n-1)/2 μπορούμε να ορίσουμε το DH μαντείο $\hat{Q}$ σε σχέση με τους όρους του Q ως εξής :

$$\hat{Q}\ (A, B, g, n) = Q\ (\ A, B+g^{(n-1)/2}, 2, g, n, (n-1)/2\ )$$

και αντικαθιστώντας τα A και B με g^{α} και g^{b} αντίστοιχα, παίρνουμε

$$\hat{Q}\ (\ g^{\alpha}, g^{b}, g, n\ ) = g^{\alpha\beta}$$

Επομένως, αν υπήρχε κάποια μέθοδος η οποία να μπορεί να υπολογίσει το session key του SRP με passive attack, η ίδια μέθοδος θα μπορούσε να σπάσει και το DH σε πολυωνυμικά ισοδύναμο χρόνο.

Παρατηρούμε ότι ικανοποιείται το κριτήριο 6 της παραπάνω λίστας ανάλυσης ασφαλείας των πρωτοκόλλων, αφού η ανάκτηση του password δεν επιτρέπει τον υπολογισμό κανενός session key προηγούμενης session. Επίσης, ικανοποιείται και το κριτήριο 2, αφού ένας intruder ο οποίος δεν ξέρει το password x έχει ακόμα λιγότερη πληροφορία για το session key.

Ανακεφαλαιώνοντας, θα λέγαμε ότι είναι υπολογιστικά αδύνατος ο υπολογισμός του session key ακόμα και αν είναι γνωστό το password x του χρήστη και όλες οι δημόσιες πληροφορίες. Ένας intruder ο οποίος υποκλέπτει ένα επιτυχές τρέξιμο του SRP δε μπορεί να μαντέψει το session key, αφού τα M_1 και M_2 χωρίς το K δεν του δίνουν καμία επιπλέον πληροφορία ώστε να μαντέψει το K.

Μία *Denning-Sacco Attack* [20] συμβαίνει όταν ένας intruder αποκτά το session key K από μία υποκλεμμένη session και το χρησιμοποιεί είτε για να μπορέσει να υποδυθεί τον χρήστη είτε για να διεξάγει μία brute-force search εναντίον του password του χρήστη.

Εάν ένας passive eavesdropper, έστω η Eve, μάθει το K δε μπορεί να συνάγει κάποια καινούρια πληροφορία συνδυάζοντάς το με τα M_1 και M_2. Πράγματι, τα M_1 και M_2 μπορούν να υπολογισθούν απευθείας από τα δεδομένα που είναι δημόσια γνωστά και το K. Επειδή η Eve δε μπορεί να κάνει υποθέσεις για το session key K

από τα guessed passwords, η πιο εύκολη μέθοδος που μπορεί να χησμοποιήσει έιναι η brute-force dictionary attack. Έτσι, παρατηρούμε ότι ικανοποιείται και η προυπόθεση 5 της λίστας.

5.8 Υποθέσεις σχετικά με την ασφάλεια και περιορισμοί

Όπως αναφέραμε και σε προηγούμενες παραγράφους, η ασφάλεια του SRP εξαρτάται από την επιλογή της συνάρτησης P() και η καταλληλότητά της έγκειται στο κατά πόσο δύσκολο είναι αν αντιστραφεί. Με άλλα λόγια, οι τιμές των w και y πρέπει να μην απορρέουν απο τη γνώση των P(w) και P(y) αντίστοιχα. Στο SRP έχει επιλεγεί η συνάρτηση g^x ($P(x)=g^x$) με τα g και n να είναι δημόσια γνωστά από πριν.

Στην πρηγούμενη παράγραφο αποδείχθηκε ότι η ασφάλεια του SRP ανάγεται στο DH, ενώ για το τελευταίο είναι γνωστό ότι ανάγεται στο DLP (Discrete Logarithm Problem). Εδώ θα πρέπει να σημειωθεί ότι δεν έχει αποδειχθεί ακόμη η ισοδυναμία DH και DLP.

Προκειμένου να αυξηθεί η δυσκολία του υπολογισμού των διακριτών λογαρίθμων στο GF(n), ο n θα πρέπει να μην είναι *smooth prime number*, δηλαδή να μην αποτελείται εξ ολοκλήρου από μικρούς παράγοντες.

Ορισμένα πρωτόκολλα authentication επιδέχονται *subgroup confinement attack*, όπου ο εισβολέας υποχρεώνει το session key που χρησιμοποιείται από οποιαδήποτε πλευρά να περιοριστεί σε μία μικρή υποομάδα του GF(n). Όμως το SRP, λόγω του τρόπου υπολογισμού των session keys του αντιστέκεται σε αυτού του είδους την επίθεση.

Για να είναι ο n ένας ασφαλής πρώτος αριθμός πρέπει να είναι της μορφής

$$n = 2p + 1$$

όπου ο p είναι επίσης πρώτος αριθμός. Αυτοί οι αριθμοί αντιστέκονται στον υπολογισμό του διακριτού λογαρίθμου και περιέχουν τον ελάχιστο αριθμό υποομάδων, αφου το n-1 περιέχει το μικρότερο δυνατό αριθμό παραγόντων (2). Έτσι λοιπόν, αφού το n είναι ένας ασφαλής πρώτος το πρωτόκολλο δεν είναι ευάλωτο σε subgroup confinement attacks.

Ακολουθούν οι έλεγχοι περιορισμών που πρέπει να εκτελεστούν και από τις δύο πλευρές για να επιβεβαιωθεί η ασφάλεια του SRP : [20]

Client : Το n είναι ένας μεγάλος ασφαλής πρώτος αριθμός

Ο Client πρέπει να βεβαιωθεί ότι το n είναι αρκετά μεγάλο ώστε να αντισταθεί στις επιθέσεις. Χρησιμοποιώντας έναν primality tester, ο client μπορεί επίσης να βεβαιωθεί ότι και ο n και ο (n-1)/2 είναι πρώτοι.

Client : ο g είναι μία πρωταρχική ρίζα του GF(n)

Υποθέτοντας ότι η παραγοντοποίηση του n-1 είναι γνωστή, μπορεί να χρησιμοποιηθεί ένας αλγόριθμος που τεστάρει γεννήτορες για να επαληθεύσει το g. Εάν το n είναι ένας ασφαλής πρώτος αυτό το τεστ είναι πολύ εύκολο και γρήγορο.

Server : $A \neq 0$

Αυτό προστατεύει το session key του server από το να αναγκαστεί να είναι μία γνωστή τιμή, ήτοι 0.

Client : $B \neq 0$

Αυτός ο έλεγχος προστετεύει από μία dictionary attack στο password από κάποιον που υποκρίνεται ότι είναι ο host.

$\alpha, b > \log_g n$

Αυτός ο έλεγχος πρέπει να γίνει ώστε να αποτραπεί ένας attacker από το να πάρει τον αλγεβρικό λογάριθμο του g^{α} για να υπολογίσει το α.

5.9 Επιδόσεις του SRP

Κατά το στάδιο της εφαρμογής ενός πρωτοκόλλου σε πραγματικά συστήματα, πέρα από τα θεωρητικά θέματα που αναλύθηκαν διεξοδκά στις προηγούμενες παραγράφους, μας απασχολούν και ζητήματα που έχουν να κάνουν με τις επιδόσεις του και τα οποία είναι κρίσιμα για το αν το πρωτόκολλο είναι τελικά εφαρμόσιμο στην πράξη ή όχι και αν ναι σε τι βαθμό.

Μερικά από τα κριτήρια που επηρρεάζουν τις επιδόσεις ενός πρωτοκόλλου και κατ' επέκταση είναι καθοριστικά για το σχεδιασμό του είναι ο αριθμός των γύρων του πρωτοκόλλου (message rounds), το μέγεθος των ανταλλασσόμενων μηνυμάτων και –για την περίπτωση του SRP- ο αναμενόμενος χρόνος εκτέλεσης μίας επιτυχούς προσπάθειας authentication.

Επομένως, μία πρώτη προσπάθεια βελτιστοποίησης του SRP μπορεί να γίνει προς την κατεύθυνση της μείωσης του αριθμού των γύρων του πρωτοκόλλου. Το SRP, όπως το έχουμε δει μέχρι τώρα είναι (ως προς τους γύρους) :

C → S	C
C ← S	s
C → S	A
C ← S	B
C → S	M_1
C ← S	M_2

Πίνακας 5.5 Οι γύροι του SRP

Σε μία βελτιωμένη εκδοχή του SRP ο αριθμός των γύρων θα μπορούσε να έχει περιοριστεί, όπως δείχνει και ο παρακάτω πίνακας :

C → S	C, A
C ← S	s, B
C → S	M_1
C ← S	M_2

Πίνακας 5.6 Οι μειωμένοι γύροι του SRP

Είναι δυνατόν να μειωθεί και ο αριθμός των ανταλλασσόμενων μηνυμάτων στην περίπτωση που είναι επιθυμητό μόνο το μονής κατεύθυνσης authentication. Μία τέτοια λύση περιλαμβάνεται στον παρακάτω πίνακα :

C → S	C, A
C ← S	s, B
C → S	M_1

Πίνακας 5.7 SRP με μειωμένους γύρους και μειωμένα ανταλλασσόμενα μηνύματα

Τέλος, αναφορικά με το χρόνο εκτέλεσης του SRP θα πρέπει να σημειωθεί ότι ο πιο αργός υπολογισμός είναι αυτός του εκθετικού υπολοίπου modulo n (modular exponentiation). Σε σχέση με αυτόν, οι άλλοι υπολογισμοί όπως για παράδειγμα οι hash functions, η πρόσθεση και ο πολλαπλασιασμός απαιτούν αμελητέο χρόνο από τον επεξεργαστή.

Στον πίνακα που ακολουθεί παρατίθεται ενδεικτικά ο χρόνος τρεξίματος του SRP συγκριτικά με άλλα πρωτόκολλα της ίδιας οικογένειας (μερικά από τα οποία αναφέρθηκαν και στην αρχή του κεφαλαίου) :

Πρωτόκολλο	**Χρόνος Εκτέλεσης (ms)**
DH-EKE	626
SPEKE	758
A-EKE	1252
B-SPEKE	1384
SRP	873

Πίνακας 5.8 Πρωτόκολλα και οι χρόνοι εκτέλεσής τους

Οι παραπάνω χρόνοι μετρήθηκαν από έναν υπολογιστή 167 MHz μονού επεξεργαστή Sun ULTRASpark-1 που έτρεχε λειτουργικό Solaris 2.5. [20]

ΚΕΦΑΛΑΙΟ 6

Εφαρμογή της PCL στο SRP

6.1 Εισαγωγή

Στα πρώτα τέσσερα κεφάλαια μελετήσαμε διεξοδικά την Protocol Composition Logic. Είδαμε το συντακτικό της καθώς και μεθόδους που χρησιμεύουν σε αποδείξεις ιδιοτήτων των πρωτοκόλλων ασφαλείας. Στο πέμπτο κεφάλαιο είδαμε τη δομή, τη λειτουργία και την κρυπτογραφία του πρωτοκόλλου SRP(Secure Remote Password Protocol). Στο κεφάλαιο που ακολουθεί θα εκφράσουμε το SRP στην PCL μεσω του cord calculus και στη συνέχεια θα παρουσιάσουμε τα αξιώματα και τις υποθέσεις που χρειάζονται για την απόδειξη κάποιων ιδιοτήτων του όπως για παράδειγμα secrecy, authentication κτλ.

6.2 Το SRP στον Cord Calculus

Όπως είδαμε και στο προηγούμενο κεφάλαιο, το SRP ολοκληρώνεται στα παρακάτω 8 βήματα :

	Carol		**Steve**
1.		$\rightarrow C$	αναζήτηση s,v
2.	$x=H(s,P)$	$\leftarrow s$	
3.	$A=g^{\alpha}$	$\rightarrow A$	
4.		$\leftarrow B,u$	$B=\upsilon+g^{b}$
5.	$S=(B-g^{x})^{\alpha+ux}$		$S=(A\upsilon^{u})^{b}$
6.	$K=H(S)$		$K=H(s)$
7.	$M_1=H(A,B,K)$	$\rightarrow M_1$	verify M_1
8.	verify M_2	$\leftarrow M_2$	$M_2=H(A,M_1,K)$

Η Carol στο παραπάνω διάγραμμα αντιπροσωπεύει κάθε πιθανό client, γι' αυτό, σύμφωνα με τον cord calculus θα είναι ένας thread του [Client] και ομοίως ο Steve θα είναι ένας thread του [Host].

Στο πρώτο βήμα η Carol στέλνει το όνομα της στον Steve, κάτι που στον cord calculus συμβολίζεται ως <{|"Carol"|}>. Επομένως, ο Host πρέπει να δεχθεί αυτό που του έστειλε η Carol, δηλαδή πρέπει να δεχθεί ένα μήνυμα του τύπου ({|"name"|}) όπου η μεταβλητή "name" θα πάρει την τιμή "Carol".

Στο δεύτερο βήμα, ο Steve, αφού είχε ψάξει πρηγουμένως τα s και υ που αντιστοιχούν στην Carol, της στέλνει πίσω το s, δηλαδή <{|s|}> και στη συνέχεια αυτή υπολογίζει το x=H(s,P).

Στο τρίτο βήμα η Carol δημιουργεί ένα τυχαίο α , υπολογίζει το $A=g^{\alpha}$ και το στέλνει στον Steve δηλαδή (να)<{|A|}> , ενώ ο Steve πρέπει να λάβει το A, δηλαδή ({|A'|}) όπου η A' είναι μία μεταβλητή της μορφής $A'=g^{\alpha'}$ με το α' να είναι η μεταβλητή που θα πάρει την τιμή α.

Στο τέταρτο βήμα ο Steve δημιουργεί ένα τυχαίο b ,ένα u, υπολογίζει το $B=υ+g^{b}$ και το στέλνει στην Carol, δηλαδή (vb)(vu)<{|B,u|}>. Η Carol πρέπει να δεχθεί το B,u, δηλαδή ({|B',u'|}) όπου B' και u' είναι οι μεταβλητές που θα πάρουν τις τιμές B και u αντίστοιχα.

Στο πέμπτο βήμα η Carol υπολογίζει το $S=(B-g^{x})^{\alpha+ux}$ και ο Steve το $S=(Aυ^{u})^{b}$.

Στο έκτο βήμα και η Carol και ο Steve υπολογίζουν το K=H(S).

Στα δύο προηγούμενα βήματα δεν υπάρχει ανταλλαγή μηνυμάτων μεταξύ των δύο πλευρών.

Στο έβδομο βήμα η Carol υπολογίζει το $M_1=H(A,B,K)$ και το στέλνει στον Steve, δηλαδή <{|M_1|}>. Ο Steve το δέχεται και κάνει verification με το M_1 που υπολογίζει ο ίδιος, δηλαδή ({|M_1'|})({|M_1'|}/{|M_1|} όπου M_1' η μεταβλητή που θα πάρει την τιμή M_1.

Στο όγδοο βήμα ο Steve υπολογίζει το $M_2=H(A,M_1,K)$ και το στέλνει στην Carol, δηλαδή <{|M_2|}>. Η Carol με τη σειρά της το δέχεται και στη συνέχεια κάνει verification δηλαδή ({|M_2'|})({|M_2'|}/{|M_2|}).

Όλα τα βήματα που περιγράφηκαν παραπάνω μπορούν να εκφραστούν μέσω του cord calculus :

[Client] = [<{|"Carol"|}> ({|s'|}) (να)<{|A|}> ({|B',u'|}) <{|M_1|}> ({|M_2'|})({|M_2'|}/{|M_2|})]

[Host] = [({|"name"|}) <{|s|}> ({|A'|}) (νb)(νυ)<{|B,u|}> ({|M_1'|})({|M_1'|}/{|M_1|}) <{|M_2|}>]

όπου τα σύμβολα και οι μεταβλητές εξηγήθηκαν παραπάνω.

6.3 Αξιώματα Computes για το SRP

Στην παράγραφο αυτή θα αναλύσουμε τα αξιώματα Computes για το SRP.

Τα αξιώματα **CP1** , **CP2** και **CP3** ισχύουν και στην περίπτωση του SRP. Συγκεκριμένα :

CP1 Computes(X,t) $\supset$ Has(X,t)

CP2 Has(X,t) $\supset$ (Computes(X,t) $\vee \exists m$.(<->Receive(X,m) $\wedge$ Contains(m,t)))

CP3 (<->Receive(X,m) $\wedge$ Contains(m,t)) $\supset \exists Y . \exists m'$.(Computes(Y,t) $\wedge$ <->Send(Y,m') $\wedge$ Contains(m',t))

Πέρα όμως από τα παραπάνω, από το ίδιο το πρωτόκολλο μπορούν να προκύψουν νέα αξιώματα τα οποία είναι απαραίτητα για την απόδειξη των ιδιοτήτων ασφαλείας του.

1. Αξιώματα που αφορούν τον Client (Carol) :

Computes(C, x)= Has(C,x)= (-)Receive(C,s) $\wedge$ Has(C,P)

Σύμφωνα με το παραπάνω αξίωμα, το x ισούται με το H(s,P) και για να το υπολογίσει ο C πρέπει να έχει το Password P και επίσης να έχει λάβει προηγουμένως το s από τον Steve. Επιπλέον, εάν υπολογίσει το x συνεπάγεται ότι το ξέρει.

Computes(C, α+ux)= New(C, α) $\wedge$ Receive(C,u) $\wedge$ Computes(C,x)

Για να υπολογίσει η Carol το α+ux πρέπει να έχει δημιουργήσει ένα τυχαίο α, να έχει λάβει το u από τον Steve και να έχει υπολογίσει το x.

Computes(C, g^x)= Has(C,x) $\wedge$ Has(C, g)

Για να υπολογίσει το g^x η Carol, θα πρέπει να ξέρει και το g και το x.

Computes(C, S)= Computes (C, $(B\text{-}g^x)^{\alpha+ux}$)= (-)Receive(C,B) ^ Has(C,g^x) ^Computes(C, α+ux)

Το προηγούμενο αξίωμα λέει ότι αφού το S ισούται με $(B\text{-}g^x)^{\alpha+ux}$, για να το υπολογίσει η Carol πρέπει να έχει λάβει το B, να ξέρει το g^x και να έχει υπολογίσει το α+ux.

Computes(C,K)= Computes(C, H(S))= Has(C,S)= <->Computes(C,S)

Αφού το K=H(S) και η H είναι Hash Function θα πρέπει η Carol να ξέρει το S, πράγμα που ισχύει αφού το έχει υπολογίσει πρηγουμένως.

Computes(C, M_1)= Computes(C, H(A,B,K))= Has((C,A) ^ Has(C,B) ^ Has(C,K)

Επειδή το M_1=H(A,B,K) και η H είναι Hash Function θα πρέπει η Carol να ξέρει το A, το B και το K.

2.Αξιώματα που αφορούν τον Host (Steve) :

Computes(H,S)= New(H,u) ^ New(H,b) ^ (-)Receive(H,A) ^ Has(H,υ)

Το αξίωμα αυτό ισχύει αφού το S ισούται με $(Aυ^u)^b$ και ο H έλαβε το A στο προηγούμενο βήμα από την Carol, τα u και υ τα δημιούργησε και το υ το ήξερε από το βήμα 1.

Computes(H,K)= Computes(H, H(S))= Has(H,S)

Ισχύει αφού η K=H(S) έιναι Hash Function και για να την υπολογίσει ο H πρέπει να ξέρει το S.

3.Αξιώματα Verification :

Verification(C, M_2)= (-)Receive(C,M_2) ^ <->Verify(H,M_1) ^ Verify(C,M_2)

Για να γίνει το verification πρέπει πρώτα να έχει λάβει η Carol το M_2 από τον Steve , ο οποίος έχει είδη κάνει verification στο M_1.

Verification(H,M_1) = (-)Receive(H,M_1) ^ Verify(H,M_1)

Για να γίνει το verification πρέπει πρώτα να έχει λάβει ο Steve το M_1 από την Carol και μετά να κάνει verification στο M_1.

Computes(H,M_2)= (-)Verification(H,M_1) ^ Computes(H,H(A,M_1,K))=

= (-)Verification(H,M_1) ^ Has(H,A) ^Has(H,M_1) ^Has(H,K)

Το παραπάνω αξίωμα ισχύει επειδή το M_2 ισούται με H(A,M_1,K) και για να το υπολογίσει ο H θα πρέπει πρώτα να έχει κάνει verification στο M_1 . Τέλος, το ότι υπολογίζει το H(A,M_1,K) σημαίνει ότι ξέρει τα A,M_1 και K.

6.4 Υποθέσεις που αφορούν το SRP

Το template του SRP μπορεί να γραφεί ως :

C → S : C

S → C : s

C → S : L(α)

S → C : u,G(υ,g^b)

S=F_1(B,x,α,u), S=F_2(A,υ,u,b)

K=H(S)

C → S : M_1= H(A,B,K)

S → C : M_2= H(A,M_1,K)

Όπου $L(\alpha)=g^{\alpha}$ και $B=G(\upsilon,g^{b})$.

Με βάση το παραπάνω template για το SRP που συμφωνεί με τη μορφή για τα templates όπως αυτή ορίστηκε στο κεφάλαιο 3, μπορούμε να προχωρήσουμε στην κατασκευή υποθέσεων για το ίδιο πρωτόκολλο.

Υποθέσεις :

$\upsilon_1 \equiv$ Computes(X, $G(\upsilon, g^{b})$) $\supset \exists$ H.X=H

Η παραπάνω υπόθεση έχει την προφανή ερμηνεία.

$\upsilon_2 \equiv$ <->Fresh(H,b) $\wedge$ <->Fresh(H,u) $\supset \neg$ Contains ({u, $G(\upsilon,g^{b})$}, $L(\alpha)$) $\wedge \neg\exists$ u'.Contains({C,H,u'},{C,H,u}) $\wedge \neg\exists$ b'.Contains({C,H,b'}, {C,H,b})

Σύμφωνα με την παραπάνω υπόθεση εάν τα b και u είναι fresh values τότε το $L(\alpha)$ δε μπορεί να περιέχεται στο B και επίσης δε μπορεί να υπάρχει όρος ο οποίος να έχει σταλεί προηγουμένως από τον C στον H και να περιέχει το u ή το b.

$\upsilon_3 \equiv$ <->Fresh(H,b) $\wedge$ Contains({C', H' , G'(υ,g^{b}) } , {C, H, $G(\upsilon,g^{b})$ }) $\supset$

(C'=C $\wedge$ H'=H $\wedge$ G'=G)

Η υπόθεση αυτή ισχυρίζεται ότι εάν το b είναι fresh value και το $G(\upsilon,g^{b})$ που στάλθηκε από τον C στον H περιέχεται σε έναν άλλο όρο που στάλθηκε από έναν C' σε κάποιον H' τότε οι όροι και οι agents ταυτίζονται ανά δύο, δηλαδή C'=C και H'=H και G'=G .

$\upsilon_4 \equiv$ <->Fresh (H,u) $\wedge$ Contains({C', H' , u' } , {C, H, u }) $\supset$

(C'=C $\wedge$ H'=H $\wedge$ u'=u)

Σύμφωνα με την προηγούμενη υπόθεση (όπως και στην περίπτωση της υ_3) εάν το u είναι fresh value που στάλθηκε από τον C στον H περιέχεται σε έναν άλλο όρο που

στάλθηκε από έναν C’ σε κάποιον H’ τότε οι όροι και οι agents ταυτίζονται ανά δύο, δηλαδή C’=C και H’=H και u’=u .

Στο κεφάλαιο αυτό παρουσιάστηκαν αναλυτικά τα βασικά αξιώματα που ισχύουν στην περίπτωση του SRP. Με το $υ_4$ ολοκληρώνονται οι βασικές υποθέσεις, δηλαδή το σύνολο $Γ_{SRP}$ που πρέπει να ικανοποιείται από το πρωτόκολλο SRP.

ΚΕΦΑΛΑΙΟ 7

Συμπεράσματα και Προτάσεις για περαιτέρω έρευνα

7.1 Εισαγωγή

Στα προηγούμενα κεφάλαια είδαμε την Protocol Composition Logic , τις αποδεικτικες της μεθόδους, το πρωτόκολλο SRP και μία υλοποίηση του με τη χρήση της PCL. Στο παρόν κεφάλαιο θα αναπτυχθούν τα εξαγόμενα συμπεράσματα καθώς και προτάσεις για περαιτέρω έρευνα.

7.2 Συμπεράσματα για την Protocol Composition Logic

Η Protocol Composition Logic είναι μία λογική σχετικά εύχρηστη κυρίως λόγω του ότι οι αποδείξεις της δεν είναι πολύ μεγάλες συγκριτικά με άλλες αντίστοιχες μεθόδους. Τα θεωρήματα σύνθεσης επιτρέπουν στις αποδείξεις μεγάλων πρωτοκόλλων να προκύψουν από μικρότερες αποδείξεις απλούστερων πρωτοκόλλων τα οποία αποτελούν συστατικά τους μέρη.

Το θεώρημα της ορθότητας εγγυάται ότι για μία κλάση ιδιοτήτων ασφαλείας των πρωτοκόλλων, οι αξιωματικές αποδείξεις της PCL έχουν το ίδιο νόημα με τις «χειρόγραφες» αποδείξεις των κρυπτογράφων.

Η PCL έχει εφραμοστεί με επιτυχία σε ένα σύνολο πρωτοκόλλων, όπως Diffie-Hellman, GDOI, Asokan-Shoup-Waidner, Garay-Jacobson-MacKenzie, TLS, IEEE 802.11i, EAP-GPSK, Needham-Schroeder-Lowe, JFK, CR, ISO-9798-2, ISO-9798-3, SKID3, IKE, IKEv2, STS και άλλα.

Στο προηγούμενο κεφάλαιο εκφράστηκε το πρωτόκολλο SRP στην PCL μέσω του cord Calculus. Αυτό ήταν το πρώτο κρίσιμο στάδιο ώστε να γίνει πιο ξεκάθαρος ο τρόπος λειτουργίας του και να προσδιοριστούν τα αξιώματα .

Στη συνέχεια, με βάση τη δομή και τη λειτουργία του πρωτοκόλλου κατασκευάστηκαν τα βασικά αξιώματα. Αυτά προέκυψαν από τους ρόλους που διαδραματίζουν οι agents σε ένα τρέξιμο του SRP και είναι κυρίως αξιώματα computes.

Τέλος, προσδιορίστηκε ένα σύνολο βασικών υποθέσεων για το συγκεκριμένο πρωτόκολλο, δηλαδή το Γ_{SRP} η έννοια του οποίου ορίστηκε στο κεφάλαιο 4.

Από όλα τα παραπάνω προκύπτει ότι το SRP είναι ένα πρωτόκολλο του οποίου μπορούν να αποδειχθούν ιδιότητες secrecy και authentication με βάση τα αξιώματα και τις υποθέσεις που διατυπώθηκαν.

7.3 Προτάσεις για περαιτέρω έρευνα

Όπως αναφέρθηκε και στην προηγούμενη παράγραφο, η μετατροπή του πρωτοκόλλου από τη συνηθισμένη μορφή μηνυμάτων και βελών σε αυτή της PCL το καθιστά λειτουργικό ώστε να γίνουν λογικές επεξεργασιές που να καταλήγουν στην αποδοχή ή μη κάποιων ιδιοτήτων ασφαλείας.

Μία πιθανή εξέλιξη της δουλειάς που παρουσιάστηκε στο κεφάλαιο 6 θα ήταν, με τη βοήθεια κάποιου proof assistant να απδειχθεί εάν ικανοποιούνται κάποιες προτάσεις οι οποίες βασίζονται στο σύνολο υποθέσεων Γ_{SRP} όπως για παράδειγμα η

SRP, Γ_{SRP} |- $[Client]_C$ Honest($\hat{C}$) $\wedge$ Honest ($\hat{B}$) $\supset$ ϕ_{auth}

για την περίπτωση του authentication. Κάτι τέτοιο θα μπορούσε να γίνει με τη βοήθεια κάποιου proof assistant και ακολουθώντας την abstraction-refinement μέθοδο που περιγράφηκε αναλυτικά στο κεφάλαιο 4. Η PCL-Isabelle είναι ένα τέτοιο εργαλείο, το οποίο έχει δημιουργηθεί σε εργαστήριο του πανεπιστημίου Stanford και είναι μία εξειδικευμένη εκδοχή της Isabelle για την Protocol Composition Logic. Μία άλλη δυνατότητα θα ήταν να χρησιμοποιηθεί η CafeObj [9], αφού πρώτα περαστεί σε αυτήν η PCL (αξιώματα, θεωρήματα κτλ.).

Μία άλλη πρόταση για περαιτέρω έρευνα θα ήταν , με τη χρήση του Protocol Derivtion Assistant (πρόκειται για ένα εργαλείο που έχει αναπτυχθεί στο ίδιο πανεπιστήμιο και αναλύει τα πρωτόκολλα σε απλούστερα επιμέρους πρωτόκολλα), να αναλυθεί κάποιο σύνθετο πρωτόκολλο (όπως για παράδειγμα το SSL/TLS ή το EAP ή το SAML που περιέχουν το SRP) και στη συνέχεια με τις μεθόδους του Protocol Derivation System να καταλήξουμε στο σύνθετο πρωτόκολλο που θα έχει «κληρονομήσει» ιδιότητες από τα συστατικά του. Το Protocol Derivation System [8] είναι ένα σύστημα που υποστηρίζει μεθόδους, οι οποίες επενεργούν στα απλούστερα πρωτόκολλα για να καταλήξουμε στα πιο σύνθετα και σχετίζονται με τη διατήρηση ή διαφοροποίηση των ιδιοτήτων ασφαλείας τους. Για παράδειγμα, μία τέτοια μάθοδος είναι η sequential composition.

Γενικά πρόκειται για μία επιστημονική περιοχή με πολύ μεγάλα περιθώρια έρευνας και πολύ ενδιαφέροντα πεδία εφαρμογής. Χαρακτηριστικό είναι δε το γεγονός ότι τέτοιες μέθοδοι υιοθετούνται σε ολοένα και περισσότερα πανεπιστήμια, ερευνητικά κέντρα και εταιρείες προκειμένου να διασφαλιστεί με όσο το δυνατόν εγκυρότερο και πιο επιστημονικό τρόπο η ασφάλεια των συστημάτων.

Βιβλιογραφία :

[1] G. Berry and G.Boudol : «The chemical abstract machine», Theoretical Computer Science, 96:217-248, 1992.

[2] A.Datta : Phd Dissertation, «Security Analysis of Network Protocols : Compositional Reasoning and Complexity Theoretic Foundations», 2005

[3] A.Datta, A. Derek, J.C. Mitchell, D. Pavlovic : «Secure Protocol Composition»

[4] A.Datta, A. Derek, J.C. Mitchell, A.Roy : «Protocol Composition Logic»

[5] A. Datta, A.Roy, J.Mitchell : «Formal Proofs of Cryptographic Security of Diffie-Hellman based Protocols»

[6] A.Datta, A. Derek, J. Mitchell, D.Pavlovic : «Secure Protocol Composition», 2004

[7] A. Datta, A.Derek, J.C.Mitchell, B. Warinschi : «Computationally sound and compositional logic for key exchange protocols»

[8] A.Datta, A. Derek, J.C., Mitchell, D. Pavlovic : «Abstraction and Refinement Derivation»

[9] R.Diaconescu, K.Futatsugi : «CafeObj Report», Word Scientific, 1998

[10] N. A. Durgin: Phd Dissertation, «Logical Analysis and Complexity of Security Protocols», 2003

[11] N. Durgin, J.C. Mitchell, D. Pavlovic : «A compositional Logic for Proving Security Properties of Protocols»

[12] A.Roy : Phd thesis, «Formal proofs of Cryptograprhic Security of Network protocols», 2009

[13] A.Roy, A. Datta, A. Derek, J.C. Mitchell and J.P. seifert : «Secrecy Analysis in Protocol Composition Logic», in formal Logical Methods for Security System Security and Correctness, IOS Press 2008

[14] D. R. Stinson : «Cryptography.Theory and Practice», second edition, Chapman and Hall/CRC, 2002

[15] F.J. Thayer-Fabrega, J.C. Herzog and J.D.Guttman : «Strand Spaces : Why is a security protocol correct?» In proceedings of the 1998 IEEE Symposium on Security and Privacy, pages 160-171, Oakland, CA, May 1998, IEEE Computer Society Press

[16] Wembo Mao: «Modern Cryptography.Theory and Practice»,Hewlett-Packard Professional Books,Prentice Hall, 2004

[17] Wikipedia : «Secure Remote Password Protocol»

[18] Wikipedia : «Salt(Cryptography)»

[19] Wikipedia : «Cryptographic nonce»

[20] T.Wu. The Secure Remote Password Protocol, 1997