

Verificarea formală a protoalelor cu Tamarin

ADRIAN MANEA, 510 SLA

18 decembrie 2019

Cuprins

1	Introducere	2
2	Fundamente teoretice	3
2.1	Teoria urmelor	3
2.1.1	Obiecte de lucru și notații	3
2.1.2	Dependență și urme	5
2.2	Sisteme de rescriere	6
2.3	Analiza automată a protocoalelor	7
3	Exemple	8
	Bibliografie	9

1 INTRODUCERE

Tamarin este un produs software care este folosit în modelarea simbolică și analiza protocoalelor de securitate. Modul de operare este dat de

- introducerea unui model de protocol de securitate, ca date de intrare, prin specificarea acțiunilor pe care agenții participanți la protocol le iau;
- o specificație a adversarului;
- o specificație a proprietăților dorite ale protocolului.

Tamarin oferă o modalitate de a raționa asupra protocoalelor de securitate prin specificații, atât ale protocolului, cât și ale adversarului, într-un limbaj expresiv bazat pe reguli de rescriere asupra unor multi-seturi (i.e. într-o logică multi-sortată). Prin aceste reguli, se definește un sistem de tranziții etichetat, ale cărui stări sînt reprezentate de cunoștințele adversarului, de mesajele existente pe rețea, informații despre valorile proaspăt generate (variabilele *fresh*) și generarea de noi mesaje.

Totodată, putem specifica și ecuațional unii operatori criptografici, precum logaritmul (sau exponențierea) discret(ă) Diffie-Hellman.

Se oferă două moduri de a construi demonstrații:

- (1) Un mod *complet automat*, în care se combină reguli de deducție și raționamente ecuaționale, împreună cu argumente euristice pentru a ghida demonstrația. Dacă operația se finalizează, se returnează fie o demonstrație a corectitudinii, fie un contraexemplu, care reprezintă un atac ce încalcă proprietățile dorite.
- (2) Cum problemele de corectitudine sînt, în general, indecidabile, se oferă și un mod *asistat*, interactiv, în care utilizatorul ghidează demonstrația pas cu pas, inspectînd graful stărilor și luînd decizii combinate cu abordarea automată.

Descrierea formală a metodelor de funcționare a Tamarin este dată în [Schmidt, 2012] și [Meier, 2013], din care vom prelua cîteva elemente de bază în capitolul următor. Ideea fundamentală este următoarea: fie E o teorie cu egalitate care definește operatori criptografici, un sistem de rescriere R asupra unui multi-set, care definește un protocol și ϕ o formulă care definește o proprietate trasabilă. Tamarin poate să verifice validitatea sau satisfiabilitatea lui ϕ pentru urmele lui R modulo E .

2 FUNDAMENTE TEORETICE

2.1 Teoria urmelor

Vom prelua din prezentarea [Mazurkiewicz, 1995], autorul fiind fondatorul teoriei. Carl Petri este citat ca avînd o influență foarte importantă în dezvoltarea teoriei urmelor, prin introducerea (la vîrsta de 13 ani!) a *rețelelor Petri*, utilizate în primă fază pentru reacții chimice, dar dezvoltate apoi pentru modelarea sistemelor computaționale concurente.

Teoria urmelor a venit ca o încercare de a înțelege sistemele concurente folosind limbaje formale. În primă fază, la apariția teoriei în 1970, ea a fost folosită pentru a muta accentul de pe concurența în execuție pe non-determinism. Principalele dificultăți de depășit se legau de:

- *intercalarea proceselor* (eng. *interleaving*);
- *rafinarea* comportamentului unui sistem, prin rafinarea proceselor concurente ce se execută;
- *inevitabilitatea* unui anume comportament — aspect care intra în contradicție cu non-determinismul;
- *serializarea impusă* a tranzacțiilor.

2.1.1 Obiecte de lucru și notații

Prezentăm acum cîteva elemente de bază din teoria urmelor. Pornim cu o mulțime X , o relație binară R definită pe X care va avea anumite proprietăți (e.g. ordine totală). Vom nota mulțimea numerelor naturale $\{0, 1, 2, \dots, n, \dots\}$ cu ω . Printr-un *alfabet* vom înțelege o colecție finită de simboluri, iar alfabetele se vor nota cu majuscule grecești.

Dacă Σ este un alfabet, șirurile de simboluri cu număr finit de elemente se vor nota cu Σ^* , iar șirul de lungime zero se va nota ϵ .

Mulțimea șirurilor peste un alfabet, împreună cu operația de concatenare formează un monoid, care se va numi *monoidul liber* generat de (alfabetul) Σ .

Dacă w este un șir, vom nota cu $\text{Alph}(w)$ mulțimea simbolurilor din w (*alfabetul* lui w), iar prin $w(a)$ vom înțelege numărul de apariții ale simbolului a în șirul w .

Fie Σ un alfabet și w un șir, posibil definit peste un alt alfabet. Se definește *proiecția* lui w peste Σ funcția:

$$\pi_{\Sigma}(w) = \begin{cases} \epsilon, & w = \epsilon \\ \pi_{\Sigma}(u), & w = ua, a \notin \Sigma \\ \pi_{\Sigma}(u)a, & w = ua, a \in \Sigma \end{cases}$$

În esență, proiecția pe Σ șterge dintr-un șir toate simbolurile care nu se găsesc în Σ .

Reducerea la dreapta a simbolului a din șirul w este un șir notat cu $w \div a$ și definit prin:

$$\begin{aligned} \epsilon \div a &= \epsilon \\ (wb) \div a &= \begin{cases} w, & a = b \\ (w \div a)b, & \text{altfel} \end{cases} \end{aligned}$$

definiție care are sens pentru orice șir w și simboluri a, b .

Se poate verifica ușor că proiecția și reducerea comută, adică avem:

$$\pi_{\Sigma}(w) \div a = \pi_{\Sigma}(w \div a),$$

pentru orice șir w și simbol a .

Vom defini un *limbaj* printr-un alfabet Σ și o mulțime de șiruri definite peste Σ , adică o submulțime a Σ^* . Formal, vom deosebi două limbaje care au aceeași mulțime de șiruri, dar sînt definite peste alfabet diferite. În particular, două limbaje care conțin doar simbolurile vide, dar definite peste alfabet diferite, vor fi considerate diferite.

Limbajele se vor nota cu majuscule din alfabetul latin.

Dacă A, B sînt două limbaje definite peste același alfabet, se poate defini *concatenarea* lor AB , care conține $\{uv \mid u \in A, v \in B\}$, peste același alfabet.

Puterea unui limbaj A se definește inductiv:

$$A^0 = \{\epsilon\}, \quad A^{n+1} = A^n A, \forall n \in \mathbb{N},$$

iar iterația A^* se definește prin:

$$A^* = \bigcup_{n \geq 0} A^n.$$

În general, putem extinde funcția de proiecție de la șiruri la limbaje. Fie A un limbaj, Σ un alfabet. Se definește proiecția lui A pe Σ ca fiind limbajul $\pi_{\Sigma}(A)$ dat de:

$$\pi_{\Sigma}(A) = \{\pi_{\Sigma}(u) \mid u \in A\}.$$

Dacă w este un șir, mulțimea:

$$\text{Pref}(w) = \{u \mid \exists v, uv = w\}$$

se va numi *mulțimea prefixelor* lui w . Evident, $\epsilon, w \in \text{Pref}(w)$, iar dacă A este un limbaj peste Σ , definim:

$$\text{Pref}(A) = \bigcup_{w \in A} \text{Pref}(w).$$

În general, are loc $A \subseteq \text{Pref}(A)$, iar dacă și incluziunea reciprocă este adevărată, limbajul se numește *închis la prefixe*. Rezultă că, pentru orice limbaj A , $\text{Pref}(A)$ este un limbaj închis la prefixe.

Se definește *relația de prefix* ca fiind o relație binară $\sqsubseteq \in \Sigma^* \times \Sigma^*$ definită prin:

$$u \sqsubseteq w \iff u \in \text{Pref}(w),$$

Se poate arăta simplu că relația de prefix este o relație de ordine pentru orice mulțime de șiruri.

2.1.2 Dependență și urme

Vom numi o *dependență* orice relație finită, reflexivă și simetrică, adică o mulțime finită de perechi ordonate D , astfel încât, dacă $(a, b) \in D$, atunci $(b, a) \in D$ și $(a, a) \in D$.

Fie acum D o dependență. Domeniul lui D se va nota Σ_D și se va numi *alfabetul* lui D . Dacă D este o dependență, atunci relația $I_D = (\Sigma_D \times \Sigma_D) - D$ se numește *independența* indusă de D . Evident, această relație este simetrică și nereflexivă. În particular, relația vidă, relația de identitate și relația completă pe Σ (aceasta din urmă fiind $\Sigma \times \Sigma$) sînt dependențe. Prima are alfabetul vid, a doua este cea mai mică dependență în Σ , iar ultima este cea mai mare dependență în Σ .

De exemplu, fie relația:

$$D = \{a, b\}^2 \cup \{a, c\}^2.$$

Aceasta este o dependență și avem:

$$\Sigma_D = \{a, b, c\}, \quad I_D = \{(b, c), (c, b)\}.$$

Teoria urmelor va avea dependențele ca noțiuni primare. De asemenea, se mai pot folosi și *alfabete concurente* ca noțiuni primare, alcătuite din orice pereche (Σ, D) , unde Σ este un alfabet, iar D este o dependență sau *alfabete suport* (eng. *reliance alphabet*), alcătuite dintr-un triplet format dintr-un alfabet Σ , o dependență D și independența I indusă de D .

Ajungem în fine la definiția principală.

Definiție 2.1: Fie D o dependență. Se definește *echivalența de urmă* (eng. *trace equivalence*) pentru D ca fiind cea mai mică congruență \equiv_D în monoidul Σ_D^* astfel încât, pentru orice $a, b \in \Sigma$ să avem:

$$(a, b) \in I_D \implies ab \equiv_D ba.$$

Clasele de echivalență din Σ_D^* / \equiv_D se numesc *urme* (eng. *traces*) peste D .

Urma reprezentată de șirul w se va nota $[w]_D$, iar prin $[\Sigma^*]_D$ vom nota mulțimea factor, respectiv $[\Sigma]_D$ va nota mulțimea claselor care include și clasa vidă ($\{[a]_D \mid a \in \Sigma_D\}$).

Reluând exemplul de mai sus, pentru dependența:

$$D = \{a, b\}^2 \cup \{a, c\}^2,$$

urma peste D dată de șirul $abbca$ este:

$$[abbca]_D = \{abbca, abcba, acbba\}.$$

În general, din definiție, toate șirurile care diferă numai prin ordinea a două simboluri consecutive determină o singură urmă.

Se definește *monoidul urmelor* $M(D)$ ca avînd mulțimea suport dată de mulțimea factor Σ_D^*/\equiv_D , iar operația definită în mod natural. Amintim că există aplicația de proiecție canonică de la mulțimea Σ^* la mulțimea factor, care de fapt induce un morfism natural de monoizi:

$$\varphi_D : \Sigma^* \rightarrow M(D), \quad \varphi_D(w) = [w]_D.$$

Presupunem în continuare că lucrăm cu o dependență fixată D , astfel că vom omite indicele în cele ce urmează. De asemenea, I va fi independența indusă de D (fixată), Σ va fi domeniul lui D , toate simbolurilor vor fi din Σ_D , toate șirurile vor fi peste Σ_D , iar toate urmele vor fi peste D , dacă nu se specifică altfel.

Implicația $u \equiv v \Rightarrow \text{Alph}(u) = \text{Alph}(v)$ este clară. Rezultă că definiția $\text{Alph}([w]) = \text{Alph}(w)$ este corectă.

Fie $\sim : \Sigma^* \times \Sigma^*$ o relație binară definită astfel încît $u \sim v$ dacă și numai dacă există $x, y \in \Sigma^*$ și $(a, b) \in I$ cu $u = xaby$ și $v = xbay$. Cu această definiție, se poate vedea că \equiv este închiderea simetrică, reflexivă și tranzitivă a \sim . Cu alte cuvinte, obținem:

$$u \equiv v \iff \exists (w_0, \dots, w_n), w_0 = u, w_n = v,$$

iar pentru toți $0 < i \leq n$ are loc $w_{i-1} \sim w_i$.

Fie $[u], [v]$ două urme peste aceeași dependență. Urma $[u]$ se numește *prefix* al urmei $[v]$ (iar $[v]$ se numește un *dominant* al lui $[u]$) dacă există o urmă $[x]$ astfel încît $[ux] = [v]$.

E important de știut că:

Propoziție 2.1: Fie $[w]$ o urmă, iar $[u], [v]$ prefixe ale lui $[w]$. Atunci există cel mai mare prefix comun și cel mai mic dominant comun al lui $[u]$, respectiv $[v]$.

Demonstrație. [Mazurkiewicz, 1995, Prop. 1.3.5]. □

2.2 Sisteme de rescriere

Preluăm o prezentare sumară a subiectului din [Genet și Klay, 2000], indicînd pentru mult mai multe detalii Capitolul 3 din [van Leeuwen (ed.), 1990].

Fie \mathcal{F} o mulțime finită de simboluri, care conține și o funcție de aritate definită pe \mathcal{F} . Fie \mathcal{X} o mulțime numărabilă de variabile, $\mathcal{T}(\mathcal{F}, \mathcal{X})$ o mulțime de termeni, iar $\mathcal{T}(\mathcal{F})$ mulțimea termenilor fără variabile. Mulțimea pozițiilor dintr-un termen t , notată $\text{Pos}(t)$, este ordonată lexicografic. Șirul vid ϵ ține poziția cea mai de sus. Dacă $p \in \text{Pos}(t)$, atunci $t|_p$ este subtermenul lui t de la poziția p , iar $t[s]_p$ este termenul obținut prin înlocuirea subtermenului $t|_p$ la poziția p cu termenul s . Mulțimea variabilelor din termenul t se va nota cu $\text{Var}(t)$.

O *substituție* este o aplicație

$$\sigma : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X}),$$

care poate fi extinsă în mod unic la un endomorfism al mulțimii termenilor (cu concatenarea). Domeniul său se definește prin:

$$\text{Dom}(\sigma) = \{x \in \mathcal{X} \mid x\sigma \neq x\}.$$

Un *sistem de rescriere a termenilor* \mathcal{R} este o mulțime de reguli de rescriere de forma $l \rightarrow r$, unde $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$, iar $\text{Var}(l) \supseteq \text{Var}(r)$.

Relația $\rightarrow_{\mathcal{R}}$ indusă de un sistem de rescriere \mathcal{R} pe o mulțime de termeni $\mathcal{T}(\mathcal{F}, \mathcal{X})$ se definește astfel:

$$s \rightarrow_{\mathcal{R}} t \iff (\exists l \rightarrow r \in \mathcal{R}, \exists p \in \text{Pos}(s), \exists \sigma \text{ subst. a.î. } (l\sigma = s|_p \wedge t = s[r\sigma]_p)).$$

Închiderea tranzitivă a relației se va nota $\rightarrow_{\mathcal{R}}^*$.

2.3 Analiza automată a protoalelor

Preluăm acum din [Schmidt, 2012, Cap. 3] pentru un exemplu general de analiză formală a protoalelor de criptare care folosesc exponențierea Diffie-Hellman.

3 EXAMPLE

BIBLIOGRAFIE

- [Cremers și Mauw, 2012] Cremers, C. și Mauw, S. (2012). *Operational Semantics and Verification of Security Protocols*. Springer.
- [Diekert și Métivier, 1997] Diekert, V. și Métivier, Y. (1997). Partial commutation and traces. In *Handbook of Formal Languages*.
- [Genet și Klay, 2000] Genet, T. și Klay, F. (2000). Rewriting for cryptographic protocol verification. In McAllester, D., editor, *Automated Deduction - CADE-17*, pages 271–290, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Mazurkiewicz, 1995] Mazurkiewicz, A. (1995). Introduction to trace theory. In Diekert, V. și Rozenberg, G., editors, *The Book of Traces*, chapter 1, pages 3–41. World Scientific.
- [Meier, 2013] Meier, S. (2013). *Advancing automated security protocol verification*. PhD thesis, ETH Zürich.
- [Schmidt, 2012] Schmidt, B. (2012). *Formal analysis of key exchange protocols and physical protocols*. PhD thesis, ETH Zürich.
- [Tamarin, 2019] Tamarin, T. (2019). Tamarin-prover manual. tamarin-prover.github.io.
- [van Leeuwen (ed.), 1990] van Leeuwen (ed.), J. (1990). *Handbook of Theoretical Computer Science. Volume B: Formal Models and Semantics*. Elsevier; MIT Press.