Formal Methods for Information Security

63745 characters in 10246 words on 1614 lines

Florian Moser

June 23, 2021

1 introduction

1.1 history

1983 dolvey-yao attacker model (symbolic execution abstraction)
1983 proof that secrecy is undecidable
1989 BAN logic applicable for security protocol analysis
1995 Casper used for MitM in needham schroeder
1998 inductive approach to verify security protocols
2001 ProVerif as efficient symbolic verification tool
2001 constraint-solving for bounded sessions
2001 NP-completeness of protocol insecurity for bounded sessions
2005 avispa verification tools
2006 scyther verification tools
2012 tamarin verification tools

1.2 literature

term rewriting and all that (Baader, Nipkow, 1998)
operational semantics and verification (cremers, mauw, 2012)
formal analysis of key exchange protocols (schmidt, 2012)

1.3 protocol

distributed algorithm with emphasis on communication
set of rules determining messages between principals
small receipts but non-trivial to design

security protocol
uses cryptographic mechanisms to archive security goals
like authentication, integrity, non-repudiation
security defined relative to assumptions & goals

principles (abadi, needham)
every message should say what it means
specific clear conditions for a message to be acted on
mention names explicitly if essential to meaning
clarify why crypto is used (confidentiality, authentication, binding)
be clear which properties are assumed
beware of clock variations

1.4 security protocol analysis

analysis types often incomparable, with its own results / attacks

computational
fine-grained, but manual reduction to security
argue with probability / bitstrings

formal
coarser, but able to analyse more
use deductive or automatic reasoning
automatic formal reasoning with bounded or unbounded sessions

1.5 alice and bob notation

S → R: M for sender S sending message M to receiver R
I(A) means I poses as A
$\{M\}_K$ for asymmetric encryption under key K, $\{|M|\}_K$ for symmetric
messages do not contain identities (unless explicitly specified)

omitted
internal actions of principals
missing / invalid message actions
protocol run info (roles simply "know")

1.6 adversary

assumptions
passive (eavesdropping only)
active (intercept & send messages to anyone)
participating (legitimate use of system and/or external party)
can break old session keys
can start parallel sessions

perfect cryptography assumption
encrypted messages require decryption key to be read
hence has to be known to receiver (honest one or adversary)
assumes confidential & integer encryption

generic attacks
man-in-the-middle (A <-> I <-> B)
replay (reuse (parts of) old messages)
masquerading (pretend to be other principal)
reflection (replay messages back to originator; like from parallel runs)
oracle (use normal protocol responses as enc/dec services)
binding (use messages in different context / purpose)
type flaw (substitute different message field type; abuse parsing)
(some may overlap)

MitM example (Diffie-Hellman)
A → B: g^x
B → A: g^y
but I(B) and I(A) can sit in between, respond with g^z
then A,B still believe key shared, but actually with I
need to authenticate half-keys with signatures

reflection example
A → B: $\{N_A\}_{AB}$
B → A: $\{N_A + 1\}_{AB}$
but I(B) can intercept (1) then start parallel execution
I(B) sends (1) to A, which will increment and respond
then I(B) can resume original execution with incremented response

type flaw example (Otway-Rees)
A → B: M, A, B, $\{N_A, M, A, B\}_{AS}$
B → S: M, A, B, $\{N_A, M, A, B\}_{AS}$, $\{N_B, M, A, B\}_{BS}$
S → B: M, $\{N_A, K_{AB}\}_{AS}$, $\{N_B, K_{AB}\}_{BS}$
B → A: M, $\{N_A, K_{AB}\}_{AS}$
but I(B) can replay (1) as (4)
then A mistakes M,A,B as the K_{AB}
or I(S) replays (2) as (3)
then both A,B mistake M, A, B as K_{AB}

generic defenses
evesdropping → encrypt session keys with long-term keys
binding/reflection attack → cryptographically bind names
replay attacks → use challenge-response based on nonces

1.7 key establishment protocol example

users want to archive a session key, can use honest server
roles are initiator (A), responder (B), server (S)
intruder I (while I is not identified by others)

session key advantages
encrypted messages vulnerable to time-intensive attacks
relieves key distribution problem (for n principals, need n^2 keys)
messages from previous sessions cannot be replayed

security goals
key secrecy (key known only to A, B and S)
key freshness (A, B know key is freshly generated)
key confirmation (A, B know the other party knows the same key)

simplifications
only successful received messages are specified
roles detect which protocol run they are part of

1st attempt
A → S: A, B
S → A: K_{AB}
A → B: K_{AB}, A
but eavesdropping breaks secrecy (passive adversary)

2nd attempt

assume S has key shared with each participant
(1) A → S: A,B
(2) S → A: $\{K_{AB}\}_{SA}$, $\{K_{AB}\}_{SB}$
(3) A → B: $\{K_{AB}\}_{SB}$, A
but binding attack impersonates B (active adversary)
by capturing all messages sent by A ("injecting at (1) and (3)")
at (1) request instead (A,I) from S
at (3) capture A's forwarded $\{K_{AI}\}_{SI}$

3rd attempt
like 2nd + include other party in encrypted payload
but replay attack enforces old session key (active adversary)
by injecting at (1), and replying with a old messages

Needham-Schroeder conventional keys (NSCK)
use nonce (number used only once) to ensure freshness
A → S: A,B,N_A
S → A: $\{K_{AB}, B, N_A, \{K_{AB}, A\}_{SB}\}_{AS}$
A → B: $\{K_{AB}, A\}_{SB}$
B → A: $\{N_B\}_{AB}$
A → B: $\{N_{B-1}\}_{AB}$
but able to convince B to use old session key
by injecting at (3) and sending old $\{K_{AB}, A\}_{SB}$ to B

5th attempt
can guarantee fresh keys when both parties use nonce
B → A: A, B, N_B
A → S: A, B, N_B, N_A
S → A: $\{K_{AB}, B, N_A\}_{SA}$, $\{K_{AB}, A, N_B\}_{SB}$
A → B: $\{K_{AB}, A, N_B\}_{SB}$
but key confirmation missing (which NSCK supports)

1.8 Needham-Schroeder public key (NSPK)

(1) A → B: $\{n_a, A\}_{pk(B)}$
(2) B → A: $\{n_a, n_b\}_{pk(A)}$
(3) A → B: $\{n_b\}_{pk(B)}$
want mutual authentication of A,B

MitM
A → I: $\{n_a, A\}_{pk(I)}$
I(A) → B: $\{n_a, A\}_{pk(B)}$
B → I(A): $\{n_a, n_b\}_{pk(A)}$
I → A: $\{n_a, n_b\}_{pk(A)}$
A → I: $\{n_b\}_{pk(I)}$
I(A) → B: $\{n_b\}_{pk(B)}$
but now B does not authenticate A
as B believes to be talking to A, but talks to I(A)

prevent MitM
core issue is that message (2) can be forwarded
hence (2) must include sender $\{n_a, n_b, B\}_{pk(A)}$

2 term rewriting and protocol syntax

2.1 towards formalization

formal language
if it has well-defined syntax and semantics
additionally often deductive system to determine truth of statements
like propositional logic, first-order logic

formal model
if defined by formal language
for example of protocol & their properties
to provide mathematically sound means to reason about it

required tools for formal protocol
system specification (how it operates on suitable abstraction)
security properties (target achievements)
proof (show satisfaction)

from sequence charts to execution
sequence charts usually visualize only sender & receiver
messages are sent and received by specified parties
but execution over untrusted network, parallel executions
hence sequence chart into execution per role
instead "A → B: X" write "A sends X, B receives X"

A&B specification
model involved roles and their respectively expected / sent messages
Dolev-Yao attacker might be able to intersect / inject messages
as sender / receive decoupled, might need to alter protocol
like checking message format late when all values known

2.2 semi-formal NSPK

types
Agents A, B
Number NA, NB
Function pk

knowledge
A: A, B, pk, sk(A)
B: B, pk, sk(B)
note that nonces are not part of knowledge

actions
A → B: $\{NA, A\}_{pk(B)}$
B → A: $\{NA, NB\}_{pk(A)}$
A → B: $\{NB\}_{pk(B)}$

2.3 term rewriting

generally useful & flexible mechanism
used here to represent protocols formally

signature (Σ)
set of function symbols, each with arity n ≥ 0
functions with n = 0 called constants
like 0/0, increment/1, add/2; Σ = {0, increment, add}

term algebra ($T_{\{\Sigma\}}$)
for Σ signature, X variables, N names (all mutually disjoint)
algebra over Σ is least set such that
(1) X ∪ N ⊂ T_Σ(X, N)
(2) if t_1, ..., t_n ∈ $T_{\{\Sigma\}(X,N)}$ and f ∈ Σ of arity n
(2) then f(t_1, ..., t_n) ∈ $T_{\{\Sigma\}(X,N)}$
$T_{\{\Sigma\}(\emptyset,N)}$ (terms without variables) called ground terms
like natural numbers s(0) + s(X) + s(s(0)) ∈ T_Σ

equation (E)
pair of terms t = t'
set of equations called equation theory (Σ, E)
oriented equations (t → t', t ← t') called a rewriting rule
like E = {X + 0 = X, X + s(Y) = s(X + Y), s(0) + s(0) = s(s(0))}
like for Σ = {add, sub} then E = {add(sub(X, Y), Y) = X}

quotient algebra ($T/=_E$)
interprets term by its equivalence class
equivalence class $[t]_E$ of term t with all terms by $=_E$
for $=_E$ congruence relation given by equations E
if only interpreted syntactically called free algebra
like $[\{\{M\}_K\}_{K^{-1}}]_E$ = M

substitution (σ)
function σ : X → $T_{\{\Sigma\}}$
where σ(x) != x for finitely many x
written in postfix notation (after value)
applied homomorphically (on function arguments)
like f(t_1, t_2) σ = f(t_1 σ, t_2 σ)

substitution composition
multiple substitutions tied together called
like for σ = [x → y], τ = [y → z]
then σ τ = [x → z, y → z] (apply τ homomorphically to σ)

position (p)
term position defined by tree coordinate (as sequence of integer)
(base case) p = [] then t|p = t
(expansion) [i]*p' then t|p = t_i|p'
like for (A, (B, C)) then p = [2,1] gives B

matching
term t matches term l if substitution σ exists (t = l σ)
applying σ called matching substitution

term replacement
$t[u]_p$ denotes term t after replacing t|p with u

rewriting
for rewrite rule l → r replacing l by r
applicable if t|p = l σ exists for some p, σ
rewrite step of applying l → r is t → $t[r\sigma]_p$

unification
if substitution σ exists such that t σ $=_E$ t' σ
most general substitution if extends all others with some substitution
for E=0 ("syntactic unification"), decidable & mgu exists if unifiable

else likely undecidable (like addition + distribution)
even if decidable (associativity), can be infinitary (arbitrarily long)

equational proofs
u $<->_E$ v called E-equality step (consist of u → v and u ← v)
transitive clousure of $<->_E$ is E-equality relation $=_E$
$t_0 <->_E t_1 <->_E ... <->_E t_n$ called an equality proof
to avoid lengthy equational proofs require convergence

convergence
requires termination (finite transitions; not like E={a=b, b=a})
requires confluence (order of applying transitions irrelevant)
if both termination and confluence, then convergent
now every term has unique normal form t ↓
for convergent (Σ, E) then t $=_E$ u iff t ↓ = u ↓
like for t = s(0) + s(s(0)), then t ↓ = s(s(s(0)))

2.4 model protocols

message (t ∈ $T_{\{\Sigma\}}$)
let PV ∪ FV ⊆ N
for X set of variables (denoted uppercase A, B, ...)
for PV set of public values (denoted lowercase a,b; like agents)
for FV set of fresh values (denoted lowercase na, nb, ...; like nonces, keys)
let Σ = {pair, fst, snd, pk, sk, aenc, adec, senc, sdec}
for pair, fst, snd pair operations
for pk(t), sk(t) public/private key
for aenc(m, k), adec(c, k) asymmetric encryption/decryption
for senc(m, k), sdec(c, k) symmetric encryption/decryption

2.5 dolev-yao

public key system
for encryption E_X and decryption D_X
any user can apply E_X, only X can apply D_X
it holds that $E_X\ D_X = D_X\ E_X = 1$

adversary
controls the network (read, intercept, send)
has knowledge K with learned terms during execution

derivations
can generate fresh values (Fr(x) → K(x))
learns all messages sent (Out(x) → K(x))
sends any message (K(x) → In(x))
freely combines knowledge (K(t_1) ... K(t_n) → K(f(t_1, ..., t_n)))

3 multi-set rewriting protocol syntax

3.1 multi-set rewriting (MSR)

multiset
set of elements; each assigned a multiplicity m: X → N
like [0 → 1, 1 → 3] = [0, 3, 3, 3]
union U# and difference # works on elements

fact F
F(t_1, ..., t_k) ∈ Σ_{fact}
takes k terms t ∈ $T_{\{\Sigma\}}$ as arguments for k > 0
note that facts cannot be applied recursively (contrary to terms)

labeled multiset rewriting
l -a> r called labeled multiset rewriting rule
for l,r multiset of facts called state facts
for a multiset of facts called action facts or events
many such rules called labeled multiset rewrite system

example NSPK
[$St_{A_{1(A,tid,skA,B,pkB)}}$, Fr(NA)]
- Send(A, $\{NA, A\}_{pkB}$) →
[$St_{A_{2(A,tid,skA,B,pkB,NA)}}$, Out($\{NA, A\}_{pkB}$)]

3.2 adversary rules

to determine which messages adversary can derive
modelled as knowledge facts K(t) (representing knowledge of term t)

knowledge derivations
learns all messages sent (Out(x) → K(x))
sends any message (K(x) → In(x); labelled with K(x))
generates fresh values (Fr(x) → K(x))
combines knowledge (K(t_1) ... K(t_n) → K(f(t_1, ..., t_n)))

notes
labels have different namespace, hence overloading K(x) is fine
to combine knowledge, applies equational theory

3.3 fresh rule

to formalize unique, fresh values for nonces or thread identifiers
modelled as fresh facts F(X)

fresh & public values
fresh (FV) and public values (PV) are disjoint, countable finite
FV U PV ⊂ N of N in $T_{\{\Sigma\}(X,N)}$

fresh rule
rule [] → [Fr(N)]
no precondition; single rule which can produce such facts
each created nonce N is unique semantically

3.4 protocol rules

to formalize sending, receiving of messages & progress of protocol
use state facts to track role's progress
use Out, In facts to send / receive message

role
protocol rules (send, receive, use of fresh values, ...)
uses agent state facts to track progress
agent actually executes the role

agent state fact ($St_{R_{s(...)}}$)
$St_{R_{s(A,id,k_1,...,k_n)}}$
St for state, R for role, s number of protocol step
A agent executing, id threat identifier, k_i terms in agents knowledge

protocol rule restriction
restrict ourselves to subset of tamarin to reduce hassle
[some facts] -[label]> [some other facts]
l -a→ r is protocol rule iff (except initialization rules)
(1) only In, Fr, agent state facts in l
(2) only Out, agent state facts in r
(3) either In or Out occur in single rule, never both
(4.1) single agent state fact in each l and r
(4.2) if left contains $St_{R_{s(A,id,k_1,...,k_n)}}$
then right contains $St_{R_{s'(A,id,k_{1'},...,k_{m'})}}$ for s' = s+1
(note that some k_i might be missing right, but not other way around)
(5) every variable in r must occur in l

protocol rule examples
receive rule [$St_{A_{1(...)}}$, In(m)] - Recv(A, m) → [$St_{A_{2(...)}}$]
send rule [$St_{A_{2(...)}}$] - Send(A, m) → [$St_{A_{3(...)}}$, Out(m)]
hence either send value, or receive value (do not do both)
and always create new state out of old knowledge (to progress protocol)

executable rule (l → r)
if all knowledge terms $k_{i'}$ and out terms Out(t) ∈ r
are derivable from knowledge terms k_i, In(u), Fr(x) ∈ l

well-formedness
a rule is well-formed if it is a protocol rule and executable
hence check protocol rule definition matches
check if initialization rule → anything goes
check state incremented ($St_{R_{s(...)}} \to St_{R_{s'(...)}}$ for s' = s + 1)
check only allowed facts left and right
check only valid equational rules applied
check all knowledge present to apply equational rules
check only either In or Out used

3.5 infrastructure rules

to formalize generation of cryptographic keys
like public key infrastructure (PKI)

key-generation rule
to model PKI
[Fr(sk)] → [Ltk(A, sk), Pk(A, pk(sk)), Out(pk(sk))]
for Ltk long-term key, Pk public key, Out publish public key

initialization rule
creates new thread with some id, role R, owned by some agent
[Fr(id), Ltk(A, skA), Pk(B, pkB)]
- $Create_{R(A,id)}$ →
[$St_{I_{0(A,id,skA,B,pkB)}}$, Ltk(A, skA), Pk(B, pkB)]

4 protocol semantics and properties

4.1 protocol semantics

instance
for X being term, fact or rewrite rule
instance is result of substitution to all terms in X
when all terms are ground called ground instance

ginsts(R)
set of all ground instances of rules of multiset rewriting system R
G is set of all ground facts (and G# its respective multiset)

state
finite multiset of ground facts, state ∈ G#
like $[St_{R_{1(alice,17,k_1)}}$, Out($k_1$)]

fact types
linear if consumed by application (removed from state)
else persistent (prepended with bang ! in tamarin)

labeled multiset rewriting step
l -a> r ∈ ginsts(R) (ground instance of rule)
lin(I) $\subseteq^{\#}$ S (multiset of linear facts)
per(I) ⊆ S (set of persistent facts)
S' = S $\sim^{\#}$ lin(I)) $\cup^{\#}$ r (new state without linear facts)
—- (preconditions above, rule below)
(S, l -a> r, S') ∈ steps(R)

execution R
alternating sequence S_0, $(l_1 -a_{1\rightarrow} r_1)$, S_1, ..., S_k
for S_0 initial empty state
transition sequences valid ($S_{\{i-1\}}$, l_i - a_i $\rightarrow r_i$, S_i) ∈ steps(R)
fresh names are unique (for two transitions using same ([] → [Fr[n]]), must be equal)

trace
one execution
defined by multiset of its action labels

4.2 protocol security goals

protocol goals
authenticate messages (to bind to originator, ...)
timeliness of messages (recent, fresh, ...)
secrecy (generated keys, ...)
anonymity
non-repudiation (receipt, submission, delivery, ...)
fairness
availability
sender invariance (always same sender)

properties
semantics of protocol P is set traces(P) = || P ||
security goal also defines set traces(ϕ) = || ϕ ||

correctness
property satisfied if || P || ⊆ || ϕ ||
attack traces are || P || - || ϕ ||

4.3 formalizing security properties

using direct formulation
in terms of send/receive events
but properties are heavily protocol-dependent

using protocol instrumentation
add special claim events into protocol roles
then can express properties independently of protocol
like $Claim_{secret(A,N_A)}$ to say that N_A is only known to A

claim events
part of the protocol rules (the "labels")
cannot be observed, modified, generated by adversary
allows to formulate secrecy, authenticity, ...

frequently used events
Send(A, t), Recv(A, T), $Create_{R(A,id)}$ for protocol events
$Claim_{claimtype(A,t)}$ for claim event
Honest(A), Rev(A) for honesty / reveal events
K(t) for adversary knowledge

property specification language
F@i for timestamped event (has to hold at position i in trace)
t = u for term equality
i = j for timepoint equality
i < j for timepoint inequality

4.4 secrecy

target that adversary cannot discover guarded data

role instrumentation
insert $Claim_{secret(A,M)}$ to claim M of A secret
add at end of the role
within graphs, visualize as hexagon with text secret(M)

first definition attempt
∀ A, M, i. $Claim_{secret(A,M)@i}$
⇒ not (∃ j. K(M)@j)
but works only for passive adversary (active breaks this)

model compromised agent
by explicitly sharing long-term secrets with adversary
[Ltk(A, skA)] -Rev(A)> [Ltk(a, skA), Out(skA)]

model honest agent
agent is honest in trace tr if Rev(A) ∉ tr
must accompany any Claims to prevent trivial adversary

definition
∀ A, M, i. $Claim_{secret(A,M)@i}$
⇒ not (∃ j. K(M)@j) or (∃ B k. Rev(B, M)@k ˆ Honest(B)@i)
so either attacker did not learn secret
or some B exists which revealed secret besides declared honest
note that k, i, j unordered, as has to hold on all traces

5 authentication & other properties

5.1 authentication

guarantees existence or presence of intended communication partner
may includes an agreement on protocol execution elements

many definitions
multiple classifications / specifications worked out
Lowe's authentication hierarchy used in this course

examples
ping authentication, aliveness
weak / non-injective / injective agreement
weak and strong authentication
synchronization
matching histories

5.2 authentication hierarchy (Lowe)

increasingly stronger authentication properties
all include recentness (B was run within t time units)

aliveness
when a (role A) completes run of protocol with b (apparently role B)
then b has run protocol at some point in time

weak agreement
when a (role A) completes run of protocol with b (apparently role B)
then b has run protocol at some point in time with a

non-injective agreement
when a (role A) completes run of protocol with b (role B)
then b has run protocol at some point in time with a
both a and b agree on some message m
b was running in role B (not required before)

injective agreement
non-injective agreement + one-to-one correspondence guarantee
requires run of a (role A) and b (role B) one-to-one match

5.3 role instrumentation for authentication

commit claim (owner) authenticates running claim (peer)
formalize as term t (for A) and term u (t's view of B)
add $Claim_{commit(A,B,t)}$ to A (after A can construct t)
add $Claim_{running(B,A,u)}$ to B (after B can construct u, casually preceeding commit)

content of claimed term
may include claimed roles R_1, R_2 (usually I, R)
may include claimed message t (by A) and u (by B)

aliveness
assure that b has started (nothing more)
model by requiring b's Create label
$Claim_{commit(a,b,<>)@i}$
⇒ (∃ R j. Create(b, id, R)@j) or

(∃ X r. Rev(X)@r ˆ Honest(X)@i)

weak agreement
no agreement on term or roles (but require b talking to a)
$Claim_{commit(a,b,<>)@i}$
⇒ (∃ j. $Claim_{running(b,a,<>)@j}$) or
(∃ X r. Rev(X)@r ˆ Honest(X)@i)

non-injective agreement
agreement on R_1, R_2 and t
$Claim_{commit(a,b,<R_1,R_2,t>)@i}$
⇒ (∃ j. $Claim_{running(b,a,<R_1,R_2,t>)@j}$) or
(∃ X r. Rev(X)@r ˆ Honest(X)@i)

injective agreement
additionally requires only single commit claim in trace
not (∃ a2 b2 i2. $Claim_{commit(a2,b2,<R_1,R_2,t>)@i2}$ ˆ not (i2 = i))

5.4 examples

insert running as soon as able to construct t
insert commit as soon as able to construct u
agreement might not always be possible (due to causal dependencies)
(shorthand notation here omits implicit arguments to claims)

non-injective vs injective
A: $Claim_{sk(A)}$
A → B: $\{A, B\}_{sk(A)}$
B: $Claim_{commit(A,\{A,B\}_{sk(A)})}$
non-injective is fulfilled
but injcetive fails (attacher replays message)

5.5 aliveness vs weak-agreement

A → B: A, $\{N_A\}_{pk(B)}$
B: $Claim_{running(A)}$
B → A: N_A
A: $Claim_{commit(B)}$
aliveness fulfilled
but weak-agreement fails (attacker MitM & replaces first message A by I)
then A claims with B, but B claims with I

aliveness fails
A → B: $\{|N_A|\}_{k(A,B)}$
B: $Claim_{running(A)}$
B → A: $\{|N_A|\}_{k(A,B)}$, N_A
A: $Claim_{commit(B)}$
A → B: $\{|N_A|\}_{k(A,B)}$
but attacker can reflect message 2
then A claims $Claim_{commit(A)}$ (instead of B)

5.6 key-related properties

basic goals
key freshness
key authentication (only known to communicating parties A, B)
key confirmation of A to B (if B knows that A possesses K)
explicit key authentication = key authentication + confirmation
can be expressed with secrecy and agreement

goals for compromised keys
(perfect) forward secrecy
when long-term key reveal does not compromise previous session keys
resistance against key-compromise impersonation
when A's keys do not allow others to impersonate as different principle to A

5.7 forward secrecy

∀ A, M, i. $Claim_{secret(A,M)@i}$
⇒ not (∃ j. K(M)@j) or (∃ B k. Rev(B)@k ˆ Honest(B)@i ˆ k < j)
so M is secret unless previously compromised
but M stays secret if afterwards compromised

modified station-to-station protocol
A → B: g^x
B → A: g^y, $\{g^y, g^x, A\}_{sk(B)}$
A → B: $\{g^x, g^y, B\}_{sk(A)}$
provides forward secrecy

MTI A(0) protocol
combination of long-term & session DH keys
A has u, g^v; B has v, g^u
A → B: g^x
B → A: g^y
$K_{AB} = (g^y)^u * (g^v)^x$
but attacker compromising A & B can reconstruct

key transport w/o DH
A → B: A, N_A, $\{pk(K), B\}_{sk(A)}$
B generates K_{AB}
B → A: $\{K_{AB}\}_{pk(T)}$, $\{h(K_{AB}), A, N_A\}_{sk(B)}$
provides forward secrecy without DH

5.8 key compromise impersonation (KCI)

KCI-resistance as additional property for authentication property
if adversary learns private key of A
must not be able to impersonate others to A

example (NSL)
A → B: $\{na, A\}_{pk(B)}$
B → I: $\{na, nb, B\}_{pk(A)}$
but now can simulate B against A (hence not KCI-resistant)

6 algorithmic verification of security protocols

6.1 hard problems

post correspondence problem (PCP)
given list of pairs of words ((u_1, v_1), ..., (u_n, v_n))
find ordering such that u_1 ... u_{nb} ... = v_1 ... v_n
like (u_1, v_1) = (ab, b), (u_2, v_2) = (b, ba) has solution 2, 1

6.2 undecidability

input program P and functional specification S
output if P satisfies S or not

rice theorem
for S as non-empty, proper subset of computable functions
then verification problems of S (using programs P) are undecidable

source of unboundedness
state space can be inifinite
messages (as adversary can form arbitrary complex messages)
sessions (arbitrary number of threads)
nonces (unbounded fresh nonces)
can freely combine which dimension is bounded or unbounded
(except if sessions bounded, then nonces must also be bounded)

undecidability of secrecy (threads, messages unbounded)
undecidable with unbounded threads and messages (1983)
shown by reduction on PCP generated protocols
but artificial (require intruder intervention for execution)
undecidable when restricting to executable protocols (2001)
as undecidable for unbounded messages, sessions
also undecidable for unbounded messages, sessions, nonces

decidability in search tree (all bounded)
each node is state with root initial state
edges labelled with transition justified by rewriting rule
verify property by checking each node
browse tree exhaustively (like depth / breadth first)
when everything is bounded, then terminates (hence decidable)
even if unbounded, but attack exists, will terminate with that attack

decidability of symbolic forward search (sessions, nonces bounded)
protocol insecurity for bounded #threads is NP-complete (2001)
within NP by guessing execution for specific #threads, then checking property
NP-hardness by reduction to 3-SAT
hence NP-complete

decidability as long as two dimensions bounded
unbounded threads & nonces is undecidable (1999)
bounded messages & nonces is DEXPTIME-complete (1999)

conclusion
require at least two dimensions to be bounded
hence can either have unbounded sessions or messages
(unbounded nonces but bounded sessions impossible)

6.3 dolev-yao attacker

prolific (can arbitrary compose & send to any agent & at any time)

results in enormous search tree
but can make problem decidable by separating composition / decomposition
(this chapter assumes normalized deduction rules to simplify)

deduction rules overview
create public values, derive fresh values
compose messages out of previous knowledge
unpair message either left or right
symmetric & asymmetric encryption

derivable messages DY(M)
for K(M) the set of messages
for all t derivable from assumptions K(M)
in real setting, use equational theory to derive additional M

proof DY(M) is decidable
split intruder in composition & decomposition
composition is easily decidable
decomposition is finite closure of M

composition / decomposition deduction rules
composition rules (public values, fresh values, composition)
decomposition rules (tuple to value, symmetric & asymmetric decryption)
key for symmetric/asymmetric requires composition term
intruder receive (resulting in decomposition)
intruder send (resulting in composition)
coercion rule (switches mode of decomposition into composition)

deciding DY(M)
let DY^{up} be terms t derivable out of $K^{up(M)}$ using composition rules
same for DY^{down} and $K^{down(M)}$, but using decomposition rules
first compute M' = DY^{down} (analysed intruder knowledge)
then check if term t within $DY^{up(M')}$
to show $DY^{up(DY^{down(M)})}$ = DY(M), we have to show both < and >
< as every up/down rule has direct corresponding rule in DY(M)
> as can normalize DY(M) deductions into composition first, decomposition after
(by continuously eliminating composition-decomposition pairs)
inductively show all decomp. in DY^{down}, all comp. within DY^{up}

substitutions
deciding DY(M) also works with substitutions
instantiate substitution with value out of M
like t = $\{|X|\}_k$ → for $\{|n|\}_k \in$ M, use substitution [X → n]
or compose target term (circumventing substitution)
like t = $\{|X|\}_k$ → for k ∈ M, can create t with any m ∈ M

6.4 constraint-based automatic verification

decide if for protocol P and property ϕ
[P] ⊆ [ϕ] or equivalently [P] cap [inverted(ϕ)]
for [] traces or set of states

verification approaches
forward search by post(S) (append any S' onto existing trace S)
backward search by pre(S) (prepend any S' onto existing trace S)
for finite state, both reach fixpoint sometime
for infinite state, need symbolic representations (constraints, formulas)

symbolic forward search (bounded)
forward search through tree of symbolic executions
use post(S) transformer to find reachable states
each state as system of intruder constraints ⇒ solution = attack
decidable if bounded number of sessions
like OFMC, CL-Atse
layer 1 is tree of symbolic states (≤ 1 successor per ln(t) fact)
layer 2 solves constraints in symbolic tree; solutions = attacks

symbolic backward search (unbounded)
search backwards from negated property (hence attack states)
use pre(S) transformer to find reachable states
check if initial state is reached ⇒ yes = attack
if not & terminated, then no attack possible
like ProVerif, Maude-NPA, Scyther, Tamarin

tamarin
protocols & attackers by multiset rewriting (supports many intruder theories)
properties by first-order logic over traces (supports messages & timepoints)
verification by backwards reachability analysis
algorithm is proven sound & complete (but might not terminate)

6.5 constraint-system

set of constraints of different types

constraint types
formula
like ∃ k,i.Rev(k)@i
node i : ri for i node index, ri rewrite rule instance
like node j1 applies fresh rule, node j2 applies Out rule
edge (i,u) → (j,v) linking conclusion u of node i to premise v of node j
like result of j1 used as premise in j2

solution
pair (dg, θ)
for dg dependency graph
for θ index assignment (dg indices to index variable)
and term assignment (ground terms to variables)
such that all constraints in system satisfied
like θ = [j1 → 1, j2 → 2, x (of fresh rule) → a]

6.6 constraint reduction rules

transform constraint system into set of constraints systems

types
logical rules (working on formula constraints)
graph rules (working on node / edge constraints)

properties
soundness (no solutions are invented)
completeness (no solutions are lost)

solved constraint system
if no reduction rules can be applied anymore
then can construct a solution
index assignment by topologically ordering node constraints
term assignment by instantiating each variable with different const

constraint solver setup
add to constraint set negated initial constraint (invert property)
while non-empty constraint set, choose reduction rule and apply it
early out if solution found, return attack
else return success

logical reduction rules
simple decomposition (remove ∃, ˆ)
splitting decomposition (creates constraint systems)
contradiction (results in empty set of constraint systems)
instantiation (instantiate ∀)

graph reduction rules
fact formula (F(x)@i ⇒ node constraint i:ri for F(x) label of ri)
backwards completion (open premise ⇒ node constraint for each applicable rule)
uniqueness fresh values (collapse nodes with same fresh rule)
uniqueness linear conclusions (collapse nodes with same linear conclusion)

process
safety property τ is given (∀ ...)
negate τ into τ' to negation normal form (∃ ...)
apply logical rules and get first rule(s)
for each rule, build up deduction trees using forward reasoning
might be able to collapse premises (uniqueness of fresh values)
might be able to collapse rules (if same linear rules (Fresh) used)

example (two Fin rules)
τ given & inverted gives two Fin(x_1, k)@i rules & some side conditions
write both Fin rules and build up trees to explain their construction
note that k out of Fr(k) → hence join Fr rule of both trees
note that trees use same Fr & same rules → hence join trees
now contradicts side conditions, tree is invalid & property proven

principles for intruder deduction
separate composition / decomposition with coerce rule
derive premise of coerce rule by forward reasoning
derive every $K^{up(m)}$ and $K^{down(m)}$ at most once
$K^{up(X)}$ facts are irreducible
no coerce rule instances with pairs
⇒ avoid redundancies & unnecessary non-termination

process with intruder
set up trees as described in process without intruder
resolve open premise using intruder send rule
now need to deduce required knowledge for this send rule
apply coersion rule (for pairs, first decompose)
then try to find source; using fresh / Out rule

example (intruder)
prepare & build up tree as before
use In() rule to fulfil open premises
fully decompose In() rule preconditions
try to fulfil preconditions with Fr() and check if unifiable
else apply coerse rule (switching to composition mode)
bruteforce Out() rules & try to construct coerse rule precondition

7 tamarin

uses multiset rewriting to represent protocol
adversary message deductions given with multiset rewriting rules
properties specified in first-order logic
algorithm proven to be sound and complete

backwards reachability analysis
negates secure property, then searches for solutions
if some found ⇒ counter-example to security property
uses constraint solving

internal workflow
equational theory using builtins, functions and equations
together with protocol rules, generates variant formulas
adds restrictions then 1st constraint solving starts
security properties added to 2nd constrain solving
then shows proof with all cases or attack (or does not terminate)

tamarin notation
tilde for fresh values
bang for persistent fact
dollar for public value
else fact used only once (declared right, used left)

simplify attacks
simplify attacks by adding restrictions (like allowing only single Create rule)
restrictions apply to all properties
hence before verifying other properties need to remove them again
or formulate like restriction ⇒ property

8 noise protocol framework

8.1 noise framework

modern alternative to TLS
family of authenticated key exchanges

how TLS differs
optimized for server authentication
high code complexity (50k - 600k LoC vs Noise 5k)
allows to negotiate used crypto (adds complexity)

8.2 overview

can choose one of 53 handshakes, then rest of protocol determined
each handshake has source/destination security levels associated

source security levels
0 no authentication
1 sender authentication vulnerable to key-compromise impersonation
2 sender authentication resistant to key-compromise impersonation

destination security level
0 non-confidentiality
1 encryption to an unauthenticated recipient
2 encryption to known recipient, but vulnerable to reply
3 weak forward secrecy (message confidential if attacker compromises later)
4 weak forward secrecy even with sender's key compromise
5 strong forward secrecy

formalization
assumes ephermeral keys to be secure
defines source & destination level for each handshake
but formal definition / analysis was missing

8.3 example handshake KK

→ s
s ←
...
→ e, es, ss
← e, ee, se

notation
ephemeral key e (RAM stored, short lived)
static key s (disk stored, long-term key)
... divides pre-defined values & protocol run

explanation
requires known static key between the two parties
create ephemeral keys e, es and ss (as exponents over DL)
hence es means to build g^{es}, possible knowing e and g^s

source security claim
(authentication of sender to receiver)
sender authentication resistant to key-compromise impersonation

target security claim
(confidentiality of payload from senders perspective)
strong forward secrecy

8.4 formalizing

multiple attacker models

attacker models
active if attacker active
D_x if attacker generated keys (for x ephemercal keys, PKI)
R_x if attacker revealed keys (for x ephemercal / static keys)
$R_x{}^<$ if reveal before claim of property happened
can order adversary capabilities in partial order

security notions
secrecy, (non-)injective agreement
weak forward secrecy (R_s ˆ active) or (R_{rs} ˆ active) or ($R_s{}^<$ or $R_{rs}{}^<$)
strong forward secrecy ($R_s{}^<$ or $R_{rs}{}^<$)

analysis goal
for each noise handshake
identify strongest thread such that security notion holds

architecture
noice handshakes converted with Vacarne into tamarin code
tamarin generates proofs (˜410'000 lemmas)

reduce proof size
remove redundant conjunctions / threat models, equivalent classes, ...
like if injective agreement fails, then stronger models fail too
and vice versa

map proofs to informal levels
simplify threat models (no ephemeral key revealing)
then can map attacker model to noise security levels

conclusions
noise does not consider ephemeral key compromise
hence 5 is not necessarily more secure than 3 if e compromised
some handshakes superior to others in every way
hence inferior handshakes should be removed

9 RFID protocols & privacy

radio-frequency identification

9.1 RFID system

tag & reader over insecure channel
reader & backend over secure channel

applications
keyless entry (doors, public transport, passports)
shopping, supply chains (identify stock)
identify small animals & medical identification

RFID challenges
need to be manufactured as cheaply as possibnle
efficient symmetric crypto ist not enough for strong auth / privacy
randomness generation challenging (passive RFID cannot)

realizations
5 cents - 100 dollars depending on application
as small as 0.05mm
5cm - 200m reception
few bits until kilobytes of memory
communicate constant / apply asymmetric crypto
passive / active variants

RFID tracking
include chip in garment (like school uniform)
then can track wherever person goes

RFID properties
convenient way to register/track items
popular for access control to secure premises
RFID tags respond automatically to queries by readers

privacy implications
can trace wearer without even noticing
can trace competitors products

RFID untraceability
if attacker cannot distinguish
two protocol executions of same tag vs of different tags

example challenge (car keyless entry)
want authentication (car unlocks)
want privacy (prevent identification)
resource constraints (limited key capabilities)

9.2 e-passport

why not NSL
in NSL, tag needs to initiate
hence need to add 4th message so reader initiated
together with async crypto simply too expensive for tags

various implementations
designed with security and privacy in mind
differences in implementations allow to identify country
different error messages / response times

identify issuing country
italy / new zealand do not randomize id (traceable!)
french can be traced with reply attacker
german, greek, irish fast response time than UK
russian even slower + are not fully compliant

9.3 hopper & blum (HB) protocol

alternative to NSL which uses less resources

protocol (reader → tag)
R, T have preshared x (a k-bit secret), μ (probability)
R starts with c = 0
(repeat m times)
R picks a $\in \{0, 1\}^k$ and sends to T
T picks v ∈ {0,1} such that Pr[v=1] = μ
T sends z = (a ∗ x) XOR v (∗ bitwise multiplies & sums in mod 2)
R increments c iff a∗x = z
(end loop)
execution accepted if c > (1 - μ) ∗ m

properties
recent aliveness (tag has been alive during execution)
secrecy x against passive adversary (but not against active)
secrecy x also important for privacy

active adversary tag identification
query tags with same challenge a, then expect same answer for same tag
reader has to bruteforce expected tag based on all known secret x

9.4 trace hyperproperties

untraceability
every trace has correspoding trace
where all occurrences of tag has been replaced by different tag

non-interference
commands of "high" users are not observable by "low" users
hence for each trace with "high" commands there exists another trace
where all "high" commands have been removed

9.5 untraceability

for any trace tr with two runs of same tag
a trace tr' exists with two runs of different tags
tr indistinguishable from tr'

tamarin
can analyse observational equivalence (but tedious)
hence use here notion based on reinterpretation of message terms

idea
reinterprete attack trace
replacing some tag actor with another tag actor
and attacker cannot differentiate

9.6 reinterpretation

if permutation of ground terms & its inverse are both a semi-reinterpretation
semi-reinterpretation requires structure preservation & "further conditions"
simplified requires that attacker cannot see inside message blobs

structure preserving
π(f(m_1, ..., m_k)) = f($m_{1'}$, ..., $m_{k'}$)
does not modify pairs (simply goes inside)

further conditions
π(m) = m if m is $\sum^0$ (constant)
$\pi(\{m\}_K) = \{\pi(m)\}_K$ if attacker knows key k or can construct $\{m\}_K$
(for both symmetric / asymmetric encryption)
π(f(m)) = f(π(m)) if attacker knows m and (f = pk or hash)
π(f(m_1, ..., m_n)) = f(π(m_1), ..., π(m_n)) if f != (pk or hash)
(note that last rule is underapproximation; other f might also work like XOR)

examples
for M = {a, pk(skA), $\{a\}_{pk(skB)}$} and $\pi(\{a\}_{pk(skB)}) = \{b\}_{pk(skA)}$
valid as same structure and M cannot construct due to unknown pk(skB)
π(M) = {a, pk(skA), $\{b\}_{pk(skA)}$}
π^{-1} also valid, as do know ska (no deconstruct), or b (no construct)

9.7 indistinguishability of traces

definition
for IK(tr) knowledge of adversary derivable of trace tr
tr is indistinguishable from tr' if reinterpretation π exists such that
(1) same events executed in both traces
(2) π is reinterpration for all sent terms
(3) π is reinterpration for all received terms

traceable protocol
R sends nonce a
T responds with h(ID_T, a)
no π exists which can map different runs to different actors

untracable protocol (ID-protocol)
R sends nonce a
T responds with b, h(ID_T, a, b) for b nonce
define π = {h(ID_T, x, b') ⇒ h(ID_G, x, b')} for any x
hence can reinterprete trace with b' to as belonging to some G

9.8 tamarin untraceability

untraceability (untrac-1)
(tr_i means at position i of trace tr)
$Claim_{untrac-1(T,id)} \in tr_k$ ˆ
Create(T, id, 'T') ∈ tr_i ˆ Create(T, id, 'T') ∈ tr_j ˆ id != id'
⇒ ∃ tr' indistinguishable tr
Create(T', id, 'T') ∈ $tr_{i'}$ ˆ Create(T", id', 'T') ∈ $tr_{j'}$ ˆ T' != T"

untraceability explanation
for untrac claim of some trace tr and
for any two different runs of the same tag
there exists an indistinguishable other trace tr' and
tr' contains two runs of different tags

weak untraceability (untrac)
untraceability holds if agents not compromised
untrac-1 v (∃ R, l . Rev(R) ∈ tr_l ˆ Honest(R) ∈ tr_k)

untrac vs untrac-1
add reveal rule to protocol
then untrac-1 broken, but untrac still holds

forward untraceability (forward-untrac)
untraceability of past events holds even if agent compromised
untrac-1 v (∃ R, l . R != T ˆ Rev(R) ∈ tr_l ˆ Honest(R) ∈ tr_k)
v (revealed and ∀ $Claim_{forward-untrac}$ are after reveal)

forward-untrac vs untrac
ID-protocol is untraceable, but not forward untraceable
because learning ID_T allows to construct h(ID_T, a, b)
use public-key + nonce system
because learning the public key does not allow to decrypt past messages

strong untraceability (strong-untrac)
untraceability holds even if compromised
untrac-1 v (∃ R, l . R != T ˆ Rev(R) ∈ tr_l ˆ Honest(R) ∈ tr_k)

strong-untrac vs forward-untrac
public-key + nonce system is strongly untraceable

forward-untraceable but not strongly untraceble by type flaw
R queries nr (for honest adversaries, nr is a nonce)
T responds with {nt, nr}{_pk(R)}, {nt, K}{_pk(R)}
if R queries nr = K, then not strongly traceable
still forward-untraceable because cannot do this on past traces

examples
bit-leaking RFID protocol by Kand/Nyang 2005
required probabilistic model to demonstrate flaw
elliptic curve traceable protocol by ECRAC 2008
required 3 threads of tag to trace

10 Human Factors

human is part of the communicating parties
limited computation capabilities & error-proneness

10.1 model

human H using platform P
P communicates with server S
H might use device D over secure channel
while device D communicates with P & S

devices
RSA-secure ids
credit card terminals
voting codes
second factor verification apps

10.2 formal model

extend tamarin multi-set rewriting rules
add authentic, confidential and secure channel rules
add agents & humans

insecure channel
[SndI(A, B, m)] → [Out(A, B, m)] ("send insecure rule")
[In(A, B, m)] → [RcvI(A, B, m)] ("receive insecure rule")

authentic channel
[SndA(A, B, m)] → [!Auth(A, m), Out(A, B, m)]
[!Auth(A, m), In(B)] → [RcvA(A, B, m)]
!Auth(A,m) declares that m is authentically coming from A
In(B) models that adversary can decide upon receiver
Out() fact bc only authentic channel (adversary learns message)
in agent rules, use SendA and RcvA

confidential channel
[SndC(A, B, m)] → [!Conf(B, m)]
[!Conf(B, m), In(A)] → [RcvC(A, B, m)]
[In(A, B, m)] → [RcvC(A, B, m)]
!Conf(B,m) declares that m can only be received by B
In(A) models that adversary can fake sender
last rules enables adversary to create new messages

secure channels
[SndS(A, B, m] → [!Sec(A, B, m)]
[!Sec(A, B, m] → [RcvS(A, B, m)]

alice and bob extension
send authentic with A (filled dot) → (dot) B
send confidential with A (dot) → (filled dot) B
send secure with A (filled dot) → (filled dot) B

honest & dishonest agents
[AgSt(A, step, kn)] → [Dishonest(A)] → [Out(A, step, kn)]
[In(step, kn)] → [Dishonest(A)] → [AgSt(A, step, kn)]
hence dishonest can rewrite local state
[In(x)] → [Dishones(A)] → [Fresh(A, x)]
[Fr(x)] → [Fresh(A, x), Honest(A)] → [Fresh(A, x)]
hence dishonest can fake fresh values, honest cannot
agents can either be honest or dishonest (restrict traces)

modelling humans
communicate over provided Snd and Rcf interfaces
knowledge is modelled with agent state
can concatenate and split messages (but no crypto)

10.3 graph theoretic model

require modelling assumptions
comunication channels
agent's capabilities
initial knowledge
trustworthyness

role assumptions
human honest, only pairings / projections, usually empty start state
server/device honest, no computational restrictions
platform dishonest, no computational restrictions

(HISP) topology
nodes are honest, dishonest or honest & restricted
links are secure or insecure
HISP topology is subgraph representing assumptions

10.4 security properties

security property & protocol execution defined over traces
if protocol execution fully within security property, then fulfilled

authenticity
if B claims that A communicated with m
then A sent message m, and B learned message m

confidentiality
if A claims m is secret
then adversary does not learn m

executability lemma
for each rule add also an ∃ lemma
to prevent trivial protocols (which simply never claim Secret)

unrestricted communication
restricted use of secure channel (like yes/no vote)
unrestricted use of originating secure channel (like E-Mail)
latter modelled using Fresh(m), added to ∃ lemma

10.5 conditions for secure communication

assume A communicates to B over compromised platform

no confidential channel
for A != B and initial knowledge empty
A outgoing links are authentic, incoming confidential
B outgoing links are confidential, incoming authentic
bc can modify each valid trace with B with trace where B impersonated

no authentic channel
for A != B and initial knowledge empty
A outgoing links are confidential, incoming authentic
B outgoing links are authentic, incoming confidential

trusted device required
corollary out of the previous two lemmas

trusted device example
over path H → P → D → S
D, S share secret key K
H sends fresh(m) to D
D encrypts, sends to P, forwards to S
secure channel verified in tamarin

necessary conditions
path between H and S (obvious)
channel from H → D or D → H (else device no effect)

minimal HISP for originating secure communication
H → S requires at least secure channel H → D
S → H requires at least secure channel D → H
H is always assumed to have no initial knowledge
rather restricted as H or S have to generate value (Fr(x))

minimal HISP for secure communication
H → S requires at least H → D or D → H
S → H requires at least H → D or D → H
note that more possibilities as device could generate value

10.6 examples

USB stick authenticator
does not work, as USB works over untrusted platform
no secure link with human, hence impossible
USB must contain display or keypad

generalized conclusions
take potential topologies of use case
then can deduce possible communication pattern

10.7 human error

protocol specification unknown, errors, social engineering

model knowledge
HK(H, t, m) for t tag, m knowledge
like HK(H, "password", p)
concatenate and split messages
like [!HK(H, <t_1, t_2>, <m_1, m_2>)] → [!HK(H, t_1, m_1), ...]

human error
skilled users as infallible agents with small mistakes
inexperienced users as arbitrary behaviour with few simple rules
partial order of error behaviour

skilled users
start with correct, specified behaviour
add failure rules extending behaviour

message confusion (skilled users)
[Snd(H, < t_1, t_2), !HK(H, t_2, m_2), Fail(H, 'message confusion')]
→ [Snd(H, <t_2, m_2>)]
hence mistakenly send <t_2, m_2>
trace this mistake to be able to restrict

inexperienced users
start with random behaviour, ignorance of protocol specification
add simple guidelines ("dont expect, reject", "password email, fail")
model as trace restrictions

inexperienced users guideline examples
NoTell(H, tag) (H does not reveal tag)
NoTellExcept(H, tag, D) (H does reveal tag, except to D)
for example to D device, where password has to be entered
NoGet(H, tag) (H rejects received tag)
ICompare(H, tag) (H compares tag with its initial knowledge)

10.8 phone-based authentication protocol

authentication properties
entity authentication (recent aliveness of entity H)
device authentication (recent aliveness of device D, H has exclusive access)
message authentication (H has indeed sent m)
recent aliveness means event occurrs between two verifier events

protocol
D knows pk(skS), pw; S knows skS, pw
S sends S, fresh(r_s) to D
D responds with $\{r_d\}\{$_pk(skS)$\}$, $\{$h(r_s), H, pw$\}\{$_h(r_s, r_d)$\}$
S confirms with $\{$h(r_d)$\}\{$_h(r_s, r_d)$\}$
satisfies multual, weak agreement on h(r_s, r_d)

extend protocol
add platform between D → S; insecure channel
add human role H <-> D; secure channel

MP-auth analysis
infallible human satisfies entity authentication & device authentication
but untrained human not; as enters password on corrupted platform
use NoTellExcep(H, 'password', D) guideline for it to hold

MP-auth-MA analysis
infallible human satisfies message authentication
but untrained human not; as does not read the display
all guidelines insufficient, including ICompare
as can send OK message before applying ICompare rule
can add "enter confirmation code" exchange to enforce ICompare

11 advanced cryptographical models

11.1 signatures

KGen → (sk, pk)
Sign(sk, m) → σ
Vfy(pk, m, σ) → 1 | 0
correctness by Vfy(Sign(m, sk), m, pk) = 1 for all (sk, pk)

existencial unforegeability under adaptive chosen message attacks (EUF-CMA)
challenger calls KGen → (sk, pk), sends pk to attacker
adversary can query signing oracle for m_i, gets σ_i
adversary wins if outputs m∗, σ∗ for m∗ != m_i

decision tree
pk by KGen, σ by Sign → Verify() → true (correctness)
pk by KGen, σ otherwise → Verify() → false (EUF-CMA)
undefined if pk not generated by KGen

11.2 station to station protocol (STS-MAC)

A → B g^x
B → A g^y, $cert_b$, $\sigma_b = sig_{skb(g^x, g^y)}$, MAC($g^{xy}$, σ_b)
A → B $cert_a$, $\sigma_a = sig_{ska(g^x, g^y)}$, MAC($g^{xy}$, σ_a)

claimed properties
secrecy (finish(x,y,k) ⇒ secret(k))
identity agreement (finish(x,y,k) & finish(z,w,k) ⇒ x=z & y=w)
but can break identitiy agreement for B with bad signature properties

key substitution attack (KS)
given key substitution (different sk/pk for same valid σ)
attacker changes message (3), to add own certificate
introduce STS-ID to fix KS
include 1, id_a, id_b into σ_b in message 2
include 2, id_a, id_b into σ_a in message 3

message-key substitution attack (MKS)
given message substitution (different m', sk', pk' for same valid σ)
attacker changes message $id_a \to id_m$ in σ_a
introduce STS-KSIG to fix MKS
include public key in first message
include secret g^{xy} in signature (as attacker does not know)

re-signing (RS)
given re-signing (given σ, get σ∗ valid for different pk)
attacker resigns last signature
introduce STS-KENC to fix RS
additionally encrypt signature using public key crypto

colliding signatures (Coll)
given collision (get sk, pk, sig such that for random m Vfy suceeds)
attacker can generate signature over garbage in last step

signature properties
RSA-PKCS, RSA-PSS, DSA, ECDSA-FreeBP to all vulnerable / unknown
ECDSA-FixedBP proven safe against KS and MKS
Ed25519, Ed25519-IETF proven safe against KS, MKS, RS

11.3 symbolic verification with equations

verify/2, sign/2, pk/1, true/0
equation includes verify(sign(sk, m), m, pk(sk)) = true

analysis
if pk not generated by KGen then equation returns false
but does not reflect reality (see re-signing, colliding, ...)
requires more accurate symbolic model

key substitution
add KSgen/1
include verify(sign(sk, m), m, pk(KSGen(sign(m, sk)) = true

message-key substitution
add MKSgen/2
include verify(sign(sk, m_1), m_2, pk(MKSgen(sign(m_1, sk), m_2) = true

colliding signatures
add COLgen/1
include verify(sign(n, x), m, pk(COLgen(x) = true
does not capture high probability (always works)

re-signing
add resign/2
resign(sign(m,sk1), sk2) = sign(m, sk2)

conclusion
now properly detects errors in STS-MAC and variants
but unclear if behaviours exhaustive

11.4 symbolic verification of signatures (SVS)

use rules to specific signature behaviour (instead of equations)
introduce labels verified, result, honest
then formulate signature properties over traces

tamarin overview
protocols represented as rules
premises (input) → actions (labels) > conclusions (output)
logical formulas to add restrictions
restrict upon traces generated by rules

using behaviours
do not want to constrain us to "known" equations of attacks
(as potentially not exhaustive)
instead want to explicitly model dishonest pk to unknown result

hence attacker can essentially define outcome

signature restrictions
for correctness, require verify to return true for honest keys
Honest(pk(a)) & Verified(sign(m,r,a),m,pk(a),False) ⇒ bottom
for unforgeability, require that signature produced by sign
Honest(pk(a)) & Verified(s, m, pk(a), true) ⇒ s = sign(m,r,a)
for consistency, require verify to always output same result
Verified(s,m,pk(a),r1) & Verified(s,m,pk(a),r2) ⇒ r1 = r2

execution
SVS more accurate & faster than equational model
when SVS attack found but not found in equational model
have to try to construct attack by hand
might be false positive (if pk of required form not archievable)

11.5 WS

very popular authentication protocol
pre-TLS solution from the 2000s

X.509 mutual authentication
both I and R have certificates of each other
I → R $cert_i$, $\{request\}_{k1}$, $\{k1\}_{pkr}$, $sign_{ski(request,t)} = \sigma$
R → I $\{response\}_{k2}$, $\{k_2\}_{pki}$, $sign_{skr(response,\sigma)}$
idea was to sign initial σ to bind response to request
but breaks down due to message-key substitution of signature

MitM X.509
attacker produces σ' out of σ for different request, sk, pk
can construct valid request to server
intercepts response of server
can transform to valid response fo initiator

conclusion
signature confirmation does not work
as signatures do not identify unique messages / public keys
⇒ need to instead include message & public key into signature

11.6 lets encrypt

millions of certificates every day
proof domain ownership by editing DNS records

protocol
owner registers at lets encrypt (LE)
LE generates token and sends to owner
owner signs token and adds it to DNS entry
LE checks DNS record & verifies signature
if valid, generate certificate & sends it to owner

security issue
attacker can check DNS record to get σ
then use (pk, sk) ← MSK(σ, token') to use own token
replace own pk at lets encrypt
now σ verifies at lets encrypt

fix
use MSK secure signature or include more into signature (but fragile)
use hash(token, public key) instead of signature

11.7 best practices

sign raw messages (not signatures!)
do not reuse signatures (for example in new signatures)
use hashes if already authenticated (like DNS)

12 runtime monitoring

12.1 overview

manual verification by writing proof on paper
interactive theorem proving with program feedback
automatic verification (either static or dynamic)
static at design time (types, models, theorems
dynamic at runtime (testing, runtime monitoring, assertions)

12.2 definition

studies, develops & applies verification techniques
to check whether a run of observed system (monitoring)
satisfies or violates a given specification (verdict)
also called runtime verification, trace checking, ...

challenges
developing monitors for high-level specification
also efficiency, scalability, deployment, ...

model checking
given system model & specification
outputs counterexample or true value
guarantees that modeled system satisfies specification

software testing
given system & test cases
outputs failed assertion or true value
guarantees that system on mock inputs satisfies specification

runtime verification
given system trace & specification
outputs verdict
guarantees that system on real inputs satisfies specification
in between software testing and model checking

12.3 security

as policy compliance
security policy specifies (in)acceptable system behaviour
security mechanisms enforce policies in adversarial environment
use monitors as part of potentially many security mechanisms
like chinese wall, separation of duty, ...

as risk minimization
identify assets, threats, risk
employ countermeasures
requires threat intelligence to identify, track & counter threats
like lateral movement (infecting one system after another)
use monitors as part of potentially many countermeasures

12.4 research problems

given system which is observed by instrumentation
extracts trace/log/stream, relative to time
with specification, make verdict
may employ enforcement mechanisms back into system

questions
how to instrument
what is a trace
how to give a specification
how to compute a verdict
how to enforce

12.5 taxonomy

trace
describes observable system behaviour
totally or partially ordered sequence of observations
about systems current state / action that changes state
discrete sequence or continuous signal

time
observation ϵ labeled with time τ
dense (in Q or R) or discrete (in N)
often UNIX time
typically increasing (but might not strictly)
typically progressing (eventually increasing)

timestamp types
event time (when occurred in system)
emission time (when sent to the monitor)
ingestion time (when received by monitor)
processing time (when processed by monitor)

specification
automaton (states & transitions)
regular expressions (ab∗c)
logic (p → not q)
SQL (SELECT ∗ FROM ...)
stream processor (online algorithms)
code (most open)
cannot specify implicitly (like "no deadlocks")

features
logic (if then else)
arithmetic (0 < x < 1000)
quantification (∀, ∃)
aggregation ($\sum$, $\prod$, average)
temporal dependencies (qualitative before/after, quantitative 3min ago)

verdict

boolean (yes / no)
more granular (specify what went wrong)
even more granuar (include meta-data like which user)

deployment
offline (runs after system finished)
checks finite & complete trace in arbitrary order
online (runs during system execution)
checks finite & unbounded trace in execution order

placement
outline (in different memory space as system)
has interface to read a trace, might alter system output
inline (within same memory space as system)
perform own instrumentation, might change system

12.6 linear temporal logic (LTL)

order & existence of obervations

model
for event $\alpha_i \in \Sigma$
trace is infinite sequence α_i for time point i
like {r}, {p,r}, {p} for r, p events

semantics
always relative to trace / timepoint
atomic proposition (α, i) |= p iff p $\in \alpha_i$
negation $\not\phi$ when p $\notin \alpha_i$
conjunction ϕ ˆ ψ
next O ψ iff (α, i+1) |= ψ
until $\phi \cup \psi$ iff ϕ true 0-to-n times until ψ
out of next & until, derive
eventually <> ψ iff ψ happens at some point in the future
always [] ψ iff ψ indefinitely, always true

examples
login followed by logout [](login → <>logout)
login strictly followed by logout [](login → O<>logout)
login strictly followed by matching logout [](login → (not login ∪ logout))

past operators
previous filled-O iff (α, i-1) |= ψ
since ϕ S ψ iff ϕ true 0-to-n times after ψ
out of previous and since, derive
previously filled-<> ψ iff ψ happened at some point in the past
always filled-[] ψ iff ψ happened always until now

12.7 implementing LTL

data structure
decompose ϕ into AST of subformulas SF
$|\phi|$ = number of SF(ϕ) = number of AST entries
create boolean arrays now & pre, each of size $|\phi|$
each entry collects truth value of subformula

algorithm
start with pre all false
go trough trace from left to right
update each entry of the now array according to the subformula
for each subformula, do case distinction to combine array entries
ensure updating is "bottom-up" so precondition of subformula evaluated
in the end, replace pre by now array

example
publish → T S approve ("before publish, some approve has happened")
subformulas = {publish, approve, T S approve, public → T S approve}
pre = now = [false, false, false, false]
have event as input; for now assume event is approve
now[0] = true (as approve is current event)
now[1] = false (as publish is not current event)
now[2] = true (as now[1] || pre[2])
now[3] = true (as now[0] && now[0] = now[2])

12.8 metric temporal logic (MTL)

adds timestamp to traces

model
for events α_i Σ and timestamp $\tau_i \in$ N
trace is infinite sequence of tuples (α_i, τ_i) for time point i
like ({r}, 1), ({p,r}, 2), ({p}, 3) for r, p events

assumptions
monotonicity ∀ $\tau_i \leq \tau_{i+1}$ (multiple with same time allowed)
progress ∀ i ∃ j $\tau_i < \tau_j$ (eventual increments)

intervals
interval is [a,b] for a ≤ x ≤ b
n-skewed for {a-n |a ∈ I} (like [2,3] - 1 = [1,2])
I^- for {I-n | n ∈ N} (like $[2,3]^-$ = {[2, 3], [1, 2], [0, 1], [0,0]})

semantics
add interval to U, O S and filled-O
to get LTL operators, set I to [0, ∞)
for $\phi \cup_I \psi$, then start/end events must be within interval I

examples
publish preceded by approve within 2 units
[](publish → $black-$ <>$_{[0,2]}$ approve)

12.9 implementing MTL

data structure
decompose ϕ into AST of subformulas SF (like LTL)
additionally decompose interval rules into interval-skewed variants
interval-skewed subformulas ISF(ϕ) = SF(ϕ) + interval-skewed variants
like ISF(a $U_{[0,1]}$ b) = {a $U_{[0,1]}$ b, a $U_{[0,0]}$ b}
create arrays as before with size = |ISF(ϕ)|

algorithm
in general the same as TLT
let t be difference of τ - τ_{pre}
ϕ $S_{[a,b]}$ ψ = (ψ ˆ a=0) v (ψ ˆ t ≤ b ˆ filled-O (t-skewed rule))
first part requires last rule holds and a=0
second part requires second rule holds, and t-skewed rule

13 language-based information flow security

13.1 information flow control

prevent flow of private data into public channels

language-based security
data flow analysis used to find illicit information flows
security by program analysis (using the type system)

security levels
every program variable a has security/confidentiality level L(a)
levels form lattice (greatest lower bound GLB, least upper bound LUB)
like L_1 = (natural numbers, ≤, 0, min, max)

illicit flows
for variables l (low), h & h' (high)
explicit (l = h)
implicit (l = h == h' ? 0 : 1)

non-interference
high variables do not infer with low variables
variation in confidential input does not change public output
if s_0, s_1 agree on low variables, their outputs do too

13.2 IMP

syntax
statement with name of big-step semantics in brackets
C ::= skip (SKIP)
| x := e (ASSIGN)
| C_1; C_2 (SEQ)
| if b then C_1 else C_2 (IF_t or IF_f)
| while b do C' ($WHILE_t$ or $WHILE_f$)
s[x → v] results in state with x set to y (rest stays same)

expression security level
constants are public
expressions by least upper bound of all variables

equivalence on states
for L some security levels
s ˜L t <=> all variables in s, t of security level are equal
L = ≤ λ (all below & including λ)
L = not ≥ λ (not above or equal to λ)

typing judgements
assign valid if target variable ≥ source variable (explicit flows)
implicit flows requires to keep track of guards of current command
ensure that assign valid with source taking into account environment
like "if b then x = e" then L(x) ≥ L(e) and L(b)

13.3 security type system

each statement typed with security level λ

each rule supplemented such that λ is checked / updated
like boolean expression b defining $\lambda' = \lambda$ LUB L(b)

subsumption
larger levels subsume smaller ones
$\Rightarrow$ can replace λ with larger λ

confinement
if C : λ, then C can only modify variables $\geq \lambda$
$\Rightarrow$ type system prevents high to low flows

non-interference
for typeable program C (according to the defined system)
given two $\leq \lambda$ equivalent states s_1, s_2
then $s_{1'}$ = C(s_1) and $s_{'2}$ = C(s_2) are again equivalent

alternative using subsumption rule
no automatic λ updates anymore
but adds subsumption rule to change levels explicitly
need to guess where to apply subsumption rule
equal to initially proposed type system

alternative using bottom-up
infer lambdas from assignment statements
then check all conditions (if / while) are below lambdas
infers most general type of program
equal to initially proposed type system

termination-sensitive analysis
while x $<$ 1 do skip has λ = Low
but depending on high value does not terminate
enforce that while does not depend on confidential data