Engineering Secure Software Systems

January 19, 2021: Strong Secrecy, ProVerif: Events and Incompleteness

Henning Schnoor

Institut für Informatik, Christian-Albrechts-Universität zu Kiel

Admin: Evaluation, Exam

Lecture Evaluation

running since last week

- everybody got an access code?
- use “free text” fields

Oral Exam

date

- Tuesday, February 23
  - 9:30-12:30
  - 13:30-16:00
- each exam: ≈ **25** minutes

what to expect?

- questions cover all lecture aspects
- theory lecture: precise formal knowledge of key definitions required for discussion
- sequence: definitions, results, proofs, alternatives, …

preparation

- use available material: slides, notes, exercises
- “readiness indicator:” review questions

organization

- oral exam via BigBlueButton
- registration until Sunday, February 14:
  `https://www-ps.informatik.uni-kiel.de/pruefungsanmeldung/`, access code: 1O1BIS

Part I: Crypto Protocols

Overview

Overview

(In)distinguishability Examples

distinguishable?

$I = \left\{ k_A, k_B, k_C, \hat{k}_C, \text{yes}, \text{no} \right\}$

m	m'
$\text{enc}^{\text{a}}_{k_A}(\text{yes})$	$\text{enc}^{\text{a}}_{k_A}(\text{no})$
N_A	N_B
$\text{enc}^{\text{a}}_{k_A}(N_A, \text{yes})$	$\text{enc}^{\text{a}}_{k_A}(N_A, \text{no})$
$[\text{enc}^{\text{a}}_{k_A}(N_A), \text{enc}^{\text{a}}_{k_A}(N_A)]$	$[\text{enc}^{\text{a}}_{k_A}(N_A), \text{enc}^{\text{a}}_{k_A}(N_B)]$
$\text{enc}^{\text{a}}_{k_A}([N_A, N_A])$	$\text{enc}^{\text{a}}_{k_A}([N_A, N_B])$
$\text{hash}(\text{yes})$	$\text{hash}(\text{no})$
$\text{hash}(N_A, \text{yes})$	$\text{hash}(N_A, \text{no})$
$\text{enc}^{\text{a}}_{k_C}([N_A, \text{yes}])$	$\text{enc}^{\text{a}}_{k_C}([N_A, \text{no}])$
$[N_A, \text{enc}^{\text{a}}_{k_B}(N_A)]$	$[N_A, \text{enc}^{\text{a}}_{k_B}(N_B)]$

Algorithmic Question

decision problem

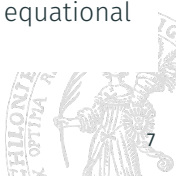
Problem:	STATIC-EQUIVALENCE
Input:	messages m and m', adversary knowledge I
Question:	are m and m' I-distinguishable?

result

polynomial-time decidable for convergent subterm theories

reference

Martın Abadi and Véronique Cortier. “Deciding knowledge in security protocols under equational theories”. In: Theoretical Computer Science 367.1-2 (2006), pp. 2–32



Exercise

Task (indistinguishability)

For the following pairs of terms, determine whether they are I-distinguishable, where $I = \left\{k_A, k_C, \hat{k}_C, \text{yes}, \text{no}\right\}$ contains the initial adversary knowledge.

$\mathbf{t_1}$	$\mathbf{t_2}$
$[N_A, \text{enc}^{\text{s}}_{N_A}(N_B)]$	$[N_B, \text{enc}^{\text{s}}_{N_B}(N_A)]$
$[N_B, \text{enc}^{\text{s}}_{N_A}(N_B)]$	$[N_A, \text{enc}^{\text{s}}_{N_B}(N_A)]$
$[N_A, \text{enc}^{\text{s}}_{N_A}(N_B)]$	$[N_A, \text{enc}^{\text{s}}_{N_B}(N_B)]$
$\text{enc}^{\text{a}}_{k_A}(N_A, \text{yes})$	$\text{enc}^{\text{a}}_{k_A}(N_B, \text{yes})$
$\text{enc}^{\text{a}}_{k_A}(N_A, \text{yes})$	$\text{enc}^{\text{a}}_{k_A}(N_A, \text{no})$
$[N_A, \text{enc}^{\text{a}}_{k_A}(\text{hash}(N_A), \text{yes})$	$[N_B, \text{enc}^{\text{a}}_{k_A}(\text{hash}(N_B), \text{yes})$
$[N_A, \text{enc}^{\text{a}}_{k_A}(\text{hash}(N_A), \text{yes})$	$[N_B, \text{enc}^{\text{a}}_{k_A}(\text{hash}(N_A), \text{yes})$

Strong Secrecy

example protocol

1. $A \to B \quad N_A$
2. $B \to A \quad \mathsf{enc}^{\mathsf{s}}_{k}([N_A, N_B])$

secrecy (privacy) of $[N_A, N_B]$

- DY model: N_B not derivable, so $[N_A, N_B]$ not derivable

  secure in DY model
- indistinguishability: $[N_A, N_B]$ distinguishable from $[N_C, N_D]$.

  insecure in SE model

yes/no situation

$A \to B \; \mathsf{enc}^{\mathsf{a}}_{k_B}(N_A, t)$ with $t \in \{\mathsf{yes}, \mathsf{no}\}$

secrecy (privacy) of yes/no

- DY model: yes, no derivable, hence message not "secret"

  insecure in DY model
- indistinguishability: $\mathcal{A}$ has no information about message content

  secure in SE model

consequence

"no implication" between security notions

Exercise

Task (strong secrecy and derivation-based secrecy)

For an equational theory E, a term t is E-derivable from a set of terms I, if there is a term M built from E-constructors (e.g., encryption functions), E-deconstructors (e.g., decryption functions) and elements from I with $M \equiv_E t$.

Example: Let E model symmetric encryption and pairing, let $I = \{k_{AC}, \underbrace{\mathsf{enc}^{\mathsf{s}}_{k_{AC}}(\mathsf{yes}, N_A)}_{=:u}\}$. Then $t = N_A$ is E-derivable from I via $M = \mathsf{proj}_2(\mathsf{dec}^{\mathsf{s}}_{k_{AC}}(u))$.

Now, the *(nonce) derivation problem* for E is to determine, given a set I of terms and a term (a nonce) t, whether t is E-derivable from I.

Show that if static equivalence for E is decidable, then the nonce derivation problem for E is also decidable.

Note: It suffices to state the (simple) algorithm deciding nonce derivation problem, which may apply the decision algorithm for static equivalence.

Strong Secrecy in ProVerif

ProVerif: distinguishability on process level

adversary: distinguish processes P_1 and P_2 by interaction

modeling

```
free c: channel.
free secret1: bitstring[private].
free secret2: bitstring[private].

let clientAlice (which:bool) =
    if which then
        out (c, secret1)
    else
        out (c, secret2).
process
    clientAlice(choice[true,false])
```

keyword: choice

- consider processes
  1. clientAlice(true)
  2. clientAlice(false)
- can attacker determine which process is running?
- can attacker distinguish secret1 and secret2?

Modeling Knowledge: Two Aspects

	expresses	ProVerif modeling
DY closure	construction / derivation of terms	**query** attacker(FAIL)
indistinguishability	knowledge about content of terms	**choice**

applications

- epistemic security properties: distinguish CDU vote from SPD vote
- strategic security properties (see, e.g., contract signing, voting)

ProVerif Scripts: Strong Secrecy

strong secrecy depends on used encryption

- `2021_01_19_lecture_10/05_strong_secrecy.pv`
  example from slides
- `2021_01_19_lecture_10/06_strong_secrecy_deterministic_encryption.pv`
  strong secrecy analysis with deterministic encryption
- `2021_01_19_lecture_10/07_strong_secrecy_randomized_encryption.pv`
  strong secrecy analysis with randomized encryption

Overview

“Weak” Secrets

“Secrets”

modeling
- nonces
- random, “long enough”
- not derivable

reality
- often “easy to remember”
- low entropy
- “dictionary attacks” possible

Popular Passwords

passwords

1. password
2. 123456
3. 12345678
4. 1234
5. qwerty
6. 12345
7. dragon
8. pussy
9. baseball
10. football
11. letmein
12. monkey
13. 696969
14. abc123
15. mustang
16. michael
17. shadow
18. master
19. jennifer
20. 111111
21. 2000
22. jordan
23. superman
24. harley
25. 1234567

PINs

1. 1234
2. 0000
3. 2580
4. 1111
5. 5555
6. 5683
7. 0852
8. 2222
9. 1212
10. 1998
11. 6969
12. 1379
13. 1997
14. 2468
15. 9999
16. 7777
17. 1996
18. 2011
19. 3333
20. 1999
21. 8888
22. 1995
23. 2525
24. 1590
25. 1235

“Weakly Secret Messages”

scenario: electronic voting

- few candidates (german election 2017: 42 parties)
- small “plaintext space”

attack scenario

- attacker knows ciphertext
- knows 42 possible plaintexts
- full search!

Election Protocol and “Weak Secrets” in ProVerif

protocol (`docs/ex_weaksecret.pv`)

```
free c: channel.
type skey.
type pkey.
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey; adec(aenc(m,pk(k)),k) = m.
free v: bitstring [private].
weaksecret v.
let V(pkA:pkey) = out(c, aenc(v, pkA)).
let A(skA:skey) = in(c,x:bitstring); let v' = adec(x, skA) in 0.
process
    new skA: skey;
    let pkA = pk(skA) in
        out (c,pkA);
        ! (V(pkA) | A(skA))
```

situation

- v not derivable
- but: v can be guessed

ProVerif Script: Weak Secrecy

command line tool

```
$ proverif 2021_01_19_lecture_10/08_weak_secrecy.pv
$ proverif -graph targetDir 2021_01_19_lecture_10/08_weak_secrecy.pv
$ proverif -html targetDir 2021_01_19_lecture_10/08_weak_secrecy.pv
```

live demo

Strong Secrecy and Weak Secrecy I

strong secrecy motivation

- indistinguishability: adversary performs term operations to find different behavior
- assumes: adversary only wants to distinguish t_1 from t_2
- models: adversary "cannot get any knowledge about the message"

weak secrets motivation

- "guessing attacks:" adversary has "candidate" c for secret s (of type t)
- adversary interacts with protocol to confirm/refute claim $s = c$
- models: adversary has "prior knowledge" about message

similarities, differences?

- difference: two defined processes in strong secrecy case
- similar: "comparison" of terms: t_1 vs. t_2, c vs. s
- similar: adversary interacts with protocol for "comparison"

Strong Secrecy and Weak Secrecy II

equivalent to weak secrecy

$P \mid \ \mathsf{phase}\ 1;\ \mathsf{out}(chan, s) \ \approx \ P \mid \ \mathsf{phase}\ 1; \mathsf{new}\ s' : t; \mathsf{out}(chan, s')$

models

- process P uses secret s
- adversary interacts with process P
- after P ends:
  - lhs: s revealed
  - rhs: unrelated value (of correct type) revealed
- adversary task: determine whether we are in lhs or rhs process
- **important**: offline attack: adversary cannot interact with P anymore
- intuitively: "can the adversary get any information during process P"?

Overview

Security Beyond Secrecy

security up to now: secrecy

theory
- **INSECURE**: derivability of FAIL
- Rusinowitch-Turuani: only secrecy

ProVerif
- query attacker(term): DY-derivation secrecy
- variants: strong / forward secrecy, weak secrets

Needham-Schroeder analysis
- key purpose: authentication
- “reduction” to secrecy $\rightsquigarrow$ exercise: unnatural modelling in ProVerif

need: different security properties
- theory: similar to secrecy (reachability)
- can use similar algorithms

realistic protocol

combination of security properties

Correspondence Properties

important class of properties: correspondence

- "if event e happens, then event e' happened before"
- required: definition of events in protocol

possible events

- S generates key k for Alice and Bob — $\mathsf{gen}(S, A, B, k)$
- Alice accepts k for communication with Bob — $\mathsf{accept}(A, B, k)$

security property

- if Alice accepts k, then k has been generated by S.
- event $\mathsf{accept}(A, B, k)$ is always preceeded by event $\mathsf{gen}(S, A, B, k)$

Events in ProVerif: Handshake-Protocol I

declarations

```
event acceptsClient(key).
event acceptsServer(key,pkey).
event termClient(key,pkey).
event termServer(key).
```

client (Alice)

```
let clientA(pkA:pkey,skA:skey,pkB:spkey)=
    out (c,pkA);
    in (c,x:bitstring);
    let y=adec(x,skA) in
        let (=pkB,k:key)=checksign(y,pkB) in
            event acceptsClient(k);
            out (c,senc(FAIL,k));
            event termClient(k,pkA).
```

events

- normal instructions in protocol
- may have parameters
- strongly typed

Events in ProVerif: Handshake-Protocol II

server (Bob)

```
let serverB(pkB:spkey,skB:sskey)=
    in c,pkX:pkey
    new k:key
    event acceptsServer(k,pkX);
    out c,aenc(sign((pkB,k),skB),pkX)
    in c,x:bitstring
    let z=sdec(x,k) in
        if pkX=pkA  then event termServer(k).
```

specification

- event $e_1(\dots)$ ==> event $e_2(\dots)$:
  - before e_1, event e_2 must have happened
  - quantifiers: lhs ∀, rhs ∃
- inj-event:
  - injective function e_1-events into e_2-events

security property

```
query attacker (FAIL).
query  x:key, y:pkey; event(termClient(x,y))==>event(acceptsServer(x,y)).
query  x:key; inj-event(termServer(x))==>inj-event(acceptsClient(x)).
```

recall earlier question: decryption statement now meaningful

question: why not always **inj-event**?

ProVerif Event Properties

limited event semantics

event $e_1(\dots)$ ==> event $e_2(\dots)$

- requirement: every e_1 is **preceeded** by some e_2
- why no analogous “followed by” requirements?

in general

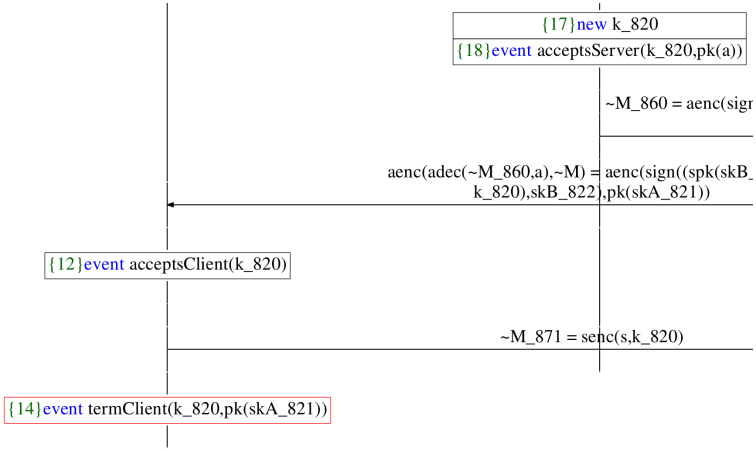
consider only “prefix-closed” properties

ProVerif Scripts: Handshake with Events

`2021_01_19_lecture_10/09_original_handshake_with_events.pv`
original (insecure) handshake protocol modelled with events

`2021_01_19_lecture_10/10_fixed_handshake_with_events.pv`
fixed handshake protocol modelled with events

ProVerif Analysis with Events




```
query x:key,y:pkey; event(termClient(x,y))==>event(acceptsServer(x,y)).
```

Correspondence and Events in ProVerif

events

- express *correspondence*
- in particular: “synchronisation” between instances (no dummy messages required)
  - **query** AliceEvent(…) ⇒ BobEvent(…)
- natural way to model authentication
- theory: how do we formalize this?
  - similar to secrecy (still reachability property)

application: key exchange

- Alice only accepts keys from Bob
- Bob only accepts if Alice confirms

missing?

- key remains secret
- later messages remain secret

realistic protocols

combination of security properties

Exercise

Task (secrecy properties and events)

In the lecture, two different kinds of (trace) properties were discussed:

- secrecy properties, modeled with derivability of the constant FAIL and in ProVerif using the statement
  query attacker(FAIL),
- event properties, modeled in ProVerif using the specification **event** and queries like
  query x:key; event(termServer(x)) ⇒ event(acceptsClient(x)).

Is one of these concepts more powerful than the other? In other words, can you “translate” any secrecy query into an event quary and/or vice versa? Which, if any, extensions would our theoretical model require to be able to handle event properties?

Note: The point of this exercise is not for you to actually specify a (rather cumbersome) translation, but to conceptually consider the relationships and differences between these two types of properties.

Overview

ProVerif and Undecidability

recall

UNBOUNDED-INSECURE is undecidable

consequence for ProVerif

there are (infinitely many) protocols P, for which ProVerif

1. returns **secure**, though P is not secure, or
2. returns **insecure**, though P is secure, or
3. does not terminate, or
4. returns **unknown**.

examples?

- for which protocols does this happen?
- how can we avoid this?
  - re-write protocols automatically to ensure decision
  - clearly not possible for every protocol

Example for Incomplete Analysis

protocol (ProVerif)

```
free c:channel.
process
    new k : key;
    out (c, senc(senc(FAIL,k),k));
    in (c, x:bitstring);
    out (c, sdec(x,k))
```

protocol secure?

“almost equivalent” for ProVerif

```
free c:channel.
process
    new k : key;
    out (c, senc(senc(FAIL,k),k));
    ! (
        in (c, x:bitstring);
        out (c, sdec(x,k))
        )
```

protocol secure?

question

why does ProVerif treat both protocols identically?

Logic Modeling Outline

facts

- $d(\hat{k}_C)$
- $d(\{0,1\})$
- …

DY deductions

- $d(\text{enc}^{\text{a}}_{k_C}(x)) \wedge d(\hat{k}_C) \rightarrow d(x)$
- $d(x) \wedge d(y) \rightarrow d([x,y])$
- $d(x) \rightarrow d(\text{hash}(x))$
- …

protocol deductions

- $d(\text{enc}^{\text{a}}_{k_B}([A,x])) \rightarrow d(\text{enc}^{\text{a}}_{k_A}([B,x]))$
- $d(\text{enc}^{\text{a}}_{k_A}([B,x,y])) \rightarrow (\text{enc}^{\text{a}}_{k_B}(y))$
- …

Horn clauses

$(x_1 \wedge x_2 \wedge \cdots \wedge x_n \rightarrow y) \quad \leftrightarrow \quad (\overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_n} \vee y)$

target clause

$\neg d(\text{FAIL})$

ProVerif Script: Incompleteness Example

command line tool

```
$ proverif 2021_01_19_lecture_10/11_incomplete.pv
$ proverif -graph targetDir 2021_01_19_lecture_10/11_incomplete.pv
$ proverif -html targetDir 2021_01_19_lecture_10/11_incomplete.pv
```

live demo

Overview

ProVerif Summary

seen

- ProVerif can analyse examples treated so far in lecture
- analysis of unbounded sessions “possible in practice”
- limitations: seen here only in contrived examples, also occur for complex protocols/properties (e.g., voting)
- ProVerif features beyond formal model:
  - strong/weak secrecy, events

caveats

- protocols can be “secure” for trivial reasons:
  event(a) ==> event(b)
  is satisfied if a never happens
- also add “liveness” tests to protocol

Exercise

Task (ProVerif modeling of Needham Schroeder)

Study the modeling of the Needham Schroeder protocol given in the ProVerif distribution (various models of the protocol can be found in `examples/pitype/secr-auth/`). Which additional properties were modeled compared to our models from the lecture and exercise class?

Overview

Crypto Protocols Summary

covered in lecture

1. theoretical foundations
   - terms and cryptographic primitives (DY, tests)
   - protocol model and security (instances, execution order, derivability of FAIL)
   - results
     - insecurity is NP-complete [RT03]
     - “parallel analysis” necessary and undecidable
2. approaches to undecidability: incomplete algorithms
   - ~~abstraction~~, Horn approach
3. tool: ProVerif
   - examples
   - strong secrecy, indistinguashability
4. complex examples
   - ~~voting protocols: Norway protocol, FOO92~~
   - ~~BitCoin~~