Engineering Secure Software Systems

Winter 2020/21, Weeks $\approx 8 - 10$: Automatic Analysis: Practice with ProVerif

Henning Schnoor

Institut für Informatik, Christian-Albrechts-Universität zu Kiel

Part I: Crypto Protocols

Overview

Applying the Theory: Analysis in Practice

so far in lecture

- formal model
- formal security properties
- decidability and complexity results
- Horn modeling
- ~~abstraction~~

now: application of the theory

different analysis tools

- **ProVerif (Bruno Blanchet, Vincent Cheval)**
- CryptoVerif (Bruno Blanchet, David Cadé)
- FDR (Formal Systems Europe)

ProVerif in Lecture I

outline

- ProVerif introduction (syntax, semantics, examples)
- generalized cryptographic primitives in ProVerif
- extended security properties in ProVerif

no detailed theory (literature references)

key references

paper	Bruno Blanchet. “Using Horn Clauses for Analyzing Security Protocols”. In: Cryptology and Information Security Series 5 (2011), pp. 86–111
update	Vincent Cheval, Véronique Cortier, and Mathieu Turuani. “A Little More Conversation, a Little Less Action, a Lot More Satisfaction: Global States in ProVerif”. In: 31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018. IEEE Computer Society, 2018, pp. 344–358. ISBN: 978-1-5386-6680-7. DOI: 10.1109/CSF.2018.00032. URL: `https://doi.org/10.1109/CSF.2018.00032`
tool	`http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`

ProVerif in Lecture II

learning goals

practice	analyzing protocols with ProVerif
security properties	study of complex properties (without detailed formal model)
conceptual	undecidable problems in real life
	consequences for tool application

not a learning goal

complete introduction to ProVerif (see reading exercise on Needham-Schroeder modeling)

ProVerif Applications

non-trivial examples

- certified email protocol, including ssh layer, JFK protocol (candidate for IKE replacement), secure filesystem Plutus, web service verification
- e-voting protocols, authenticated routing, zero knowledge protocols
- TLS (F# implementation for .NET), 5G EAP-TLS
- Telegram, Bitcoin Smart Contracts, Healthcare protocols
- …(see Google Scholar, ProVerif since 2020)

original reference

Bruno Blanchet. "Using Horn Clauses for Analyzing Security Protocols". In: Cryptology and Information Security Series 5 (2011), pp. 86–111

ProVerif

features

- analysis of protocols in symbolic model
- can handle an unbounded number of protocol sessions (necessarily incomplete)
- user-provided specification of cryptographic primitives and security properties

example security properties

- secrecy
- strong secrecy
- authentication
- correspondence
- observational equivalence

incompleteness consequences

- “don’t know”
- non-termination

Overview

ProVerif „Hello, World“

`hello.pv` (from PV doc)

```
free c:channel.

free Plain:bitstring [private].
free RSA:bitstring [private].

query attacker(RSA).
query attacker(Plain).

process
  out(c,RSA);
  0
```

elements

free	free algebraic variables
channel	communication channel
bitstring	data type
private	$\mathcal{A}$ cannot (directly) access
query	security property
out	send on channel

execution

- send “private” value RSA on public channel
- query: secrecy for
  - RSA
  - Plain

ProVerif usage

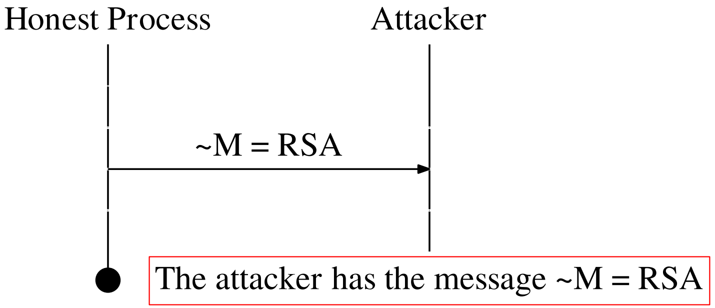
command line tool

```
$ proverif 2021_01_05_lecture_08/01_hello.pv
$ proverif -graph targetDir 2021_01_05_lecture_08/01_hello.pv
$ proverif -html targetDir 2021_01_05_lecture_08/01_hello.pv
```

live demo

ProVerif Analysis Result: Hello, World

A trace has been found.



The Handshake Protocol

example: handshake (from ProVerif tutorial)

$A \to B \quad k_A$

$B \to A \quad \mathrm{enc}^{a}_{k_A}(\mathrm{sig}_{k_B}([k_B, k]))$

$A \to B \quad \mathrm{enc}^{s}_{k}(\mathrm{FAIL})$

properties?

- intended security properties?
- protocol secure?

attack

$\mathcal{A} \to B \quad k_{\mathcal{A}}$

$B \to \mathcal{A} \quad \mathrm{enc}^{a}_{k_{\mathcal{A}}}(\mathrm{sig}_{k_B}([k_B, k]))$

$A \to B \quad k_A$

$\mathcal{A} \to A \quad \mathrm{enc}^{a}_{k_A}(\mathrm{sig}_{k_B}([k_B, k]))$

$A \to B \quad \mathrm{enc}^{s}_{k}(\mathrm{FAIL})$

fix

add receiver to message

Analyzing the Handshake Protocol in ProVerif

steps

1. specify cryptographic primitives
2. specify communication infrastructure
3. specify adversary goal
4. specify Alice & Bob
5. specify “main process”

ProVerif: Modeling Protocols With Crypto

lecture so far

behavior of crypto primitives

- asymmetric encryption
- symmetric encryption
- signatures
- hash functions

ProVerif: flexible modeling of primitives

technique: equational theories, more general than DY-like specification ısee example scripts

Overview

Crypto Primitives: Generalizing Dolev-Yao

definition

- equation: pair (l, r) of terms, also written $l = r$
  - left: “complex” term
  - right: “simple” term
- equational theory: set E of equations

caveat: choose term signature matching to E (implicit)

rewrite relation

- $t_1 \twoheadrightarrow_E t_2$: t_2 obtained from t_1 by applying rule from E
- $\twoheadrightarrow_E^*$: closure of $\twoheadrightarrow_E$ under transitivity, reflexivity, application of function symbols
- $\equiv_E$: closure of $\twoheadrightarrow_E^*$ under symmetry and transitivity

Equational Theories: Examples

primitives

- asym. encryption $\text{dec}^{\text{a}}_{\hat{k}_A}\left(\text{enc}^{\text{a}}_{k_a}(x)\right) = x$
- sym. encryption $\text{dec}^{\text{s}}_{k}\left(\text{enc}^{\text{s}}_{k}(x)\right) = x$
- signature
    - $\text{check}(k_A, \text{sig}_{k_A}(x)) = \text{ok}$
    - $\text{extr} - \text{key}(\text{sig}_{k_A}(x)) = k_A$
    - $\text{extr} - \text{msg}(\text{sig}_{k_A}(x)) = x$
- hash function
- bit commitment $\text{open}\left(\text{bc}\left(k, b\right), k\right) = b$
- trapdoor commitments
    - $\text{open}\left(\text{tdc}\left(m, r, t_d\right), r\right) = m$
    - $\text{tdc}\left(m_2, f(m_1, r, t_d, m_2), t_d\right) = \text{tdc}\left(m_1, r, t_d\right)$

“Simple” Equational Theories

definition

$\twoheadrightarrow_E$ is

- confluent, if for all t, t_1, t_2 with $t \twoheadrightarrow_E^* t_1$ and $t \twoheadrightarrow_E^* t_2$ there is some t' with $t_1 \twoheadrightarrow_E^* t'$ and $t_2 \twoheadrightarrow_E^* t'$.
- terminating, if there is no infinite sequence $t_1, t_2, \ldots$ with $t_i \neq t_{i+1}$ and $t_i \twoheadrightarrow_E t_{i+1}$ for all i.
- E is convergent, if $\twoheadrightarrow_E$ is confluent and terminating
- E is convergent subterm theory, if
    - E is convergent, and
    - for all $(l, r) \in E$: r is subterm of l or constant

Convergent Theories

definition

A term t is in E-normal-form if $t = t'$ for all $t \twoheadrightarrow_E t'$.

lemma & definition

If E is convergent, then for every term t, there is a unique term $[t]$ with

- $[t]$ is in E-normal-form,
- $t \equiv_E [t]$.

lemma

$t \equiv_E t'$ iff $[t] = [t']$.

computation of normal form

How do we compute $[t]$ from t? for a convergent theory?

Equational Theories: Examples Revisited

primitives

- asym. encryption $\mathrm{dec}^{a}_{\hat{k}_A}\left(\mathrm{enc}^{a}_{k_a}(x)\right) = x$
- sym. encryption $\mathrm{dec}^{s}_{k}\left(\mathrm{enc}^{s}_{k}(x)\right) = x$
- signature
  - $\mathrm{check}(k_A, \mathrm{sig}_{k_A}(x)) = \mathrm{ok}$
  - $\mathrm{extr} - \mathrm{key}(\mathrm{sig}_{k_A}(x)) = k_A$
  - $\mathrm{extr} - \mathrm{msg}(\mathrm{sig}_{k_A}(x)) = x$
- hash function
- bit commitment $\mathrm{open}\left(\mathrm{bc}(k, b), k\right) = b$
- trapdoor commitments
  - $\mathrm{open}\left(\mathrm{tdc}(m, r, t_d), r\right) = m$
  - $\mathrm{tdc}\left(m_2, f(m_1, r, t_d, m_2), t_d\right) = \mathrm{tdc}\left(m_1, r, t_d\right)$

discussion

- complexity?
- observations?

remember

modeling primitives at this level of abstraction loses details

algorithms

algorithms discussed so far do not cover all of these

Exercise

Task (Missing Proof)

Prove the following lemma that was stated in the lecture without proof:

If E is a convergent equational theory, then:

1. For every term t, there is a unique term $[t]$ with
   - $[t]$ is in E-normal-form,
   - $t \equiv_E [t]$.
2. For terms t and t', we have that $t \equiv_E t'$ if and only if $[t] = [t']$.

Exercise

Task (“Badly-Behaved” Equational Theories)

Define equational theories for which the resulting rewrite relation $\twoheadrightarrow_E$ is not a convergent subterm theory, i.e., one that is not confluent, not terminating, or not a subterm theory.

Algorithms for Convergent Subterm Theories

Theorem

For convergent subterm theories, the following problems are polynomial-time decidable:

- given E, t, t', does $t \twoheadrightarrow_E t'$ hold?
- given E, t, t', is $t \equiv_E t'$?

Theorem

Also computable in polynomial time: given E, t, compute $[t]$ (DAG representation)

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

Protocol Notation with Equational Theories

Needham Schroeder

Alice	ϵ	$\to$	$\text{enc}^{\text{a}}_{k_B}(A, N_A)$
	$\text{enc}^{\text{a}}_{k_A}(N_A, y)$	$\to$	$\text{enc}^{\text{a}}_{k_B}(y)$
Bob	$\text{enc}^{\text{a}}_{k_B}(A, x)$	$\to$	$\text{enc}^{\text{a}}_{k_A}(x, N_B)$

equational theory

$$\text{dec}^{\text{a}}_{\hat{k}_A}\left(\text{enc}^{\text{a}}_{k_a}(x)\right) = x$$

“simplification”

- new notation for protocols?
- advantage?

additional equations for pairing

- $\text{split1}(\text{pair}(x, y)) = x$
- $\text{split2}(\text{pair}(x, y)) = y$

Alice

$$\epsilon \to \text{enc}^{\text{a}}_{k_B}(A, N_A)$$
$$x \to \text{enc}^{\text{a}}_{k_B}\left(\text{split2}\left(\text{dec}^{\text{a}}_{\hat{k}_A}(x)\right)\right)$$

Bob

$$y \to \text{enc}^{\text{a}}_{k_A}\left(\text{pair}\left(\text{split2}\left(\text{dec}^{\text{a}}_{\hat{k}_B}(y)\right), N_B\right)\right)$$

Handshake-Protocol in ProVerif: Primitives

symmetric encryption

```
type key
fun senc(bitstring, key): bitstring
reduc forall m:bitstring, k:key; sdec(senc(m,k),k) = m
```

asymmetric encryption

```
type skey
type pkey
fun pk(skey): pkey
fun aenc(bitstring, pkey): bitstring
reduc forall m:bitstring, sk:skey; adec(aenc(m,pk(sk)),sk) = m
```

signatures

```
type sskey
type pskey
fun spk(sskey): spkey
fun sign(bitstring, sskey): bitstring
reduc forall m:bitstring, ssk:sskey; getmess(sign(m,ssk)) = m
reduc forall m:bitstring, ssk:sskey; checksign(sign(m,ssk),spk(ssk)) = m
```

Handshake-Protocol in ProVerif: Communication and Adversary Goal

channel, values, attacker goal

```
free c:channel
free FAIL:bitstring [private]
query attacker (FAIL)
```

models

- c is public channel
- FAIL: bitstring, private (not in initial $\mathcal{A}$ knowledge)
- security property: $\mathcal{A}$ cannot derive FAIL

Handshake-Protocol in ProVerif: Alice Process

protocol

$$\begin{array}{ll} A \rightarrow B & k_A \\ B \rightarrow A & \mathrm{enc}^{a}_{k_A}\left(\mathrm{sig}_{k_B}\left([k_B, k]\right)\right) \\ A \rightarrow B & \mathrm{enc}^{s}_{k}\left(\mathrm{FAIL}\right) \end{array}$$

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
            out (c,senc(FAIL,k))
```

features

- keys as arguments
- FAIL global value
- decryptions explicit (pattern matching in formal model)

Handshake-Protocol in ProVerif: Bob Process

protocol

$A \rightarrow B$	k_A
$B \rightarrow A$	$\mathrm{enc}^{a}_{k_A}\left(\mathrm{sig}_{k_B}\left([k_B, k]\right)\right)$
$A \rightarrow B$	$\mathrm{enc}^{s}_{k}(\mathrm{FAIL})$

server (Bob)

```
let serverB(pkB:spkey,skB:sskey)=
    in (c,pkX:pkey)
    new k:key
    out (c,aenc(sign((pkB,k),skB),pkX))
    in c,x:bitstring
    let z=sdec(x,k) in 0
```

features

- type checking on message receive
- last line: “no match” if decryption fails

Handshake-Protocol in ProVerif: Main Process

previous processes

- clientA(pkA:pkey,skA:skey,pkB:spkey)
- serverB(pkB:spkey,skB:sskey)

main process

```
new skA:skey
new skB:sskey
let pkA=pk(skA) in out(c,pkA)
let pkB=spk(skB) in out(c,pkB)
( (!clientA(pkA,skA,pkB) | (!serverB(pkB,skB))) )
```

features

- key generation, sending of pkA, pkB on public channel
- call of Alice and Bob with matching parameters
- !: “replication operator”

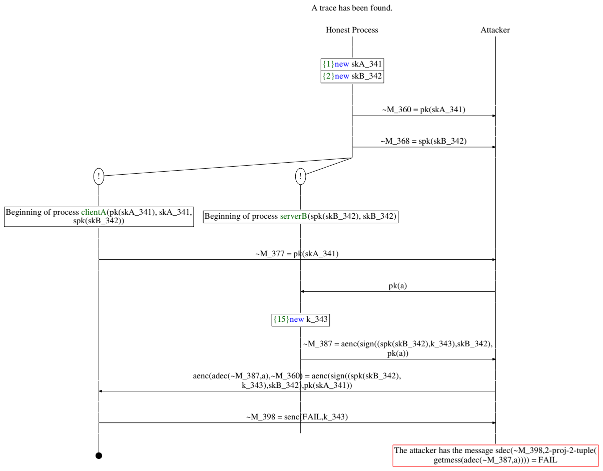
ProVerif Script: Handshake Protocol

command line tool

```
$ proverif 2021_01_05_lecture_08/02_handshake.pv
$ proverif -graph targetDir 2021_01_05_lecture_08/02_handshake.pv
$ proverif -html targetDir 2021_01_05_lecture_08/02_handshake.pv
```

live demo

ProVerif Analysis Result: Handshake


A trace has been found.
Honest Process
Attacker
{1}new skA_341
{2}new skB_342
~M_360 = pk(skA_341)
~M_368 = spk(skB_342)
!
!
Beginning of process clientA(pk(skA_341), skA_341, spk(skB_342))
Beginning of process serverB(spk(skB_342), skB_342)
~M_377 = pk(skA_341)
pk(a)
{15}new k_343
~M_387 = aenc(sign((spk(skB_342),k_343),skB_342), pk(a))
aenc(adec(~M_387,a),~M_360) = aenc(sign((spk(skB_342), k_343),skB_342),pk(skA_341))
~M_398 = senc(FAIL,k_343)
The attacker has the message sdec(~M_398,2-proj-2-tuple(getmess(adec(~M_387,a)))) = FAIL

Exercise

Task (ProVerif example)

Consider the following protocol:

1. $A \rightarrow B$ $\quad \text{enc}^{s}_{k_{AB}}(N_A)$
2. $B \rightarrow A$ $\quad [\text{enc}^{s}_{k_{AB}}(N_B), N_A]$
3. $A \rightarrow B$ $\quad N_B$

Here, k_{AB} is a long-term symmetric key shared by Alice and Bob. Is the protocol secure in the sense, that it can only be completed correctly if both Alice and Bob participate in the protocol run? Analyse the protocol “by hand” and using ProVerif.

Note: If you use the standard ProVerif **query attacker**(FAIL) modeling, you need to express the “participation property” as secrecy property. We will study a different method using events later in the lecture.

Exercise

Task (ProVerif examples)

Choose a cryptographic protocol and use ProVerif to analyze its security properties, that is:

1. Specify the protocol in ProVerif (including the required cryptographic primitives),
2. specify the security property in ProVerif,
3. run ProVerif to search for attacks. Does the result match with your expectations?

You can use any protocol you find interesting—all the protocols mentioned in the course so far are good candidates. The following is an incomplete list:

- your modeling of the WhatsApp authentication protocol in the first exercise,
- the (broken) authentication protocols presented in the first exercise class and their fixes,
- the Needham-Schroeder(-Lowe) protocol,
- the Woo-Lam protocol,
- the ffgg protocol,
- the repaired version of the handshake-protocol from the ProVerif tutorial (the broken version was discussed in the lecture).

Overview

Randomized Primitives in ProVerif I

encryption formally

- up to now: $\mathsf{enc}^{\mathsf{s}}_{k}(t)$, $\mathsf{enc}^{\mathsf{a}}_{k}(k_A)$
- problem? multiple encryptions

encryption in practice

- encrypting twice: different ciphertexts
- randomized algorithms

fact (formal statement: see cryptography lecture)

secure encryption **must be** randomized

Randomized Asymmetric Encryption

ProVerif

```
type skey.
type pkey.
type coins.
fun pk (skey): pkey.

fun internal_aenc(bitstring, pkey, coins): bitstring

letfun aenc(x:bitstring, y:pkey) = new r:coins; internal_aenc(x,y,r).

reduc forall m:bitstring, k:skey, r:coins;
    adec(internal_aenc(m,pk(k),r),k)=m.

reduc forall m:bitstring, pk:pkey, r:coins, getkey(internal_aenc(m,pk,r))=pk.
```

ProVerif Script: Randomized Encryption

command line tool

```
$ proverif 2021_01_05_lecture_08/03_randomized-encryption.pv
$ proverif -graph targetDir 2021_01_05_lecture_08/03_randomized-encryption.pv
$ proverif -html targetDir 2021_01_05_lecture_08/03_randomized-encryption.pv
```

live demo

ProVerif: Abstraction Level?

symbolic models

- messages as terms
- abstraction of cryptography
- no polynomial-time restriction
- no “real randomness”
- no probabilities

What kind of model does ProVerif use?

cryptographic models

- messages as bitstrings
- real cryptographic algorithms
- real randomness
- probabilistic polynomial-time algorithms
- probabilistic security guarantees

Overview

Typing

note

- in ProVerif, each value is **typed**
- allows to distinguish e.g., keys from nonces
- drawback?
  - requires implementation to check types!
- untyped version: use **bitstring**
- typed protocols: annotate messages with types

type flaw attacks

- attacks that use "wrong types"
- rely on protocols **not** checking types
- seen examples in ffgg protocol, Otway-Rees protocol

reference

James Heather, Gavin Lowe, and Steve Schneider. "How to Prevent Type Flaw Attacks on Security Protocols". In: Journal of Computer Security 11.2 (2003), pp. 217–244. URL: `http://content.iospress.com/articles/journal-of-computer-security/jcs162`

Overview

ProVerif: Specification

process algebra: (applied) pi calculus

- classic technique
- specification of communicating processes
- strong type system

security-specific aspects in ProVerif

- specification of crypto primitives with equational theories
- algorithms, properties

references

- Robin Milner, Joachim Parrow, and David Walker. “A Calculus of Mobile Processes, I and II”. In: Inf. Comput. 100.1 (1992), pp. 1–77. DOI: `10.1016/0890-5401(92)90008-4`. URL: `http://dx.doi.org/10.1016/0890-5401(92)90008-4`
- Davide Sangiorgi and David Walker. The Pi-Calculus - a theory of mobile processes. Cambridge University Press, 2001. ISBN: 978-0-521-78177-0

ProVerif: Syntax

part 1: terms

$M, N =$

a, b, c, k, m, n, s	names
x, y, z	variables
$(M_1, \ldots, M_k)$	tuples
$h(M_1, \ldots, M_k)$	application of functions (constructors / deconstructors)
$M = N$	equality
$M <> N$	inequality
$M \&\& N$	conjunction
$M \|\| N$	disjunction
$\textbf{not}(M)$	negation

ProVerif: Syntax

part 2: processes

$P, Q =$

0	nothing
$P \mid Q$	parallel execution
$!P$	unbounded replication, $!P = P \mid !P$
new $n : t; P$	value n of type t in process P
in$(c, x : t); P$	store message from channel c in x, check type to be t, run P
out$(c, N); P$	send message N on c, run P
if M then P else Q	conditional
let $x = M$ in P else Q	pattern matching and conditional
$R(M_1, \ldots, M_k)$	macro application

simplification: 0, **else** can often be omitted

Overview

Forward Secrecy

secrecy

up to now	black & white view: value either secret or not
reality	privacy time-dependent — today's RSA keys secure 20 years from now?

problematic scenario

2021	Alice publishes k_A
2041	Bob computes $\hat{k}_A$
	Bob computes $\mathbf{sig}_{k_A}(m)$, m dated to 2019

2021	Alice and Bob use k_A and k_B to compute k_{AB}
2021	Alice sends $\mathbf{enc}^{s}_{k_{AB}}(\mathit{secret})$
2041	Charlie computes $\hat{k}_A$ and $\hat{k}_B$, from this k_{AB} and secret

forward security

security against **later** key breaks

Forward Secrecy in Handshake-Protocol

protocol (fixed)

1. $A \rightarrow B \quad k_A$
2. $B \rightarrow A \quad \mathrm{enc}^{a}_{k_A}\left(\mathrm{sig}_{k_B}\left([k_A, k_B, k_{AB}]\right)\right)$
3. $A \rightarrow B \quad \mathrm{enc}^{s}_{k_{AB}}(\mathrm{FAIL})$.

asymmetric situation

- always: if $\hat{k}_A$ and $\hat{k}_B$ known later, protocol run can be reconstructed
- if B server: probability that $\hat{k}_B$ gets known higher (more attractive goal)
- FAIL must stay secret if $\hat{k}_B$ gets known later

analysis

- if $\hat{k}_A$ and $\hat{k}_B$ secret: FAIL secret
- if $\hat{k}_A$ known “later:” FAIL derivable
- if $\hat{k}_B$ known: protocol insecure
- if $\hat{k}_B$ known “later:” FAIL not derivable

conclusion

“cost” of revealing keys depends on usage in protocol

Forward Secrecy in ProVerif I

modeling

- split protocol into phases $0, \dots, n-1$
- keyword **phase**
- models “global time”
- leaking of $\hat{k}_B$ in phase i: **phase** i; **out**(c, $\hat{k}_B$)

Forward Secrecy in ProVerif II

fixed handshake protocol (informal)

1. $A \rightarrow B \quad k_A$
2. $B \rightarrow A \quad \mathrm{enc}^{a}_{k_A}\left(\mathrm{sig}_{k_B}\left([k_A, k_B, k_{AB}]\right)\right)$
3. $A \rightarrow B \quad \mathrm{enc}^{s}_{k_{AB}}\left(\mathrm{FAIL}\right).$

protocol: global & Alice

```
free c:channel.
free FAIL:bitstring [private].
query attacker(FAIL).
let clientA(pkA:pkey,skA:skey,pkB:spkey)=
    out (c,pkA);
    in (c,x:bitstring);
    let y = adec(x,skA) in
        let (=pkA,=pkB,k:key) = checksign(y,pkB) in
            out (c,senc(FAIL,k)).
```

note

pattern matching capabilities (cp. formal model)

Forward Secrecy in ProVerif III

protocol (informal)

1. $A \to B \quad k_A$
2. $B \to A \quad \mathsf{enc}^{\mathsf{a}}_{k_A}\left(\mathsf{sig}_{k_B}([k_A, k_B, k_{AB}])\right)$
3. $A \to B \quad \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(\mathsf{FAIL})$.

protocol: Bob

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

Forward Secrecy in ProVerif IV

protocol (informal)

1. $A \rightarrow B \quad k_A$
2. $B \rightarrow A \quad \mathsf{enc}^{\mathsf{a}}_{k_A}\left(\mathsf{sig}_{k_B}\left([k_A, k_B, k_{AB}]\right)\right)$
3. $A \rightarrow B \quad \mathsf{enc}^{\mathsf{s}}_{k_{AB}}\left(\mathsf{FAIL}\right).$

protocol: main process

```
process
    new skA:skey;
    new skB:sskey;
    let pkA = pk(skA) in out(c,pkA);
    let pkB = spk(skB) in out(c,pkB);
    ( (!clientA(pkA,skA,pkB)) | (!serverB(pkB,skB,pkA)) | phase 1; out(c, skB) )
```

ProVerif Script: Forward Secrecy for Handshake Protocol

command line tool

```
$ proverif 2021_01_12_lecture_09/04_forward_secrecy.pv
$ proverif -graph targetDir 2021_01_12_lecture_09/04_forward_secrecy.pv
$ proverif -html targetDir 2021_01_12_lecture_09/04_forward_secrecy.pv
```

live demo

Forward Secrecy and phase in ProVerif

modeling global time

abstract, divided in "phases"

0. all keys secret
1. $\hat{k}_A$ gets known
2. $\hat{k}_B$ gets known
3. ...

real-life security

How long will an RSA-key remain secret?

verifiable properties

- " if k_{AB} not used after phase 0, publication of $\hat{k}_A$ does not lead to insecurity"
- "protocol only insecure if both $\hat{k}_A$ and $\hat{k}_B$ are known before ..."
- ...

Perfect Forward Secrecy

issue

- Alice and Bob want to perform key exchange
- resulting key k must remain secret — **even if involved private keys get compromised**
- no possibility to encrypt k (private keys compromised eventually)
- can use PKI only for signatures: public authenticated channel

approach

- cannot send k in exchanged messages
- can only send “parts” of k
- only Alice and Bob can compute k from “parts”

impossible?

seems impossible — *at our level of abstraction*

consequence

treat outside of / enhance model

Diffie Hellman Key Exchange

task

- Alice and Bob agree on a secret key
- use public (authenticated) channel

protocol

A, B			choose public values g, p
A			chooses secret value a
B			chooses secret value b
A	$\rightarrow$	B	g^a
B	$\rightarrow$	A	g^b
A			compute $(g^b)^a = g^{ab}$
B			compute $(g^a)^b = g^{ab}$
A, B			use g^{ab} as key

security

- can compute a from g, g^a: discrete logarithm
- choose structure where logarithm is hard

discrete logarithm

- structure: $\mathbb{Z}_p$ for prime p
- g generator of $\mathbb{Z}_p^*$
- DH assumption implies: logarithm computationally infeasable in $\mathbb{Z}_p^*$

what have we gained?

adversary can still store g, g^a, g^b and solve logarithm 20 years later

Consequence

modeling

- analyzing perfect forward secrecy requires algebra outside our model
- many extensions of analysis approaches incorporating algebraic properties

references

- Whitfield Diffie and Martin E. Hellman. “New Directions in Cryptography”. In: IEEE Transactions on Information Theory IT-22.6 (1976), pp. 644–654
- Ralf Küsters and Tomasz Truderung. “Reducing protocol analysis with XOR to the XOR-free case in the horn theory based approach”. In: ACM Conference on Computer and Communications Security. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. ACM, 2008, pp. 129–138. ISBN: 978-1-59593-810-7
- Ralf Küsters and Tomasz Truderung. “Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation”. In: Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009. IEEE Computer Society, 2009, pp. 157–171. ISBN: 978-0-7695-3712-2. DOI: 10.1109/CSF.2009.17. URL: https://doi.org/10.1109/CSF.2009.17

Perfect Forward Secrecy Implementation

steps

1. generate and distribute private/public keys
2. perform Diffie Hellman key exchange
3. use obtained key k for communication
4. delete k

popular implementation

What's App (not verified)

- end-to-end encryption: only participants have keys

Overview

Know: DY model insufficient

modeling of knowledge

- only derivability (DY model)
- not: “knowledge about ciphertext”

example

Alice sends to Bob: $\mathsf{enc}^{\mathsf{a}}_{k_B}(t)$*, where* $t \in \{\mathsf{yes}, \mathsf{no}\}$
(requires randomized encryption)

cannot express

- protocol with “choices” for Alice
- attacker does not know whether Alice sent yes or no, but knows terms yes, no
- similar: voting, attacker knows parties

Indistinguishability

situation

- assumption: attacker does not know N_A or $\hat{k}_B$
- Alice sends $\mathsf{enc}^{\mathsf{a}}_{k_B}(N_A, t)$ with $t \in \{\mathsf{yes}, \mathsf{no}\}$
- want to model: attacker cannot distinguish possibilities “yes” and “no”

term level

- attacker can distinguish $\mathsf{enc}^{\mathsf{a}}_{k_B}(\mathit{yes})$ and $\mathsf{enc}^{\mathsf{a}}_{k_B}(\mathit{no})$
- attacker cannot distinguish $\mathsf{enc}^{\mathsf{a}}_{k_B}(N_A, \mathit{yes})$ and $\mathsf{enc}^{\mathsf{a}}_{k_B}(N_A, \mathit{no})$

Adversary Knowledge

approach

- what does the adversary “know?”
- what is the adversary’s “interface?”

abstraction level

distinguishability of terms

- $\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \mathsf{no})$ and $\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \mathsf{yes})$ should “look the same” for adversary
- in particular: the adversary does not “know” that the message is $\mathsf{enc}^{\mathsf{a}}_{k_A}(N_A, \mathsf{yes})$

adversary actions

- adversary can perform “term operations” to distinguish messages
- apply constructors, destructors from equational theory
- result interpreted modulo $\equiv_E$

Indistinguishability: Tests

definition; I-tests

for $I \subseteq \mathcal{T}$, an atomic I-test is a pair (M, M') of terms such that

- M and M' derivable from I (may contain E-destructors)
- in M and M' exactly one variable x occurs

definition: test semantics

message m satisfies test (M, M'), if $M[m/x] \equiv_E M'[m/x]$.

extension

Boolean combinations of tests: $\wedge$, $\vee$, $\neg$

definition: indistinguishability

messages m and m' I-indistinguishable if there is no I-test satisfied by exactly one of m and m'.

Indistinguishability

example

- terms: $t_1 = \mathsf{hash}(\text{yes})$, $t_2 = \mathsf{hash}(\text{no})$
- tests:
  - $(M_1, M_1') = (\mathsf{hash}(\text{yes}), x)$
  - $(M_2, M_2') = (\mathsf{hash}(\text{no}), x)$

intuition

t_1, t_2 distinguishable?

application

- $(M_1[t_1/x], M_1'[t_1/x]) = (\mathsf{hash}(\text{yes}), \mathsf{hash}(\text{yes}))$
- $(M_2[t_1/x], M_2'[t_1/x]) = (\mathsf{hash}(\text{no}), \mathsf{hash}(\text{yes}))$

consequence

terms distinguishable?

more examples

lecture notes: (randomized) encryption, randomized hash

Indistinguishability Examples

example 1

- $t_1 = \mathbf{hash}\,(\mathrm{yes})$, $t_2 = \mathbf{hash}\,(\mathrm{no})$
- tests:
  - $(M_1, M_1') = (\mathbf{hash}\,(\mathrm{yes})\,, x)$
  - $(M_2, M_2') = (\mathbf{hash}\,(\mathrm{no})\,, x)$
- application:
  - $(M_1[t_1/x], M_1'[t_1/x]) = (\mathbf{hash}\,(\mathrm{yes})\,, \mathbf{hash}\,(\mathrm{yes}))$
  - $(M_1[t_2/x], M_1'[t_2/x]) = (\mathbf{hash}\,(\mathrm{yes})\,, \mathbf{hash}\,(\mathrm{no}))$
  - $(M_2[t_1/x], M_2'[t_1/x]) = (\mathbf{hash}\,(\mathrm{no})\,, \mathbf{hash}\,(\mathrm{yes}))$
  - $(M_2[t_2/x], M_2'[t_2/x]) = (\mathbf{hash}\,(\mathrm{no})\,, \mathbf{hash}\,(\mathrm{no}))$

consequence

- t_1 satisfies first test but not second
- t_2 satisfies second test but not first
- so, t_1 and t_2 distinguishable

more examples

- analogously: $\mathbf{enc}^{\mathrm{a}}_{k_A}\,(\mathrm{yes})$ and $\mathbf{enc}^{\mathrm{a}}_{k_A}\,(\mathrm{no})$ distinguishable (if k_A known)
- $\mathbf{hash}\,([N_A, \mathrm{yes}])$ and $\mathbf{hash}\,([N_A, \mathrm{no}])$ indistinguishable (if N_A unknown)
- $\mathbf{enc}^{\mathrm{a}}_{k_A}\,([N_A, \mathrm{yes}])$ and $\mathbf{enc}^{\mathrm{a}}_{k_A}\,([N_A, \mathrm{no}])$ indistinguishable (if N_A, $\hat{k}_A$ not known)

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

Exercise

Task (indistinguishability)

Prove that Boolean combinations of I-tests are not necessary. Formally: Show that if messages m and m' are I-distinguishable, then they are distinguishable via an atomic test.

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 II)

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

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

Strong Secrecy

example protocol

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

yes/no situation

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

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

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)

Assume an equational theory E. For a set I of terms and a term t, we say t is E-derivable from I, if there is a term M built from E-constructors, E-deconstructors and elements from I with $M \equiv_E t$.

For example, let E model (deterministic) symmetric encryption and pairing, let $I = \{k_{AC}, \underbrace{\mathrm{enc}^{s}_{k_{AC}}(\mathrm{yes}, N_A)}_{=:u}\}$. Then $t = N_A$ is E-derivable from I via

$$M = \mathrm{proj}_2(\mathrm{dec}^{s}_{k_{AC}}(u)).$$

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 algorithm for the nonce derivation problem.)

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?
- can attacker distinguish N_A and N_B?

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_12_lecture_09/05_strong_secrecy.pv`
  example from slides
- `2021_01_12_lecture_09/06_strong_secrecy_deterministic_encryption.pv`
  strong secrecy analysis with deterministic encryption
- `2021_01_12_lecture_09/07_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_12_lecture_09/08_weak_secrecy.pv
$ proverif -graph targetDir 2021_01_12_lecture_09/08_weak_secrecy.pv
$ proverif -html targetDir 2021_01_12_lecture_09/08_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

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

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 $e_1(\ldots)$ ==> event $e_2(\ldots)$:
  - before e_1, event e_2 must have happened
  - quantifiers: lhs $\forall$, rhs $\exists$
- **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: 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: Handshare with Events

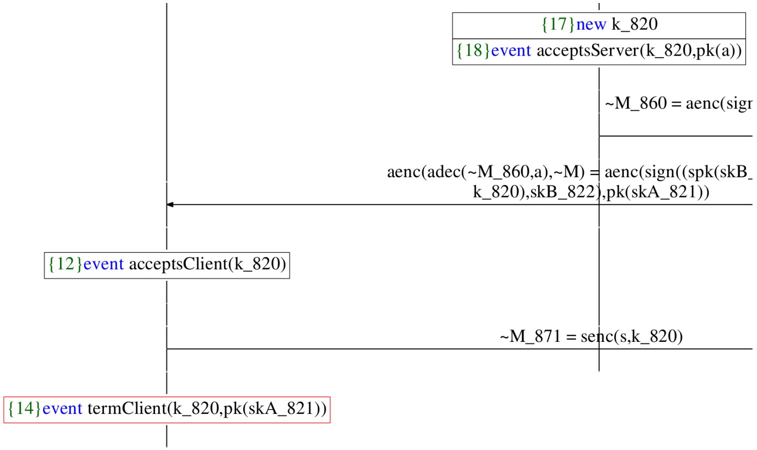
`2021_01_12_lecture_09/09_original_handshake_with_events.pv`

original (insecure) handshake protocol modelled with events

`2021_01_12_lecture_09/10_fixed_handshake_with_events.pv`

fixed handshake protocol modelled with events

ProVerif Analysis with Events


{17}new k_820
{18}event acceptsServer(k_820,pk(a))
~M_860 = aenc(sigr
aenc(adec(~M_860,a),~M) = aenc(sign((spk(skB_
k_820),skB_822),pk(skA_821))
{12}event acceptsClient(k_820)
~M_871 = senc(s,k_820)
{14}event termClient(k_820,pk(skA_821))

```
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(...) $\Rightarrow$ 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
  in the theoretical model 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;* ***inj-event(termServer(x))*** $\Rightarrow$ ***inj-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.

Exercise

Task (injective events)

Use inj-event to analyze the security of a protocol for authenticated message exchange exchange. Proceed as follows:

1. Consider a naive protocol for message exchange that only verifies whether messages come from the intended source.
2. Use event to verify the authentication property.
3. Which attacks are possible on the protocol? Use inj-event to discover the attack in ProVerif.
4. Correct the protocol and verify its correctness in ProVerif.

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?

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?

ProVerif Script: Incompleteness Example

command line tool

```
$ proverif 2021_01_12_lecture_09/11_weak_secrecy.pv
$ proverif -graph targetDir 2021_01_12_lecture_09/11_weak_secrecy.pv
$ proverif -html targetDir 2021_01_12_lecture_09/11_weak_secrecy.pv
```

live demo

Overview

Dealing with Undecidability

seen

ProVerif (and any other tool) cannot analyze all protocols

questions for practice

Can we “rewrite” protocols so that analysis becomes possible?

2 examples

- tagging
- order dependency

no general “rule”

problem remains undecidable

Protocols with Tags I

example

$B \rightarrow A$	$\mathrm{enc}^{\mathrm{s}}_{k}(N_B)$
$A \rightarrow B$	$\mathrm{enc}^{\mathrm{s}}_{k}(f(N_B))$

modelling

Alice: $\mathrm{enc}^{\mathrm{s}}_{k}(x) \rightarrow \mathrm{enc}^{\mathrm{s}}_{k}(f(x))$

consequence

- init: $\mathit{Att}(\mathrm{enc}^{\mathrm{s}}_{k}(N_B))$
- rule: $\mathit{Att}(\mathrm{enc}^{\mathrm{s}}_{k}(x)) \rightarrow \mathit{Att}(\mathrm{enc}^{\mathrm{s}}_{k}(f(x)))$
- obtain $\mathit{Att}(\mathrm{enc}^{\mathrm{s}}_{k}(f(N_B)), \mathit{Att}(\mathrm{enc}^{\mathrm{s}}_{k}(f(f(N_B))), \mathit{Att}(\mathrm{enc}^{\mathrm{s}}_{k}(f(f(f(N_B)))), \ldots$

Protocols with Tags II

solution: tags

- tag: “unique” constant for every application of cryptographic primitives
- actual message is extended with tag

earlier example

$$\begin{array}{ll} B \to A & \mathrm{enc}^s_k([\mathrm{step}_1, N_B]) \\ A \to B & \mathrm{enc}^s_k([\mathrm{step}_2, f(N_B)]) \end{array}$$

why does this help?

consequences

- ProVerif terminates for tagged protocols (for “usual” primitives and properties)
- good design practice

but: does not allow verification of original protocol!

Incompleteness due to Ordering

new values in ProVerif

new X:bitstring;

- internally: X with parameters
- dependent on position in source code
- influences completeness of analysis

example

order_dependence.pv

ProVerif Script: Order Dependence

command line tool

```
$ proverif 2021_01_12_lecture_09/12_order_dependence.pv
$ proverif -graph targetDir 2021_01_12_lecture_09/12_order_dependence.pv
$ proverif -html targetDir 2021_01_12_lecture_09/12_order_dependence.pv
```

live demo

Overview

ProVerif Summary

seen

- ProVerif can analyse examples treated so far in lecture
- analysis of unbounded sessions “possible in practice”
- ProVerif features beyond formal model:
  - strong/weak secrecy, events
- limitations: seen only in contrived examples

caveats

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

limitations

relevant limitations for analysis of more complex protocols and properties exist, see voting

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?

References i

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

Bruno Blanchet. "Using Horn Clauses for Analyzing Security Protocols". In: Cryptology and Information Security Series 5 (2011), pp. 86–111.

Vincent Cheval, Véronique Cortier, and Mathieu Turuani. "A Little More Conversation, a Little Less Action, a Lot More Satisfaction: Global States in ProVerif". In: 31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018. IEEE Computer Society, 2018, pp. 344–358. ISBN: 978-1-5386-6680-7. DOI: 10.1109/CSF.2018.00032. URL: https://doi.org/10.1109/CSF.2018.00032.

Whitfield Diffie and Martin E. Hellman. "New Directions in Cryptography". In: IEEE Transactions on Information Theory IT-22.6 (1976), pp. 644–654.

References ii

- James Heather, Gavin Lowe, and Steve Schneider. “How to Prevent Type Flaw Attacks on Security Protocols”. In: Journal of Computer Security 11.2 (2003), pp. 217–244. URL: http://content.iospress.com/articles/journal-of-computer-security/jcs162.
- Ralf Küsters and Tomasz Truderung. “Reducing protocol analysis with XOR to the XOR-free case in the horn theory based approach”. In: ACM Conference on Computer and Communications Security. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. ACM, 2008, pp. 129–138. ISBN: 978-1-59593-810-7.
- Ralf Küsters and Tomasz Truderung. “Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation”. In: Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009. IEEE Computer Society, 2009, pp. 157–171. ISBN: 978-0-7695-3712-2. DOI: 10.1109/CSF.2009.17. URL: https://doi.org/10.1109/CSF.2009.17.

References iii

Robin Milner, Joachim Parrow, and David Walker. “A Calculus of Mobile Processes, I and II”. In: Inf. Comput. 100.1 (1992), pp. 1–77. DOI: 10.1016/0890-5401(92)90008-4. URL: http://dx.doi.org/10.1016/0890-5401(92)90008-4.

Davide Sangiorgi and David Walker. The Pi-Calculus - a theory of mobile processes. Cambridge University Press, 2001. ISBN: 978-0-521-78177-0.