Engineering Secure Software Systems

January 12, 2021: Protocol Analysis with ProVerif

Henning Schnoor

Institut für Informatik, Christian-Albrechts-Universität zu Kiel

Part I: Crypto Protocols

Overview

Overview

Exercise

Task (ProVerif example I)

Consider the following protocol:

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

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 example II)

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_12_lecture_09/03_randomized-encryption.pv
$ proverif -graph targetDir 2021_01_12_lecture_09/03_randomized-encryption.pv
$ proverif -html targetDir 2021_01_12_lecture_09/03_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

cryptographic models

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

What kind of model does ProVerif use?

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

Part I: Crypto Protocols

- Foundations
- Cryptography
- An Example and an Attack
- More Examples
- Formal Protocol Model
- Automatic Analysis: Theoretical Foundations
- Automatic Analysis: Undecidability

Automatic Analysis in Practice: ProVerif

- Equational Theories
- Randomized Encryption
- Typing
- Syntax: Pi-Calculus
- Forward Secrecy
- Strong Secrecy

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
$M \mid\mid N$	disjunction
$\textbf{not}(M)$	negation

ProVerif: Syntax

part 2: processes

$P, Q =$

$\mathsf{0}$	nothing
$P \mid Q$	parallel execution
$!P$	unbounded replication, $!P = P \mid !P$
$\mathsf{new}\ n : t; P$	value n of type t in process P
$\mathsf{in}(c, x : t); P$	store message from channel c in x, check type to be t, run P
$\mathsf{out}(c, N); P$	send message N on c, run P
$\mathsf{if}\ M\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional
$\mathsf{let}\ x = M\ \mathsf{in}\ P\ \mathsf{else}\ Q$	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 $\mathsf{sig}_{k_A}(m)$, m dated to 2019

2021 Alice and Bob use k_A and k_B to compute k_{AB}

2021 Alice sends $\mathsf{enc}^{\mathsf{s}}_{k_{AB}}(\textit{secret})$

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

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 $0, \ldots, n-1$
- 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. $A \to B \quad k_A$
2. $B \to A \quad \mathsf{enc}^{\mathsf{a}}_{k_A}\left(\mathsf{sig}_{k_B}\left([k_A, k_B, k_{AB}]\right)\right)$
3. $A \to B \quad \mathsf{enc}^{\mathsf{s}}_{k_{AB}}(\mathsf{FAIL})$.

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

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,” e.g.:

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 [DH76]

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

reference

Paul Rösler, Christian Mainka, and Jörg Schwenk. "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema". In: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018. IEEE, 2018, pp. 415–429

Github [29, 30]. Since neither WhatsApp nor Threema provide official open source implementations, our analysis of these protocols mainly bases on the traffic that was received by unofficial protocol implementations [15, 16]. The respective messages and operations were sent by the official applications running on different devices and transmitted via the official messenger servers.

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* $t \in \{\mathsf{yes}, \mathsf{no}\}$ *(with randomized encryption)*

cannot express

- attacker does not know whether Alice sent **yes** or **no** (but knows words **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 $M[m/x] \equiv_E M'[m/x]$.

definition: indistinguishability

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

Indistinguishability Examples

example

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

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 (see also lecture notes)

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