Isabelle

a vehicle for Internet research

Gergely Buday University of Sheffield
Foundations of Computation Research Group
g.buday@sheffield.ac.uk

Sketch of the talk

In general, mathematics is definitions – theorems – proofs

in twenty minutes, I will talk about definitions and theorems

at IETF 118 in Prague, I will talk about formal proofs

ask questions at any time! also in e-mail

The goal of formal verification

to deduce formulae to basic inference rules: partly

but also: to deeply understand our systems

Reading for this talk

Lawrence C. Paulson:
The Inductive Approach to Verifying Cryptographic Protocols
Journal of Computer Security 1998

Lawrence C. Paulson:
Inductive Analysis of the Internet Protocol TLS
ACM Transactions on Information and System Security 1999

You can find these at the UFMRG web page

Isabelle as a software

Download it from here: https://isabelle.in.tum.de/
there are mirrors for UK, US and Australia
works for Linux (Intel & ARM), Windows, MacOS

Minimum memory: 4GB, for large projects 16GB

Archive of Formal Proofs: https://www.isa-afp.org/

GUI: Isabelle/jEdit Isabelle/VSCode

Protocols: messages

```
Isabelle2022/src/HOL/Auth/Message.thy:
datatype
      msg = Agent  agent     — ‹Agent names›
          | Number nat       — ‹Ordinary integers, timestamps, ...›
          | Nonce  nat       — ‹Unguessable nonces›
          | Key    key       — ‹Crypto keys›
          | Hash   msg       — ‹Hashing›
          | MPair  msg msg   — ‹Compound messages›
          | Crypt  key msg   — ‹Encryption, public- or shared-key›

(* Examples are from Isabelle theory files *)
```

Agents and events

```
datatype  — ‹We allow any number of friendly agents›
  agent = Server | Friend nat | Spy
consts
  bad    :: "agent set"                          — ‹compromised agents›
specification (bad)
  Spy_in_bad     [iff]: "Spy ∈ bad"
  Server_not_bad [iff]: "Server ∉ bad"

datatype
  event = Says  agent agent msg
        | Gets  agent       msg
        | Notes agent       msg
```

Operators: parts, analz, synth

```
inductive_set parts :: "msg set ⇒ msg set" for H :: "msg set"
... | Body:        "Crypt K X ∈ parts H ==> X ∈ parts H"
```

magic eye: we see even encrypted things, but no keys

```
inductive_set analz :: "msg set ⇒ msg set" for H :: "msg set"
... | Decrypt [dest]:
"⟦Crypt K X ∈ analz H; Key(invKey K) ∈ analz H⟧ ⟹ X ∈ analz H"
```

we see only things that are public or those we have key for

```
inductive_set synth :: "msg set => msg set" for H :: "msg set"
... | Crypt : "[|X ∈ synth H;  Key(K) ∈ H|] ==> Crypt K X ∈ synth H"
```

we construct almost anything from a base set H, but no nonces or keys

Algebraic properties: knows/spies

```
Isabelle2022/src/HOL/Auth/Event.thy:

primrec knows :: "agent ⇒ event list ⇒ msg set"

abbreviation (input)
  spies  :: "event list ⇒ msg set" where
  "spies == knows Spy"
```

what an agent (especially the Spy) can learn from the traffic

TLS: what the Spy can send

We model the protocol as a set of traces/runs: event list set

```
inductive_set tls :: "event list set"
  where

 | Fake: – ‹The Spy may say anything he can say.  The sender field is correct,
          but agents don't use that information.›
         "[| evsf ∈ tls;  X ∈ synth (analz (spies evsf)) |]
          ==> Says Spy B X # evsf ∈ tls"
```

synth (analz (spies events)) = construct (analyze (sees from traffic)))

TLS: the rules

```
Isabelle2022/src/HOL/Auth/TLS.thy:

inductive_set tls :: "event list set"
  where
  Nil:  — ‹The initial, empty trace›
        "[] ∈ tls"


| Fake: — ‹The Spy may say anything he can say.
       The sender field is correct,but agents
            don't use that information.›
   "[| evsf ∈ tls;  X ∈ synth (analz (spies evsf)) |]
         ==> Says Spy B X # evsf ∈ tls"


 | SpyKeys: —

‹The spy may apply PRF and sessionK
 to available nonces›
```

| ClientHello:

| ServerHello:

| Certificate:

| ClientKeyExch:

| CertVerify:

| ClientFinished:

| ServerFinished:

| ClientAccepts:

| ServerAccepts:

| ClientResume:

| ServerResume:

| Oops: — ‹The most plausible compromise is of an old session key.›

TLS: certificate

```
definition certificate :: "[agent,key] ⇒ msg" where
    "certificate A KA == Crypt (priSK Server) {|Agent A, Key KA|}"

and

inductive_set tls :: "event list set"

 | Certificate:
        — ‹SERVER (7.4.2) or CLIENT (7.4.6) CERTIFICATE.›
        "evsC ∈ tls ==> Says B A (certificate B (pubK B)) # evsC  ∈  tls"
----------------------------------------------------------------------
proved theorem:

[| certificate B KB ∈ parts(spies evs); evs ∈ tls |] ⇒ pubK B = KB
```

TLS: CertVerify

Isabelle2022/src/HOL/Auth/TLS.thy:

```
 | CertVerify:
       — ‹The optional Certificate Verify (7.4.8) message contains the
         specific components listed in the security analysis, F.1.1.2.
         It adds the pre-master-secret, which is also essential!
         Checking the signature, which is the only use of A's certificate,
         assures B of A's presence›
        "[| evsCV ∈ tls;
            Says B' A {|Nonce NB, Number SID, Number PB|} ∈ set evsCV;
            Notes A {|Agent B, Nonce PMS|} ∈ set evsCV |]
        ==> Says A B (Crypt (priK A) (Hash{|Nonce NB, Agent B, Nonce PMS|}))
             # evsCV  ∈  tls"
```

A’s certificate is trusted

```
text‹B can check A's signature if he has received A's certificate.›
lemma TrustCertVerify_lemma:
     "[| X ∈ parts (spies evs);
         X = Crypt (priK A) (Hash{|nb, Agent B, pms|});
         evs ∈ tls;  A ∉ bad |]
      ==> Says A B X ∈ set evs"
```

It was indeed A who sent this certificate

The verification of CertVerify

```
apply (erule rev_mp, erule ssubst)
apply (erule tls.induct, force, simp_all, blast)
done
```

rev_mp, ssubst: basic inference rules

force, simp_all, blast: automated proof search

tls.induct: the induction rule for the protocol trace

How to use Isabelle?

An anonymous reviewer asked: how do I actually use Isabelle?

It is not possible to detail this in a short talk

but let me try: you write your model as definitions

and state your properties as theorems

use Nitpick to find bugs, write lemmas, and use Sledgehammer!

How to use Isabelle?

```
subsubsection‹Protocol goal: if B receives CertVerify, then A sent it›

text‹B can check A's signature if he has received A's certificate.›
lemma TrustCertVerify_lemma:
     "[| X ∈ parts (spies evs);
         X = Crypt (priK A) (Hash{|nb, Agent B, pms|});
         evs ∈ tls;  A ∉ bad |]
      ==> Says A B X ∈ set evs"
apply (erule rev_mp, erule ssubst)
apply (erule tls.induct, force, simp_all, blast)
done
```

☑ Proof state ☑ Auto update Update Search:

```
proof (prove)
goal (1 subgoal):
 1. evs ∈ tls ⟹
    A ∉ bad ⟹
    Crypt (priEK A) (Hash {|nb, Agent B, pms|}) ∈ parts (knows Spy evs) ⟶
    Says A B (Crypt (priEK A) (Hash {|nb, Agent B, pms|})) ∈ set evs
```

How to use Isabelle?

```
lemma TrustCertVerify_lemma:
     "[| X ∈ parts (spies evs);
         X = Crypt (priK A) (Hash{|nb, Agent B, pms|});
         evs ∈ tls;  A ∉ bad |]
      ==> Says A B X ∈ set evs"
apply (erule rev_mp, erule ssubst)
  apply (erule tls.induct)
```

☑ Proof state ☑ Auto update Update Search:

```
proof (prove)
goal (15 subgoals):
 1. A ∉ bad ⟹
    Crypt (priEK A) (Hash {|nb, Agent B, pms|}) ∈ parts (knows Spy []) ⟶
    Says A B (Crypt (priEK A) (Hash {|nb, Agent B, pms|})) ∈ set []
 2. ⋀evsf X Ba.
       A ∉ bad ⟹
       evsf ∈ tls ⟹
```

A total of 15 subgoals…

How to use Isabelle?

```
lemma TrustCertVerify_lemma:
     "[| X ∈ parts (spies evs);
         X = Crypt (priK A) (Hash{nb, Agent B, pms});
         evs ∈ tls;  A ∉ bad |]
      ==> Says A B X ∈ set evs"
apply (erule rev_mp, erule ssubst)
  apply (erule tls.induct)
  sledgehammer
(* , force, simp_all, blast) *)
done
```

☑ Proof state ☑ Auto update Update Search:

```
Sledgehammering...
e found a proof...
zipperposition found a proof...
vampire found a proof...
cvc4 found a proof...
zipperposition: Try this: using Crypt_notin_initState knows_Nil apply presburger (0.7 ms)
e: Try this: using Crypt_notin_initState knows_Nil apply presburger (8 ms)
vampire: Try this: using Crypt_notin_used_empty apply blast (0.7 ms)
cvc4: Try this: using Crypt_notin_initState knows_Nil apply presburger (1 ms)
QED
```

Further reading

Part I of the book Concrete Semantics

http://concrete-semantics.org/

is a good introduction on how to construct formal proofs

Computational security proofs

Paulson’s inductive method follows the Dolev-Yao model

crypto primitives are perfect building blocks

there is space for inspecting crypto building blocks

Basin, Lochbihler and Sefidgar: CryptHOL (Isabelle)

includes a functional language for expressing games

Finishing thoughts

Isabelle is a general tool:
transport protocols can be modelled as well

Tamarin has more automation for security protocols
they reported an order of magnitude of speedup vs Isabelle:
Isabelle has a toolset for developing proof automation: Eisbach

Write questions to me at any time: g.buday@sheffield.ac.uk

Questions?

Algebraic operators: parts

```
inductive_set
  parts :: "msg set ⇒ msg set"
  for H :: "msg set"
  where
    Inj [intro]: "X ∈ H ==> X ∈ parts H"
  | Fst:         "{|X,Y|} ∈ parts H ==> X ∈ parts H"
  | Snd:         "{|X,Y|} ∈ parts H ==> Y ∈ parts H"
  | Body:        "Crypt K X ∈ parts H ==> X ∈ parts H"
```

magic eye: we see even encrypted things, but no keys

Algebraic operators: analz

```
inductive_set
  analz :: "msg set ⇒ msg set"
  for H :: "msg set"
  where
    Inj [intro,simp]: "X ∈ H ==> X ∈ analz H"
  | Fst:     "{|X,Y|} ∈ analz H ==> X ∈ analz H"
  | Snd:     "{|X,Y|} ∈ analz H ==> Y ∈ analz H"
  | Decrypt [dest]:
       "⟦Crypt K X ∈ analz H; Key(invKey K) ∈ analz H⟧ ⟹ X ∈ analz H"
```

we see only things that are public or those we have key for

Algebraic operators: synth

```
inductive_set
  synth :: "msg set => msg set"
  for H :: "msg set"
  where
    Inj    [intro]:   "X ∈ H ==> X ∈ synth H"
  | Agent  [intro]:   "Agent agt ∈ synth H"
  | Number [intro]:   "Number n  ∈ synth H"
  | Hash   [intro]:   "X ∈ synth H ==> Hash X ∈ synth H"
  | MPair  [intro]:   "[|X ∈ synth H;  Y ∈ synth H|] ==> {|X,Y|} ∈ synth H"
  | Crypt  [intro]:   "[|X ∈ synth H;  Key(K) ∈ H|] ==> Crypt K X ∈ synth H"
```

we construct almost anything from a base set H, but no nonces or keys