User-Guided Verification of Security Protocols via Sound Animation

SEFM 2024

Kangfeng Ye, Roberto Metere, Poonam Yadav

November 7th, 2024

Outline

Formal verification of security protocols

Successful applications

Authentication (5G AKA [BDH+18]), Handover (5G), Privacy, Access control, TLS (HTTPS), Payment (EMV[1]), and many more ...

[1] Basin et al. Tamarin: Verification of Large-Scale, Real World, Cryptographic Protocols. IEEE Security and Privacy Magazine (2022)

Formal verification of security protocols

Benefits

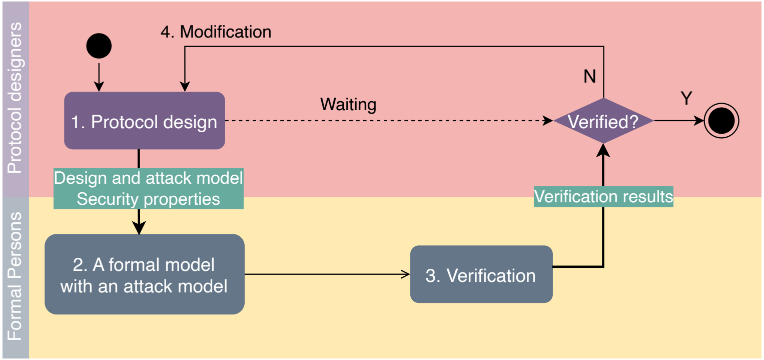
- Discovery of unknown vulnerabilities: Needham-Schroeder PKP [Low95], 5G AKA
- Identification of missing or weak assumptions: 5G AKA and handover
- Proposal of fixes or improvements to protocols: NSPK and 5G AKA
- Guarantee of correctness (EMV)

Formal verification of security protocols

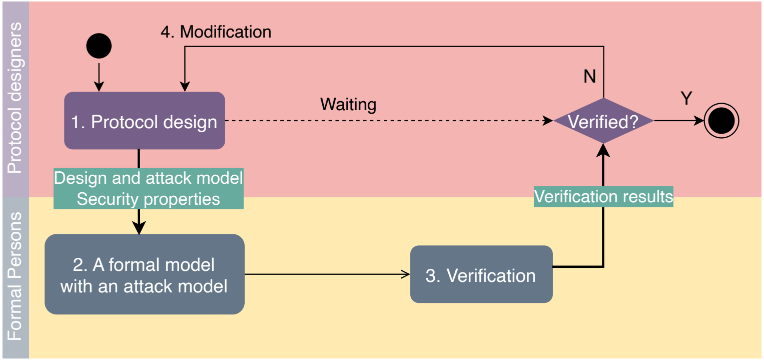
Notations and tools

- Process algebras: CSP (FDR) and Applied Pi Calculus (ProVerif)
- Inductive sets: Isabelle
- State-based: TLA^{+}-based AVISPA
- LTS and (multi-sets) rewriting rules: Maude-NPA, Tamarin-prover
- Model checking and theorem proving

Current practice and problems



Current practice and problems



Problems:

- Not accessible to designers
- Non-iterative, or slow-iteration

Accessible formal verification to engineers

Animation of a formal specification

- an executable computer program implemented in C/C++, Java, or Haskel etc.
- UI to interact with models for users
- Kazmierczak et al. [KWD98]: an animator for Z specifications
- Dutle et al. [DMNB15]: manually translate formally verified models to Java code
- Miller et al. [MBWN22]: an emulator (C++) to experiment with 5G AKE
- Boichut et al.'s SPAN [BGGH07]: an animation tool (OCaml and Tcl/Tk) for HLPSL, used in AVISPA.

Accessible formal verification to engineers

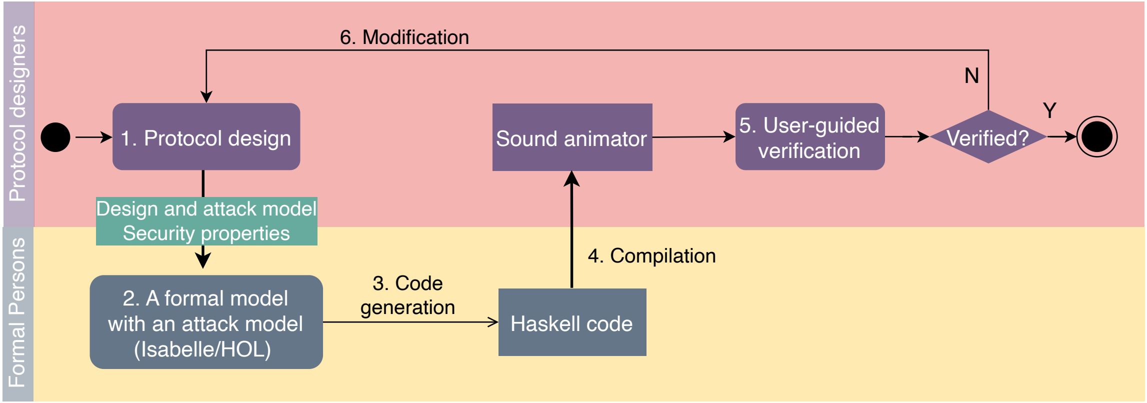
Problems: soundness is not guaranteed (implementations may have bugs), need efforts to develop code

Novel contributions

An interactive workflow supports

- automatically generated sound animators (Haskell),
- a lightweight and verified model checker: manual, automatic (exhaustive and random) exploration, reachability and feasibility checking
- a user-guided verification: automatic verification after manual exploration
- accessible to designers and fast iteration
- two case studies: Needham-Schroeder public key protocol (NSPK) and Diffie–Hellman key exchange protocol (DH)

Workflow to support user-guided verification



Interaction Trees[1] in Isabelle/HOL

Coinductive trees, with potentially infinite breadth and depth, used to represent the ways a process communicates with its environment and evolves over time.

Executable denotational semantics[2], originally mechanised in Coq

Mechanisation in Isabelle/HOL[3]

```
codatatype ('e, 'r) itree =
  Ret 'r |                              - ‹ Terminate, returning a value ›
  Sil "('e, 'r) itree" |                            - ‹ Invisible event ›
  Vis "'e ⇸ ('e, 'r) itree"          - ‹ Visible events and continuations ›
```

[1] Xia et al. 2019. Interaction trees: representing recursive and impure programs in Coq. Proc. ACM Program. Lang. 4, POPL

[2] Xia, L.y.: Executable Denotational Semantics with Interaction Trees. PhD thesis, 2022

[3] Foster et al. Formally verified simulations of state-rich processes using interaction trees in Isabelle/HOL. CONCUR (2021)

ITree-based Communicating Sequential Processes (CSP)[1,2]

- Deterministic processes
- Stateful processes: $P \fatsemi Q$
- No nondeterministic operator
- Biased external choice and parallel composition (priority)
- Hide with priority
- Renaming with priority

[1] Foster et al. Formally verified simulations of state-rich processes using interaction trees in Isabelle/HOL. CONCUR (2021)

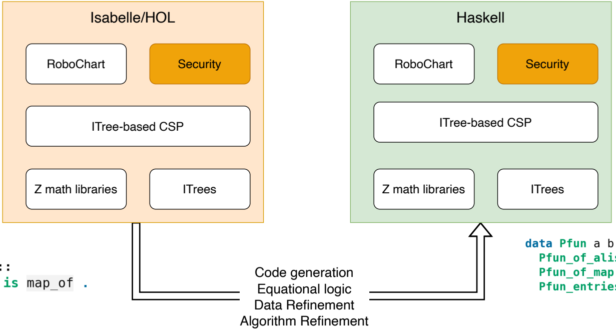
[2] Ye et al. Formally verified animation for RoboChart using interaction trees. JLAMP (2024)

Soundness: unbroken link from ITree-based CSP to Haskell

```
definition outp where
"outp c v =
  Vis
    (pfun_of_alist
      [(build⇘c⇙ v, Ret())])"
```

```
codatatype ('e, 'r) itree =
  Ret 'r |
  Sil "('e, 'r) itree" ("τ") |
  Vis "'e ⇸ ('e, 'r) itree"
```

```
lift_definition pfun_of_alist ::
  "('a × 'b) list ⇒ 'a ⇸ 'b" is map_of .
```




```
outp c v =
  Interaction_Trees.Vis
    (Interaction_Trees.Pfun_of_alist
      [(Prisms.prism_build c v,
        Interaction_Trees.Ret ())]);
```

```
data Itree a b =
  Ret b |
  Sil (Itree a b) |
  Vis (Pfun a (Itree a b));
```

```
data Pfun a b =
  Pfun_of_alist [(a, b)] |
  Pfun_of_map (a -> Maybe b) |
  Pfun_entries (Set.Set a) (a -> b);
```

```
lemma dom_pfun_alist [simp, code]:
  "pdom (pfun_of_alist xs) = set (map fst xs)"
  by (transfer, simp add: dom_map_of_conv_image_fst)
```

[1]Haftmann F., Nipkow, T.: Code Generation via Higher-Order Rewrite Systems. FLOPS (2010)

Needham-Schroeder Public-Key Protocol (NSPK)

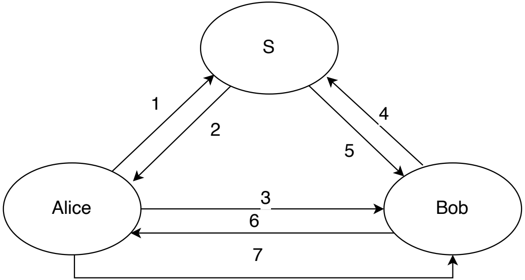
1978, Roger Needham and Michael Schroeder[1]

Establish secure communication between two parties over an insecure network

[1] Roger M. Needham and Michael D. Schroeder. 1978. Using encryption for authentication in large networks of computers. Commun. ACM 21, 12 (Dec. 1978)

Needham-Schroeder Public-Key Protocol (NSPK)

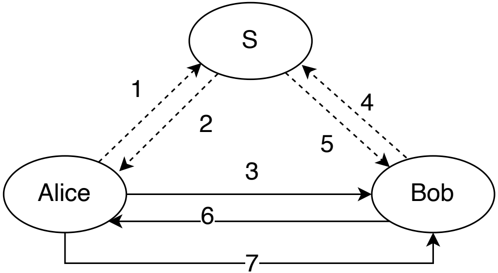
1. $A \rightarrow S : (A, B)$
2. $S \rightarrow A : \{\!|(pk(B), B)|\!\}^{s}_{sk(S)}$
3. $A \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$
4. $B \rightarrow S : (B, A)$
5. $S \rightarrow B : \{\!|(pk(A), A)|\!\}^{s}_{sk(S)}$
6. $B \rightarrow A : \{\!|(na, nb)|\!\}_{pk(A)}$
7. $A \rightarrow B : \{\!|nb|\!\}_{pk(B)}$



Messages 1,2,4,5 to retrieve public keys and Messages 3, 6, 7 for authentication

NSPK three-message version

1. $A \rightarrow S : (A, B)$
2. $S \rightarrow A : \{\!|(pk(B), B)|\!\}^{s}_{sk(S)}$
3. $A \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$: assume both A and B knows each other's public key
4. $B \rightarrow S : (B, A)$
5. $S \rightarrow B : \{\!|(pk(A), A)|\!\}^{s}_{sk(S)}$
6. $B \rightarrow A : \{\!|(na, nb)|\!\}_{pk(A)}$
7. $A \rightarrow B : \{\!|nb|\!\}_{pk(B)}$



NSPK three-message version

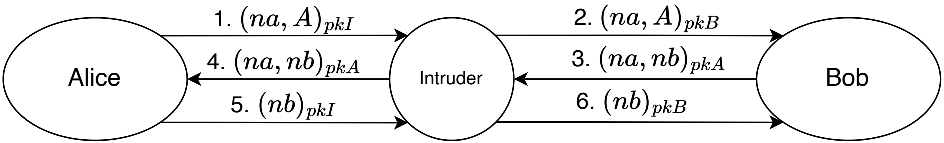
The man-in-the-middle-attack[1]: assume public keys are known to all principals

1. $A \rightarrow I : \{\!|(na, A)|\!\}_{pk(I)}$
2. $I(A) \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$: I pretends to be A
3. $B \rightarrow I(A) : \{\!|(na, nb)|\!\}_{pk(A)}$
4. $I \rightarrow A : \{\!|(na, nb)|\!\}_{pk(A)}$
5. $A \rightarrow I : \{\!|nb|\!\}_{pk(I)}$
6. $I(A) \rightarrow B : \{\!|nb|\!\}_{pk(B)}$: B believes that A has correctly established a session (shared secret (na, nb)) with him, but actually not

[1] Lowe, G. (1995). An attack on the Needham-Schroeder public-key authentication protocol. In IPL.

NSPK three-message version

The man-in-the-middle-attack[1]: assume public keys are known to all principals



[1] Lowe, G. (1995). An attack on the Needham-Schroeder public-key authentication protocol. In IPL.

Security Properties

Secrecy or confidentiality

The attacker cannot derive na and nb

Authenticity

Authentication for the initiator A:

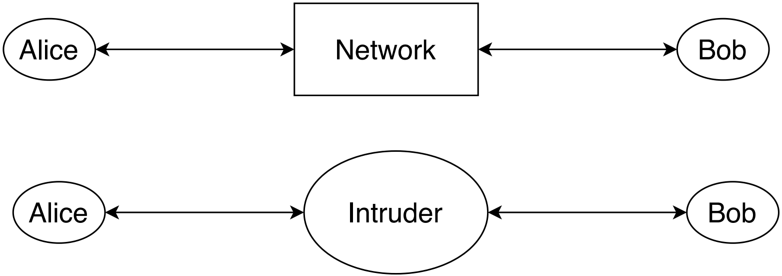
- A commits to a session with B only if B took part in the protocol run

Authentication for the responder B:

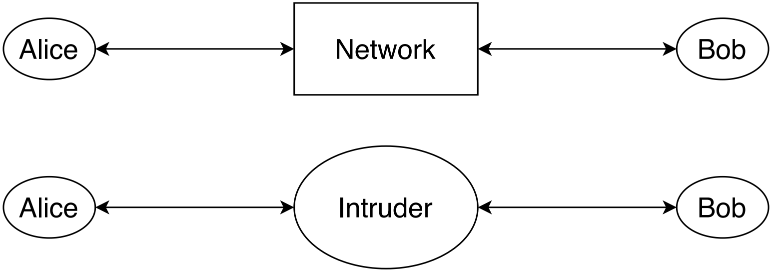
- B commits to a session with A only if A took part in the protocol run

Modelling of Protocols

Modelling of Protocols



Modelling of Protocols



PAlice $\|_A$ PIntruder $\|_B$ PBob

Dolev-Yao model[1]

The attacker controls the entire network and can

- intercept, delete, modify, delay, inject, and build new messages.

but limited by the cryptography and cannot

- decrypt messages without knowing the key. perfect blackbox cryptography

[1] D. Dolev and A. Yao, "On the security of public key protocols," in IEEE Transactions on Information Theory, vol. 29, no. 2, pp. 198-208, March 1983

Intruder message inference rules

$K \vdash_{\Downarrow} m$ - what the intruder can learn from a set of known messages K; Passive attacker

$K \vdash_{\Uparrow} m$ - what the intruder can build/fake from a set of known messages K

$\left\{m \mid \left(K \vdash_{\Downarrow} m\right)\right\} \vdash_{\Uparrow} m$: Active attacker

Name	Premise	Break down	Build up
Member	$m \in K$	$K \vdash_{\Downarrow} m$	$K \vdash_{\Uparrow} m$
Pairing	$m \in K \wedge m' \in K$		$K \vdash_{\Uparrow} (m, m')$
Unpairing	$(m, m') \in K$	$K \vdash_{\Downarrow} m$ and $K \vdash_{\Downarrow} m$	
Encryption/Sign	$m \in K \wedge k \in K$		$K \vdash_{\Uparrow} \{m\}_k$
Decryption/Verify	$\{m\}_k \in K \wedge k^{-1} \in K$	$K \vdash_{\Downarrow} m$	

Modelling of NSPK3 - type definitions

```
datatype dagent = Alice | Bob | Intruder
datatype dnonce = N dagent
datatype dpkey = PK dagent
datatype dskey = SK dagent
datatype dkey = Kp dpkey | Ks dskey
datatype dmsg = MAg (ma:dagent) | MNon (mn:dnonce) | MKp (mkp:dpkey) |
  MKs (mks:dskey) | MCmp (mc1:dmsg) (mc2:dmsg) |
  MEnc (mem:dmsg) (mek:dpkey)
```

Modelling of NSPK3 - type definitions

Used to specify and verify properties

```
datatype dsig = StartProt dagent dagent dnonce dnonce
  | EndProt dagent dagent dnonce dnonce
  | ClaimSecret (sag:dagent) (sn:dnonce) (sp: "ℙ dagent")
```

Modelling of NSPK3 - type definitions

Channels for communication between processes

```
datatype Chan =
  env :: dagent×dagent
  send, recv, hear, fake :: dagent×dagent×dmsg
  leak :: dmsg
  sig :: dsig
  terminate :: unit
```

Modelling of NSPK3

Protocol

1. $A \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$
2. $B \rightarrow A :$
   $\{\!|(na, nb)|\!\}_{pk(A)}$
3. $A \rightarrow B : \{\!|nb|\!\}_{pk(B)}$

$$
\begin{aligned}
& PAlice(A, na) \mathrel{\widehat=} \\
& \qquad env!A?B \rightarrow \\
& \qquad sig!ClaimSecrect!A!na!\{B\} \rightarrow \\
& \qquad send!\langle na, A\rangle_{pk(B)} \rightarrow \\
& \qquad recv?m : \{\langle na, nb\rangle_{pk(A)}\} \rightarrow \\
& \qquad sig!StartProt!A!B!na!nb \rightarrow \\
& \qquad send!nb_{pk(B)} \rightarrow \\
& \qquad sig!EndProt!A!B!na!nb \rightarrow \\
& \qquad terminate
\end{aligned}
$$

Modelling of NSPK3

Protocol

1. $A \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$
2. $B \rightarrow A :$
   $\{\!|(na, nb)|\!\}_{pk(A)}$
3. $A \rightarrow B : \{\!|nb|\!\}_{pk(B)}$

$$
\begin{aligned}
& PBob(B, nb) \mathrel{\widehat=} \\
& \quad recv?m : \{\langle na, A\rangle_{pk(B)}, \ldots\} \rightarrow \\
& \quad sig!ClaimSecrect!B!nb!\{A\} \rightarrow \\
& \quad sig!StartProt!B!A!na!nb \rightarrow \\
& \quad send!\langle na, nb\rangle_{pk(A)} \rightarrow \\
& \quad recv?m : \{nb_{pk(B)}\} \rightarrow \\
& \quad sig!EndProt!B!A!na!nb \rightarrow \\
& \quad terminate
\end{aligned}
$$

Modelling of NSPK3

Protocol

1. $A \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$
2. $B \rightarrow A : \{\!|(na, nb)|\!\}_{pk(A)}$
3. $A \rightarrow B : \{\!|nb|\!\}_{pk(B)}$

$$
\begin{array}{l}
PIntruder(I, ni, kn, ss) \mathrel{\widehat=} \\
\quad hear?m \rightarrow PIntruder(I, ni, breakm(kn \cup \{m\}), ss) \\
\quad \Box\ (\Box\ m : build_n(kn) \bullet fake!m \rightarrow PIntruder(I, ni, kn, ss)) \\
\quad \Box\ (\Box\ s : ss \cap kn \bullet leak!s \rightarrow PIntruder(I, ni, kn, ss)) \\
\quad \Box\ terminate \rightarrow Skip
\end{array}
$$

Modelling of NSPK3

NSPK3 protocol

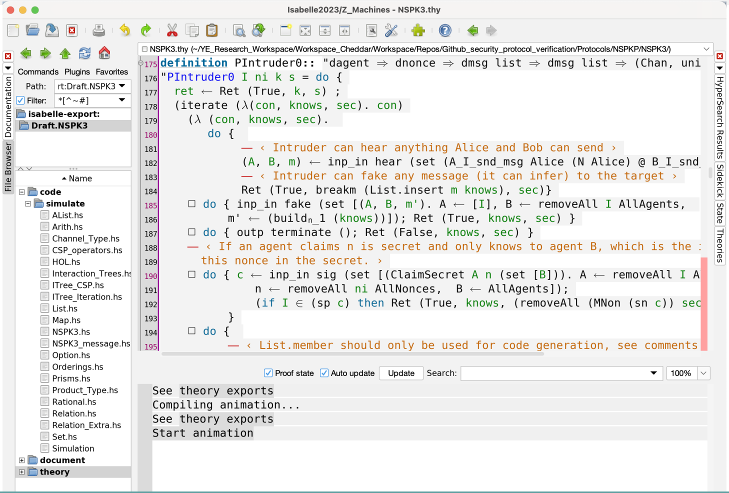
Protocol

1. $A \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$
2. $B \rightarrow A :$ $\{\!|(na, nb)|\!\}_{pk(A)}$
3. $A \rightarrow B : \{\!|nb|\!\}_{pk(B)}$

$$
\begin{aligned}
initknows &\mathrel{\widehat{=}} Agents \cup PublicKeys \cup \{ni, sk_I\} \\
secrets &\mathrel{\widehat{=}} \{na, nb\} \\
NSPK3 &\mathrel{\widehat{=}} \\
&\qquad \Big(PAlice(Alice, na) \parallel_{\{terminte\}} PBob(Bob, nb)\Big) \\
&\qquad \parallel_{\{\!|send, recv, terminate|\!\}} \\
&\qquad PIntruder(I, ni, initknows, secrets) \\
&\qquad\quad [hear \leftarrow send, fake \leftarrow recv]
\end{aligned}
$$

Demonstration

Automatic code generation from Isabelle/HOL



Demonstration

Manual exploration

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: 1
Env_C (Alice,Bob)
Events:
 (1) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (2) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;
 (3) Sig ClaimSecret Alice (N Alice) (Set [ Bob ]);

[Choose: 1-3]: 3
Sig_C (ClaimSecret Alice (N Alice) (Set [Bob]))
Events:
 (1) Send [Alice=>Intruder] {<N Alice, Alice>}_PK Bob;
 (2) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (3) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-3]: 1
Send_C (Alice,(Intruder,MEnc (MCmp (MNon (N Alice)) (MAg Alice)) (PK Bob)))
Events:
 (1) Recv [Bob<=Intruder] {<N Alice, Alice>}_PK Bob;
 (2) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (3) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;
```

Demonstration

Automatic modes

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: h
*** Usage ***
Auto n : Exhaustive search of traces up to n events or length;
Rand n : Random search of a trace up to n events or length;
AReach n %er1;er2;...%[#%em1;em2;...%] : Exhaustive search of traces up tc
RReach n %er1;er2;...%[#%em1;em2;...%] : Random search of a trace up to n
Feasible %event1;event2;...% : Check whether the specified sequence of eve
```

Demonstration

Termination

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: AReach 15 %Terminate%
AReach 15,  %Terminate%
Reachability by Auto: 15
  Events for reachability check: ["Terminate"]
  Events for monitor: []
...............................*** These events ["Terminate"] are reached! ***
Trace: [Env [Alice] Bob, Sig ClaimSecret Alice (N Alice) (Set [ Bob ]), Send [Alice=>Int
rot Bob Alice (N Alice) (N Bob), Send [Bob=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv
rot Alice Bob (N Alice) (N Bob), Recv [Bob<=Intruder] {N Bob}_PK Bob, Sig EndProt Bob Al

*** These events ["Terminate"] are reached! ***
Trace: [Env [Alice] Bob, Sig ClaimSecret Alice (N Alice) (Set [ Bob ]), Send [Alice=>Int
rot Bob Alice (N Alice) (N Bob), Send [Bob=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv
b<=Intruder] {N Bob}_PK Bob, Sig EndProt Bob Alice (N Alice) (N Bob), Sig EndProt Alice

*** These events ["Terminate"] are reached! ***
Trace: [Env [Alice] Bob, Sig ClaimSecret Alice (N Alice) (Set [ Bob ]), Send [Alice=>Int
rot Bob Alice (N Alice) (N Bob), Send [Bob=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv
b<=Intruder] {N Bob}_PK Bob, Sig EndProt Alice Bob (N Alice) (N Bob), Sig EndProt Bob Al
```

Demonstration

Secrecy

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: AReach 15 %Leak N Bob%
AReach 15,  %Leak N Bob%
Reachability by Auto: 15
  Events for reachability check: ["Leak N Bob"]
  Events for monitor: []
..........................................................................................
..........................................................................................
.......................................*** These events ["Leak N Bob"] are reached! ***
Trace: [Env [Alice] Intruder, Sig ClaimSecret Alice (N Alice) (Set [ Intruder ]), Send [A
lice=>Intruder] {<N Alice, Alice>}_PK Intruder, Recv [Bob<=Intruder] {<N Alice, Alice>}_P
K Bob, Sig ClaimSecret Bob (N Bob) (Set [ Alice ]), Sig StartProt Bob Alice (N Alice) (N
Bob), Send [Bob=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv [Alice<=Intruder] {<N Alice,
 N Bob>}_PK Alice, Sig StartProt Alice Intruder (N Alice) (N Bob), Send [Alice=>Intruder]
 {N Bob}_PK Intruder, ]

..........................................................................................
..........................................................................................
..........................................................................................
...................................*** Auto Reachability [15] Finished ***
```

Demonstration

User-guided verification

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: 1
Env_C (Alice,Bob)
Events:
 (1) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (2) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;
 (3) Sig ClaimSecret Alice (N Alice) (Set [ Bob ]);

[Choose: 1-3]: AReach 15 %Leak N Bob%
AReach 15,  %Leak N Bob%
Reachability by Auto: 15
  Events for reachability check: ["Leak N Bob"]
  Events for monitor: []
...........................................*** Auto Reachability [15] Finished ***
```

Demonstration

Authenticity for Alice

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: AReach 15 %Sig EndProt Alice Bob (N Alice) (N Bob)% # %Sig StartProt Bob A
lice (N Alice) (N Bob)%
AReach 15,  %Sig EndProt Alice Bob (N Alice) (N Bob)% # %Sig StartProt Bob Alice (N Alice
) (N Bob)%
Reachability by Auto: 15
  Events for reachability check: ["Sig EndProt Alice Bob (N Alice) (N Bob)"]
  Events for monitor: ["Sig StartProt Bob Alice (N Alice) (N Bob)"]
...............................*** These events ["Sig StartProt Bob Alice (N Alice) (N Bo
b)"] are monitored! ***
*** These events ["Sig EndProt Alice Bob (N Alice) (N Bob)"] are reached! ***
Trace: [Env [Alice] Bob, Sig ClaimSecret Alice (N Alice) (Set [ Bob ]), Send [Alice=>Intr
uder] {<N Alice, Alice>}_PK Bob, Recv [Bob<=Intruder] {<N Alice, Alice>}_PK Bob, Sig Clai
mSecret Bob (N Bob) (Set [ Alice ]), Sig StartProt Bob Alice (N Alice) (N Bob), Send [Bob
=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv [Alice<=Intruder] {<N Alice, N Bob>}_PK Ali
ce, Sig StartProt Alice Bob (N Alice) (N Bob), Send [Alice=>Intruder] {N Bob}_PK Bob, ]

......................................................................................
.......*** These events ["Sig StartProt Bob Alice (N Alice) (N Bob)"] are monitored! ***
..*** These events ["Sig StartProt Bob Alice (N Alice) (N Bob)"] are monitored! ***
.*** These events ["Sig StartProt Bob Alice (N Alice) (N Bob)"] are monitored! ***
..*** These events ["Sig StartProt Bob Alice (N Alice) (N Bob)"] are monitored! ***
.....*** These events ["Sig StartProt Bob Alice (N Alice) (N Bob)"] are monitored! ***
 *** These events ["Sig StartProt Bob Alice (N Alice) (N Bob)"] are monitored! ***
```

Demonstration

Authenticity for Bob

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: AReach 15 %Sig EndProt Bob Alice (N Alice) (N Bob)% # %Sig StartProt Alice Bob (N Alice) (N Bob)%
AReach 15,  %Sig EndProt Bob Alice (N Alice) (N Bob)% # %Sig StartProt Alice Bob (N Alice) (N Bob)%
Reachability by Auto: 15
  Events for reachability check: ["Sig EndProt Bob Alice (N Alice) (N Bob)"]
  Events for monitor: ["Sig StartProt Alice Bob (N Alice) (N Bob)"]
...............................*** These events ["Sig StartProt Alice Bob (N Alice) (N Bob)"] are monitored! ***
*** These events ["Sig EndProt Bob Alice (N Alice) (N Bob)"] are reached! ***
Trace: [Env [Alice] Bob, Sig ClaimSecret Alice (N Alice) (Set [ Bob ]), Send [Alice=>Intruder] {<N Alice, Alice>}_PK Bob, Recv [Bob<=Intruder
] {<N Alice, Alice>}_PK Bob, Sig ClaimSecret Bob (N Bob) (Set [ Alice ]), Sig StartProt Bob Alice (N Alice) (N Bob), Send [Bob=>Intruder] {<N
 Alice, N Bob>}_PK Alice, Recv [Alice<=Intruder] {<N Alice, N Bob>}_PK Alice, Sig StartProt Alice Bob (N Alice) (N Bob), Send [Alice=>Intrude
r] {N Bob}_PK Bob, Sig EndProt Alice Bob (N Alice) (N Bob), Recv [Bob<=Intruder] {N Bob}_PK Bob, ]

*** These events ["Sig EndProt Bob Alice (N Alice) (N Bob)"] are reached! ***
Trace: [Env [Alice] Bob, Sig ClaimSecret Alice (N Alice) (Set [ Bob ]), Send [Alice=>Intruder] {<N Alice, Alice>}_PK Bob, Recv [Bob<=Intruder
] {<N Alice, Alice>}_PK Bob, Sig ClaimSecret Bob (N Bob) (Set [ Alice ]), Sig StartProt Bob Alice (N Alice) (N Bob), Send [Bob=>Intruder] {<N
 Alice, N Bob>}_PK Alice, Recv [Alice<=Intruder] {<N Alice, N Bob>}_PK Alice, Sig StartProt Alice Bob (N Alice) (N Bob), Send [Alice=>Intrude
r] {N Bob}_PK Bob, Recv [Bob<=Intruder] {N Bob}_PK Bob, ]

..............................................................................................................................................
.............................................*** These events ["Sig EndProt Bob Alice (N Alice) (N Bob)"] are reached! ***
Trace: [Env [Alice] Intruder, Sig ClaimSecret Alice (N Alice) (Set [ Intruder ]), Send [Alice=>Intruder] {<N Alice, Alice>}_PK Intruder, Recv
 [Bob<=Intruder] {<N Alice, Alice>}_PK Bob, Sig ClaimSecret Bob (N Bob) (Set [ Alice ]), Sig StartProt Bob Alice (N Alice) (N Bob), Send [Bob
=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv [Alice<=Intruder] {<N Alice, N Bob>}_PK Alice, Sig StartProt Alice Intruder (N Alice) (N Bob),
Send [Alice=>Intruder] {N Bob}_PK Intruder, Leak N Bob, Sig EndProt Alice Intruder (N Alice) (N Bob), Recv [Bob<=Intruder] {N Bob}_PK Bob, ]

*** These events ["Sig EndProt Bob Alice (N Alice) (N Bob)"] are reached! ***
Trace: [Env [Alice] Intruder, Sig ClaimSecret Alice (N Alice) (Set [ Intruder ]), Send [Alice=>Intruder] {<N Alice, Alice>}_PK Intruder, Recv
 [Bob<=Intruder] {<N Alice, Alice>}_PK Bob, Sig ClaimSecret Bob (N Bob) (Set [ Alice ]), Sig StartProt Bob Alice (N Alice) (N Bob), Send [Bob
=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv [Alice<=Intruder] {<N Alice, N Bob>}_PK Alice, Sig StartProt Alice Intruder (N Alice) (N Bob),
Send [Alice=>Intruder] {N Bob}_PK Intruder, Leak N Bob, Recv [Bob<=Intruder] {N Bob}_PK Bob, ]

*** These events ["Sig EndProt Bob Alice (N Alice) (N Bob)"] are reached! ***
Trace: [Env [Alice] Intruder, Sig ClaimSecret Alice (N Alice) (Set [ Intruder ]), Send [Alice=>Intruder] {<N Alice, Alice>}_PK Intruder, Recv
 [Bob<=Intruder] {<N Alice, Alice>}_PK Bob, Sig ClaimSecret Bob (N Bob) (Set [ Alice ]), Sig StartProt Bob Alice (N Alice) (N Bob), Send [Bob
=>Intruder] {<N Alice, N Bob>}_PK Alice, Recv [Alice<=Intruder] {<N Alice, N Bob>}_PK Alice, Sig StartProt Alice Intruder (N Alice) (N Bob),
Send [Alice=>Intruder] {N Bob}_PK Intruder, Sig EndProt Alice Intruder (N Alice) (N Bob), Leak N Bob, Recv [Bob<=Intruder] {N Bob}_PK Bob, ]
```

Demonstration

Automatic feasibility check

```
Starting ITree Animation...
Events:
 (1) Env [Alice] Bob;
 (2) Env [Alice] Intruder;
 (3) Recv [Bob<=Intruder] {<N Intruder, Alice>}_PK Bob;
 (4) Recv [Bob<=Intruder] {<N Intruder, Intruder>}_PK Bob;

[Choose: 1-4]: Feasible %Env [Alice] Intruder; Sig ClaimSecret Alice (N Alice) (Set [ Intruder
 ]); Send [Alice=>Intruder] {<N Alice, Alice>}_PK Intruder; Recv [Bob<=Intruder] {<N Alice, Al
ice>}_PK Bob; Sig ClaimSecret Bob (N Bob) (Set [ Alice ]); Sig StartProt Bob Alice (N Alice) (
N Bob); Send [Bob=>Intruder] {<N Alice, N Bob>}_PK Alice; Recv [Alice<=Intruder] {<N Alice, N
Bob>}_PK Alice; Sig StartProt Alice Intruder (N Alice) (N Bob); Send [Alice=>Intruder] {N Bob}
_PK Intruder; Leak N Bob%
Feasible,  %Env [Alice] Intruder; Sig ClaimSecret Alice (N Alice) (Set [ Intruder ]); Send [Al
ice=>Intruder] {<N Alice, Alice>}_PK Intruder; Recv [Bob<=Intruder] {<N Alice, Alice>}_PK Bob;
 Sig ClaimSecret Bob (N Bob) (Set [ Alice ]); Sig StartProt Bob Alice (N Alice) (N Bob); Send
[Bob=>Intruder] {<N Alice, N Bob>}_PK Alice; Recv [Alice<=Intruder] {<N Alice, N Bob>}_PK Alic
e; Sig StartProt Alice Intruder (N Alice) (N Bob); Send [Alice=>Intruder] {N Bob}_PK Intruder;
 Leak N Bob%
Feasibility check the sequence of events: ["Env [Alice] Intruder","Sig ClaimSecret Alice (N Al
ice) (Set [ Intruder ])","Send [Alice=>Intruder] {<N Alice, Alice>}_PK Intruder","Recv [Bob<=I
ntruder] {<N Alice, Alice>}_PK Bob","Sig ClaimSecret Bob (N Bob) (Set [ Alice ])","Sig StartPr
ot Bob Alice (N Alice) (N Bob)","Send [Bob=>Intruder] {<N Alice, N Bob>}_PK Alice","Recv [Alic
e<=Intruder] {<N Alice, N Bob>}_PK Alice","Sig StartProt Alice Intruder (N Alice) (N Bob)","Se
nd [Alice=>Intruder] {N Bob}_PK Intruder","Leak N Bob"]
*** The specified trace is feasible ****
```

Evaluation

NSPK3: secrecy and authenticity for Bob are violated

NSLPK3 - Lowe's corrected version[1]: safe

1. $A \rightarrow B : \{\!|(na, A)|\!\}_{pk(B)}$
2. $B \rightarrow A : \{\!|(na, nb, B)|\!\}_{pk(A)}$
3. $A \rightarrow B : \{\!|nb|\!\}_{pk(B)}$

DH: used the animation to find the similar man-in-the-middle-attack (several counterexamples)

DH with digital signature: not subject to the same attack

[1] Gavin Lowe. 1996. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In (TACAs '96).

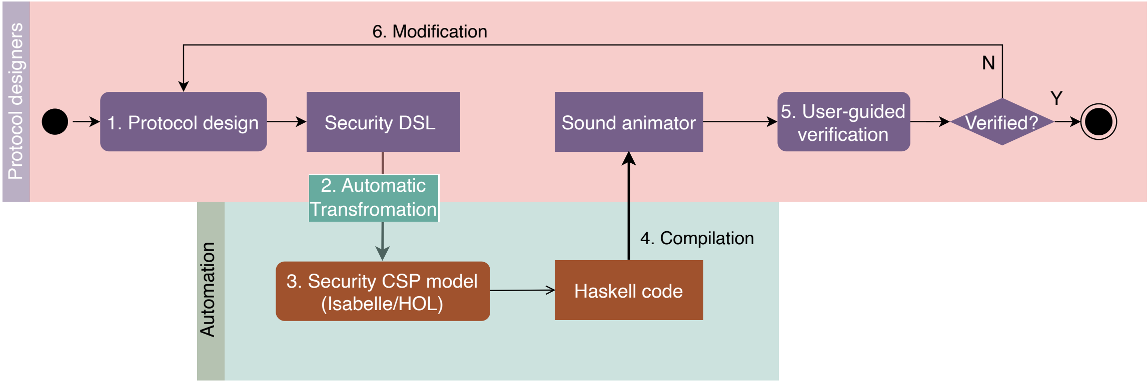
Conclusions

- Proposed an iterative workflow to support verification by designers through sound animation
- Enhanced animator to be a lightweight model checker: reachability and feasibility
- Verified two security protocols: NSPK and DH, and their corrected variants

Future work

- Generalisation of the framework for animation of security protocols
- Case studies
  - Physical layer security: watermarking and jamming, as security mechanism for IoT devices in edge-computing networks
  - 5G EAP-TLS, 5G AKA
  - Mesh Commissioning Protoco (MeshCoP) for the Thread network protocol
- Improvement of UI

Current not fully accessible → Fully automated workflow



Current limitations → Theorem Proving: (ITree-based) CSP and Circus

Limitations

- Enumerable and finite data types: agents, nonces, etc. Finite number of sessions
- Intensive space and computation time requirements to infer messages for the intruder

Animation can only verify a protocol with finite agents and bounded message inference in a finite number of interactions or steps. These are due to the executable nature of animation.

Current limitations → Theorem Proving: (ITree-based) CSP and Circus

Limitations

- Enumerable and finite data types: agents, nonces, etc. Finite number of sessions
- Intensive space and computation time requirements to infer messages for the intruder

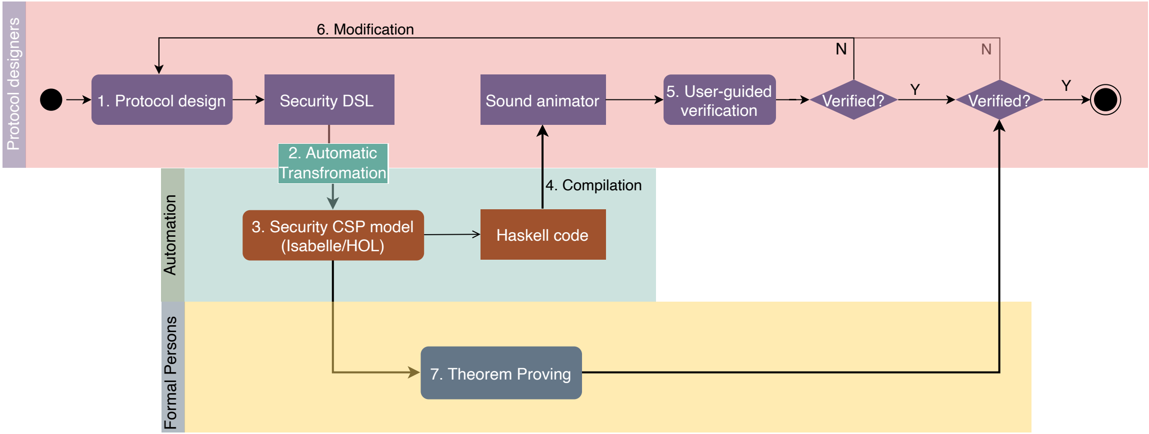
Animation can only verify a protocol with finite agents and bounded message inference in a finite number of interactions or steps. These are due to the executable nature of animation.

However, these are not the limitations of ITrees and ITree-based CSP.

Combined sound animation with theorem proving

- ITree-based CSP or Circus: could be for deductive verification
- General CSP or Circus: with nondeterministic and refinement

Current limitations → Theorem Proving: (ITree-based) CSP and Circus



Thank you!

https://www-users.york.ac.uk/~ky582/

Kangfeng **Ye**

University of York

kangfeng.ye@york.ac.uk

Roberto **Metere**

University of York

roberto.metere@york.ac.uk

Poonam **Yadav**

University of York

poonam.yadav@york.ac.uk

DataMod24: Towards Achieving Energy Efficiency and Service Availability in O-RAN via Formal Verification

David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler.
A Formal Analysis of 5G Authentication.
In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1383–1396, New York, NY, USA, 2018. Association for Computing Machinery.

Yohan Boichut, Thomas Genet, Yann Glouche, and Olivier Heen.
Using animation to improve formal specifications of security protocols.
In *2nd Conference on Security in Network Architectures and Information Systems (SARSSI 2007)*, pages 169–182, 2007.

Aaron M. Dutle, César A. Muñoz, Anthony J. Narkawicz, and Ricky W. Butler.
Software Validation via Model Animation, pages 92–108.
Springer International Publishing, 2015.

Edmund Kazmierczak, Michael Winikoff, and Philip W. Dart.
Verifying Model Oriented Specifications through Animation.
In *5th Asia-Pacific Software Engineering Conference (APSEC '98), 2-4 December 1998, Taipei, Taiwan, ROC*, pages 254–261. IEEE Computer Society, 1998.

Gavin Lowe.
An attack on the Needham-Schroeder public-key authentication protocol.
Information Processing Letters, 56(3):131–133, November 1995.

Rhys Miller, Ioana Boureanu, Stephan Wesemeyer, and Christopher J. P. Newton.
The 5G Key-Establishment Stack: In-Depth Formal Verification and Experimentation.
In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 237–251, New York, NY, USA, 2022. Association for Computing Machinery.