Rabbit: A Language to Model and Verify Data Flow in Networked Systems

T. Inaba[1], Y. Ishikawa[2], A. Igarashi[1], T. Sekiyama[2]

[1]Graduate School of Informatics
Kyoto University, Kyoto, Japan

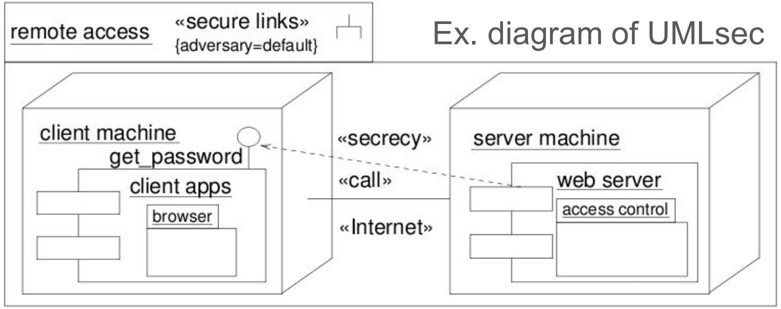
[2]Information Systems Architecture Science
Research Division
National Institute of Informatics, Tokyo, Japan

Background

- With increasing reliance on digital systems, cybersecurity is a growing concern.
- *Threat modeling* is an security-by-design approach where system designers
    - make abstract models of the target system and possible attackers,
    - identify security-critical data as *assets*, and
    - analyze security properties under the specified system model and the attacker model.
- In threat modeling, tracking <u>the data flows</u> of assets are crucial.
    - “where each asset originates, is stored temporarily, and finally reaches”

Existing work

- *UMLsec* (2002, Jürjens) has laid important ground work, with
  - an ability to express data flows,
  - formal semantics, and
  - plug-ins to verify security properties.



Ex. diagram of UMLsec

- **No low-level constructs such as processes, files, memory, and system calls.**

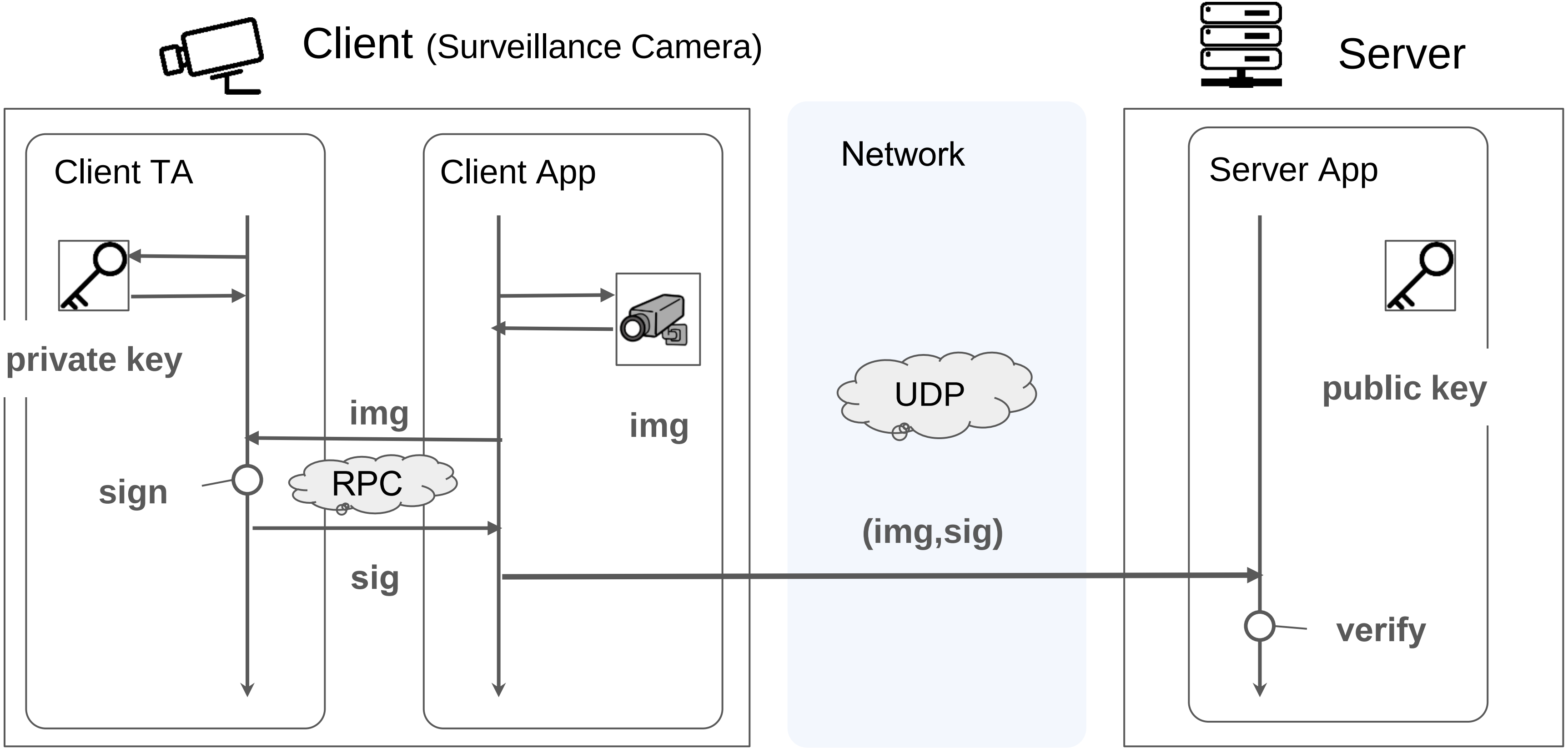
Rabbit language

- We develop *Rabbit*, a language to model data flows, with
  - low-level constructs (processes, files, communication channels, etc.) and
  - formal syntax and semantics.
- In this paper, we demonstrate
  - that Rabbit can model a case study (a client-server system),
  - that Rabbit can be systematically translated into an input of Tamarin (a model checker), and
  - that a verification experiment discovers a potential security weakness.

```
process Client(ch_net, ch_rpc) with client_t {
  …
   for i in range(1, 4) {
     let image = read(image_fd);
     let sig = invoke(ch_rpc, invoke_func, …);
     send(ch_net, (sig, image)) @ ImgSend(image);
   }
  …
```

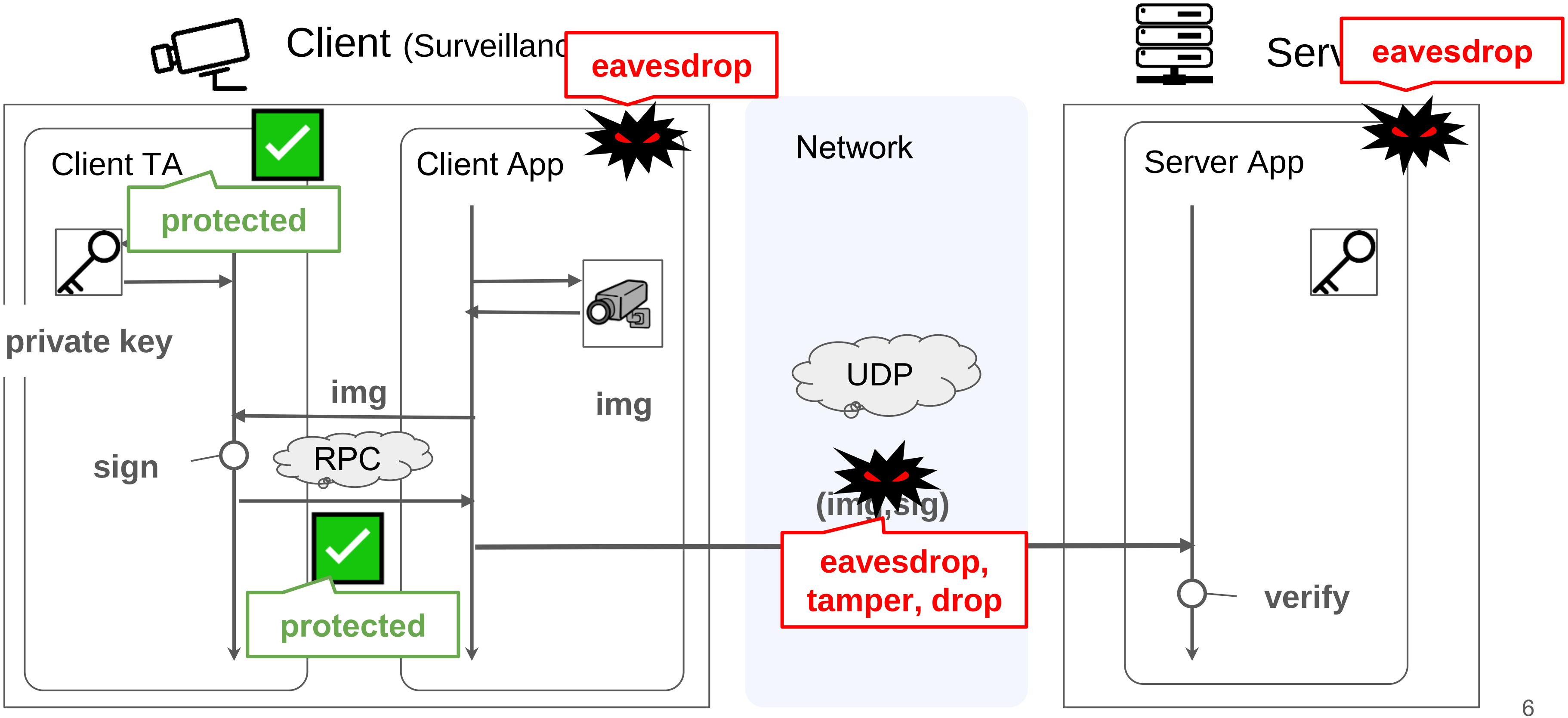
The Cam-Image system

TA ··· Trusted Application
(an application running in a secure module)



The Cam-Image system

TA ··· Trusted Application
(an application running in a secure module)



Client (Surveillanc

eavesdrop

Serv

eavesdrop

Client TA

protected

Client App

Network

Server App

private key

img

img

UDP

sign

RPC

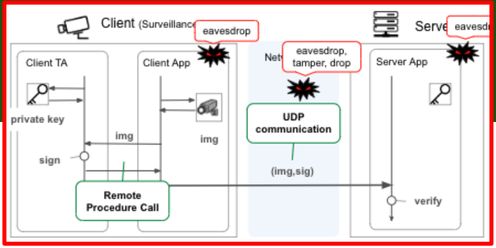
(img,sig)

eavesdrop, tamper, drop

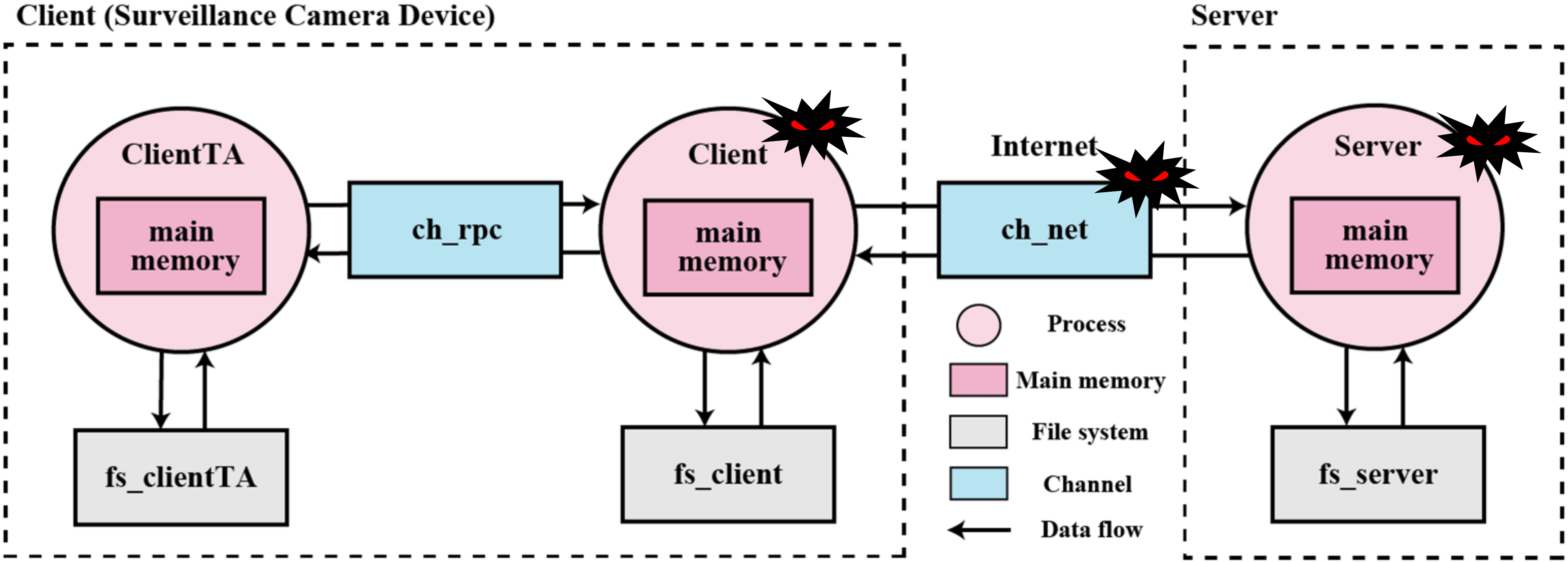
protected

verify

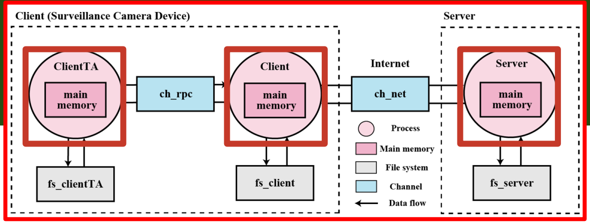
Rabbit model



Rabbit components consists of **processes, file systems, channels.**



Rabbit model



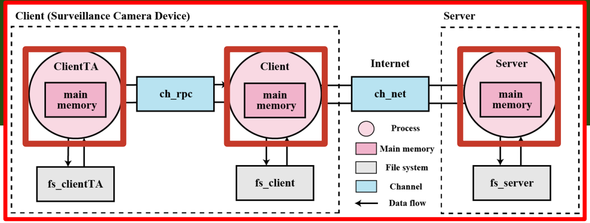
- Library functions like system calls in Linux
- Control Structures like for/if statements

```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

  main {
    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  }
}
```

```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

```
process Server(ch_net) with server_t {
  …
     let res = verify(p.fst, p.snd, pubkey);
     if (res) {
       skip @ ImgRecvValid(p.snd);
     }
  …
}
```

Rabbit model



- Library functions like system calls in Linux
- Control Structures like for/if statements

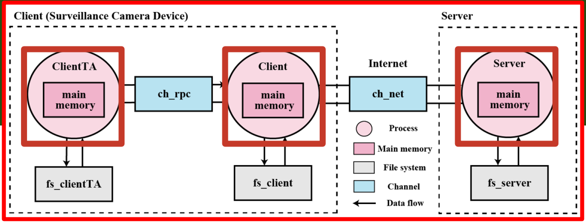
```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

  main {
    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  }
}
```

open system call

```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

```
process Server(ch_net) with server_t {
  …
      let res = verify(p.fst, p.snd, pubkey);
      if (res) {
        skip @ ImgRecvValid(p.snd);
      }
  …
}
```

Rabbit model



- Library functions like system calls in Linux
- Control Structures like for/if statements

```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  }
}
```

for statement

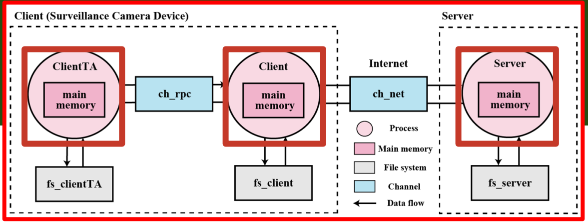
open system call

```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

```
process Server(ch_net) with server_t {
  …
      let res = verify(p.fst, p.snd, pubkey);
      if (res) {
        skip @ ImgRecvValid(p.snd);
      }

}
```

if statement

Rabbit model



- Library functions like system calls in Linux
- Control Structures like for/if statements

```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_ne(sig, image)) @ ImgSend(image);
    }
  }
}
```

for statement

open system call

rpc communication

```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

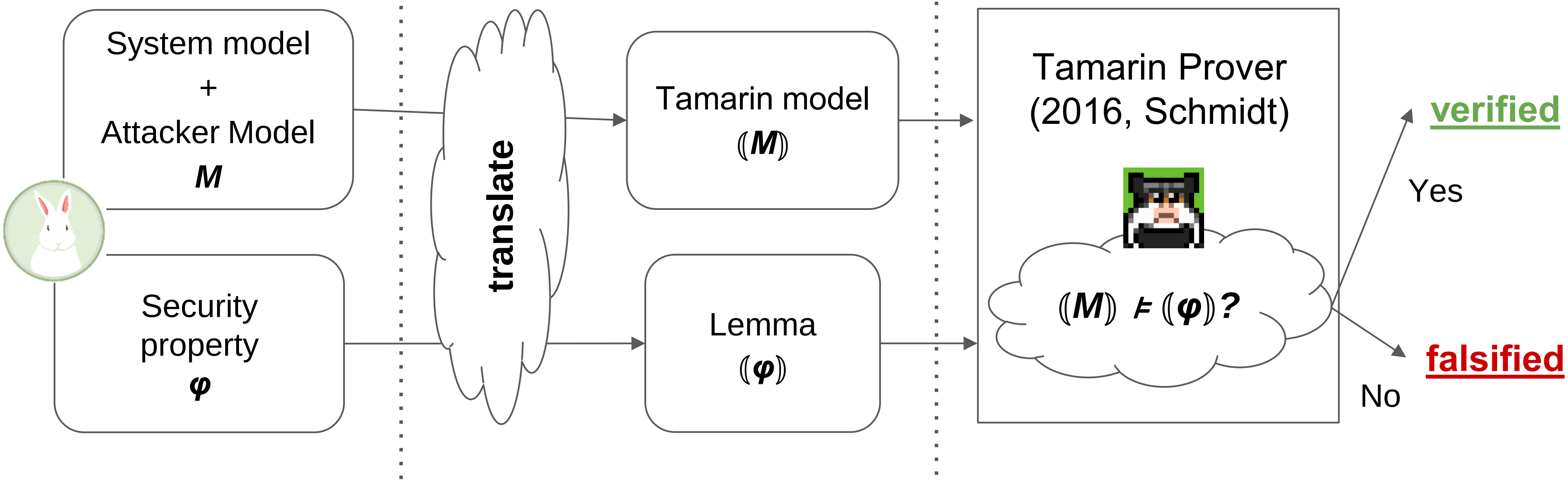
digital signature

```
process Server(ch_net) with server_t {
  …
     let res = verify(p.fst, p.snd, pubkey);
     if (res) {
       skip @ ImgRecvValid(p.snd);
     }

}
```

if statement

Translation & Verification

- Systematic translation
- Security properties are directly written in Rabbit



Verification of authenticity property

System Model

- The Cam-Image system (N=1,2,3 where N is the number of loop iterations).

= # of images sent

Attacker Model

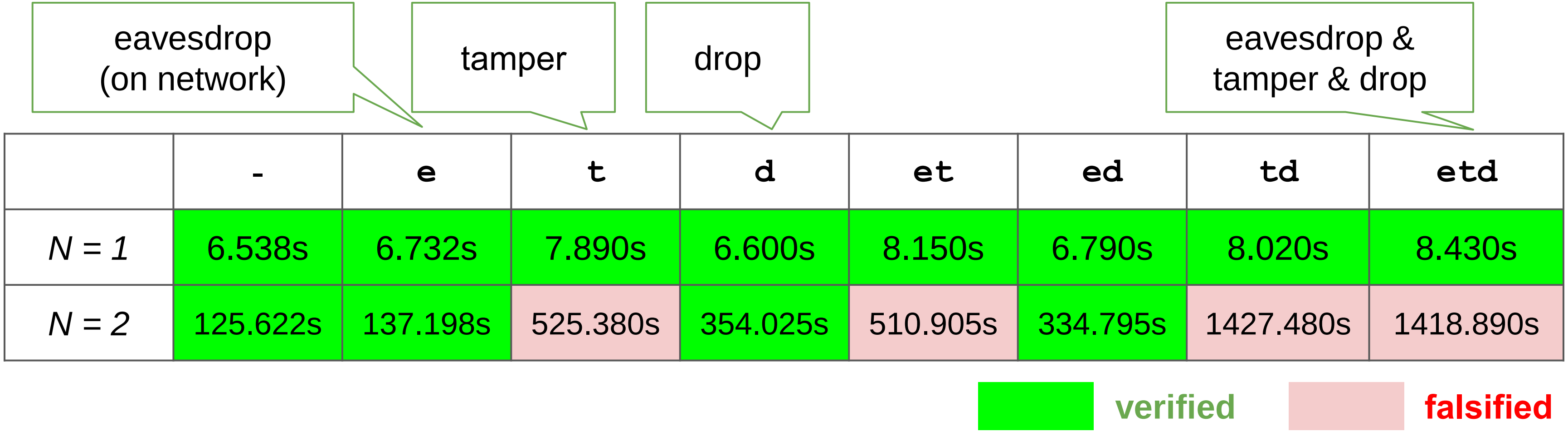
- The attacker capable of eavesdropping on the main memory of the client & server app.
- Parameter: Attacks on the network (eavesdrop, tamper, drop)

Property to verify

When the server verifies signature successfully, then the image is sent by the right client before (***corresponding property***).

```
lemma Authenticity :
  all-traces
  "All x #i . ImgRecvValid(x) @ #i ==> Ex #j . ImgSend(x) @ #j & #j < #i"
```

Verification results



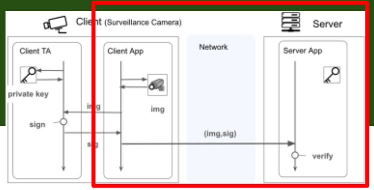
	`-`	`e`	`t`	`d`	`et`	`ed`	`td`	`etd`
N = *1*	6.538s	6.732s	7.890s	6.600s	8.150s	6.790s	8.020s	8.430s
N = *2*	125.622s	137.198s	525.380s	354.025s	510.905s	334.795s	1427.480s	1418.890s

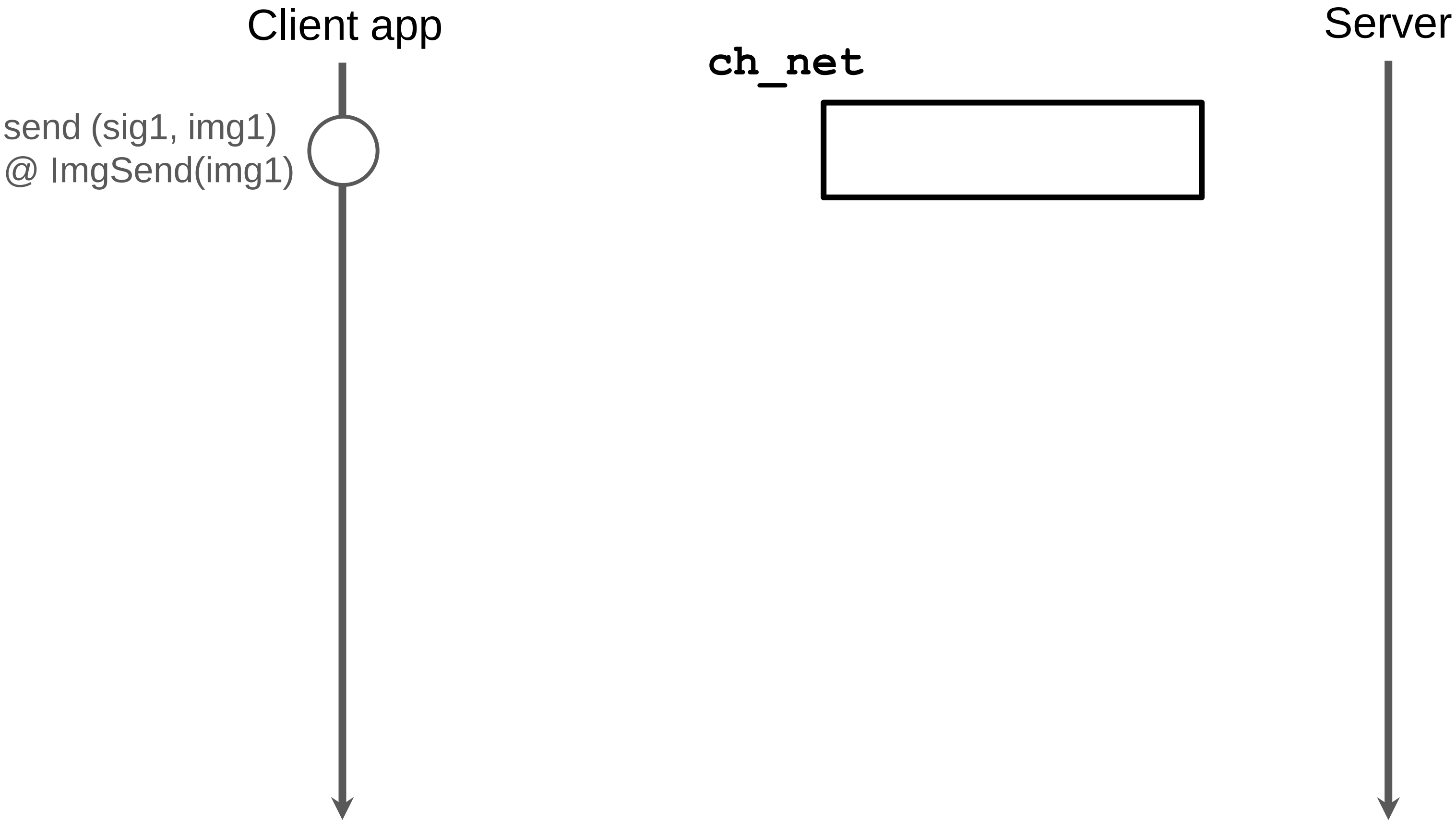
🟩 **verified** 🟥 **falsified**

Observation

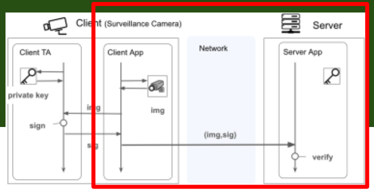
- **When the attacker is able to tamper messages, the corresponding property is falsified.**
- The increase of N largely affect the verification time.
- The verification time is large when attackers can perform active attacks.

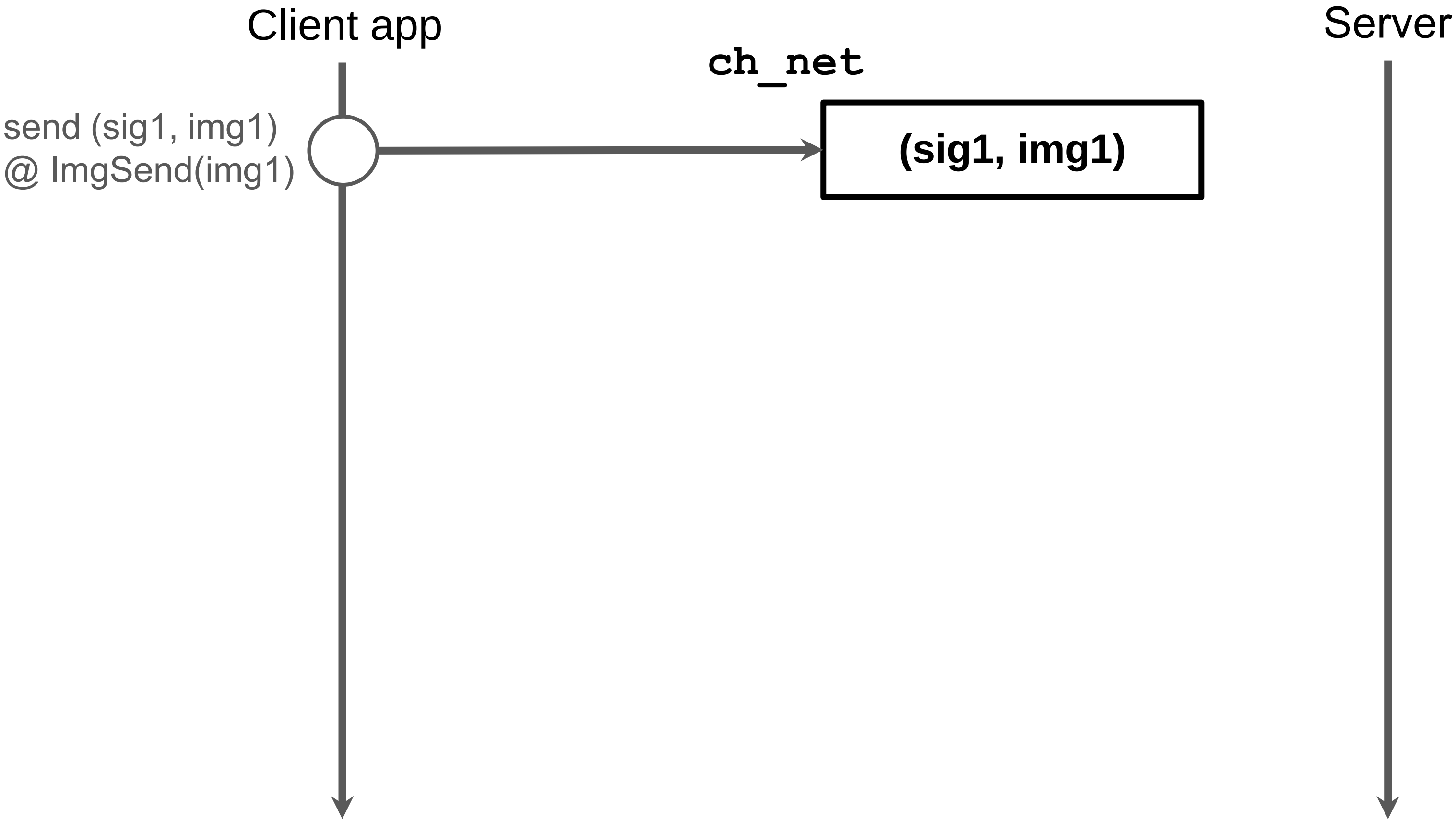
Automatically-found trace (falsified)





Automatically-found trace (falsified)





Client app

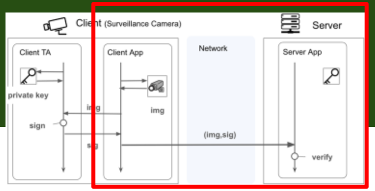
Server

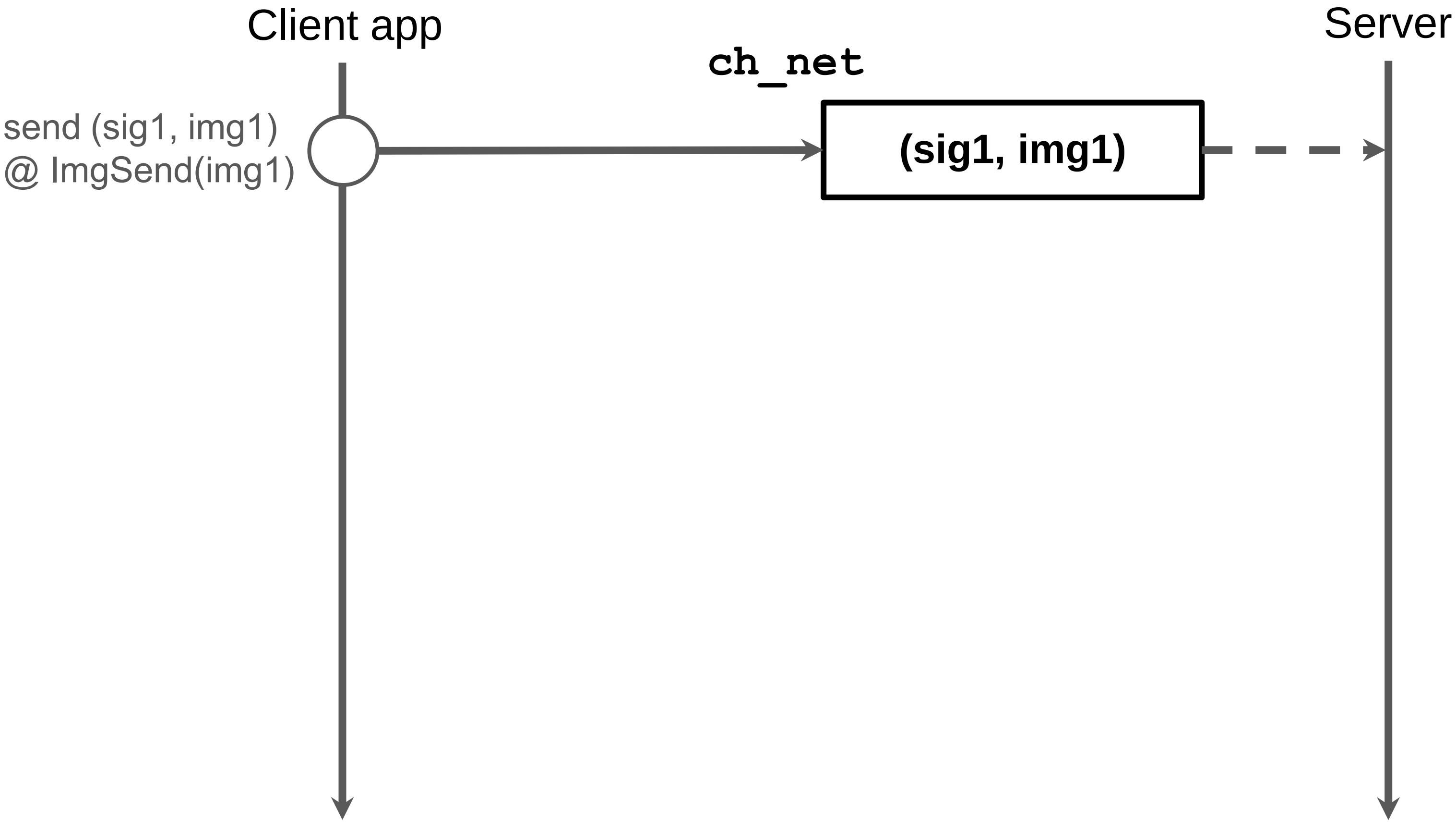
`ch_net`

send (sig1, img1)
@ ImgSend(img1)

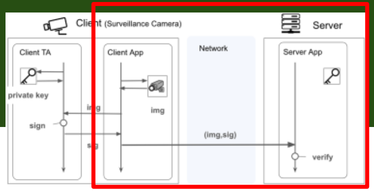
(sig1, img1)

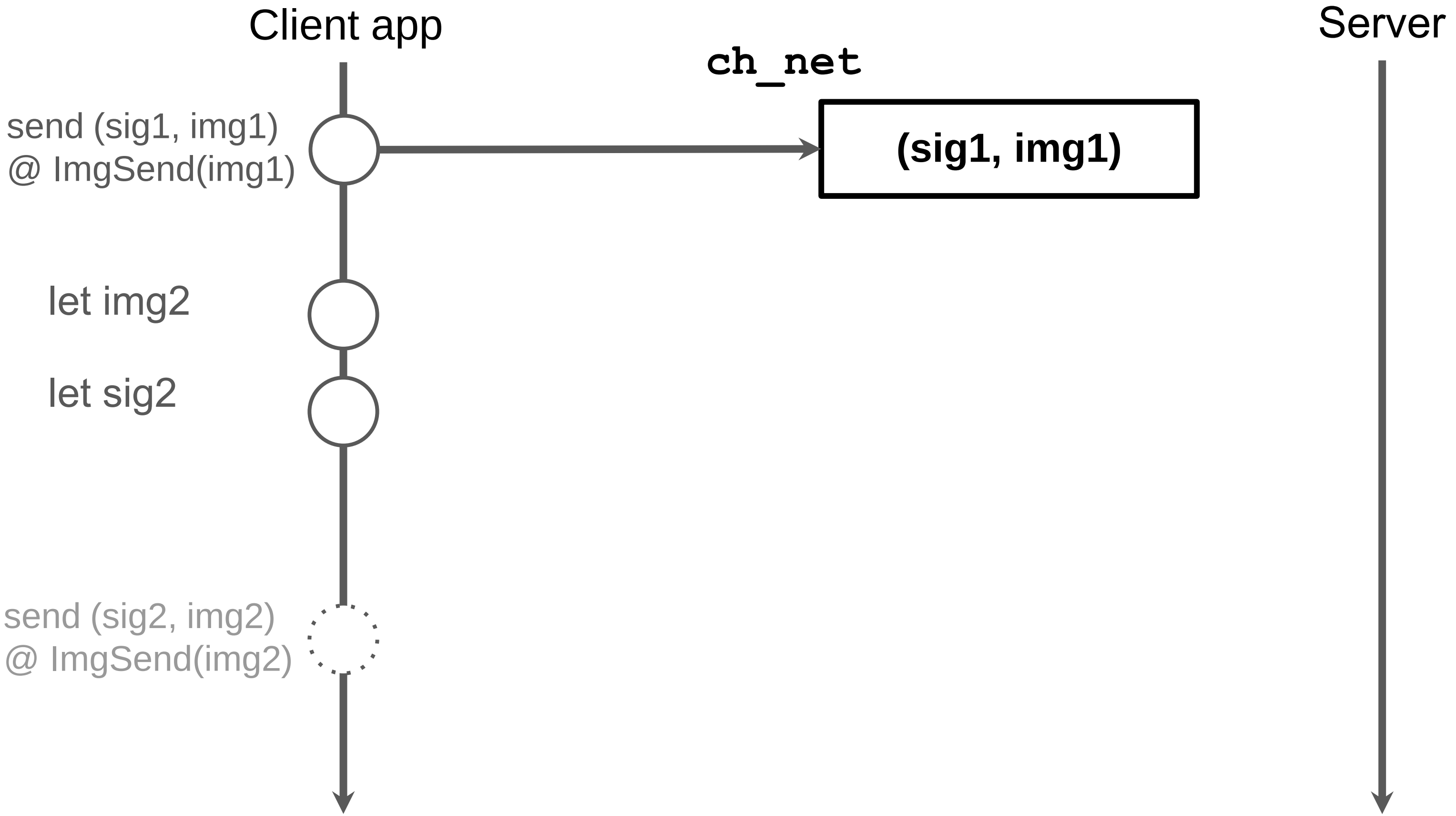
Automatically-found trace (falsified)





Automatically-found trace (falsified)





Client app

Server

`ch_net`

send (sig1, img1)
@ ImgSend(img1)

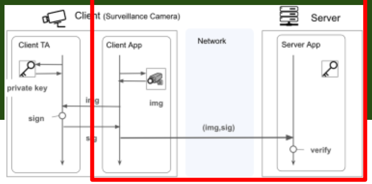
(sig1, img1)

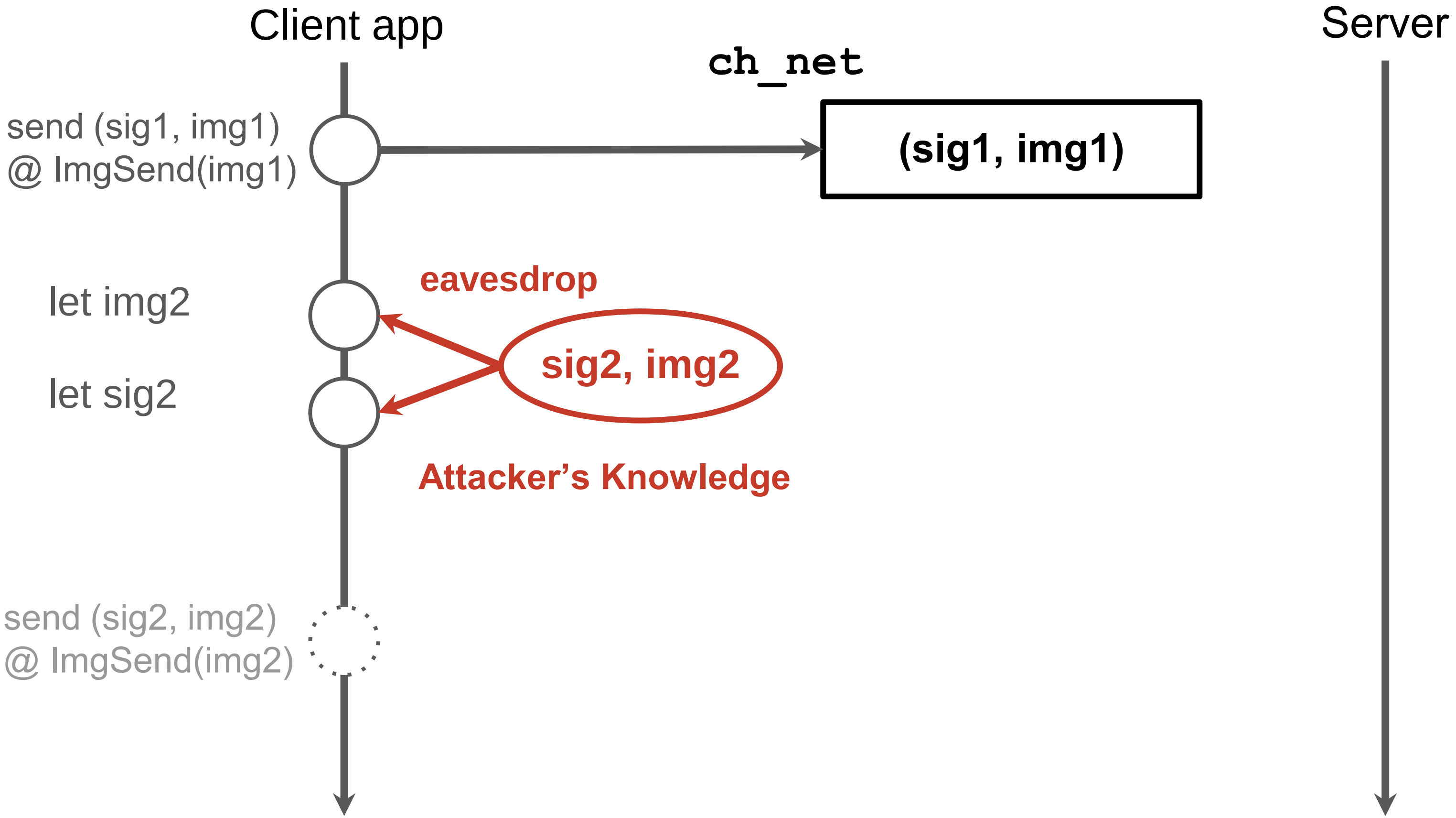
let img2

let sig2

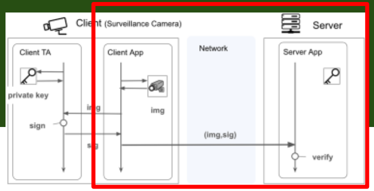
send (sig2, img2)
@ ImgSend(img2)

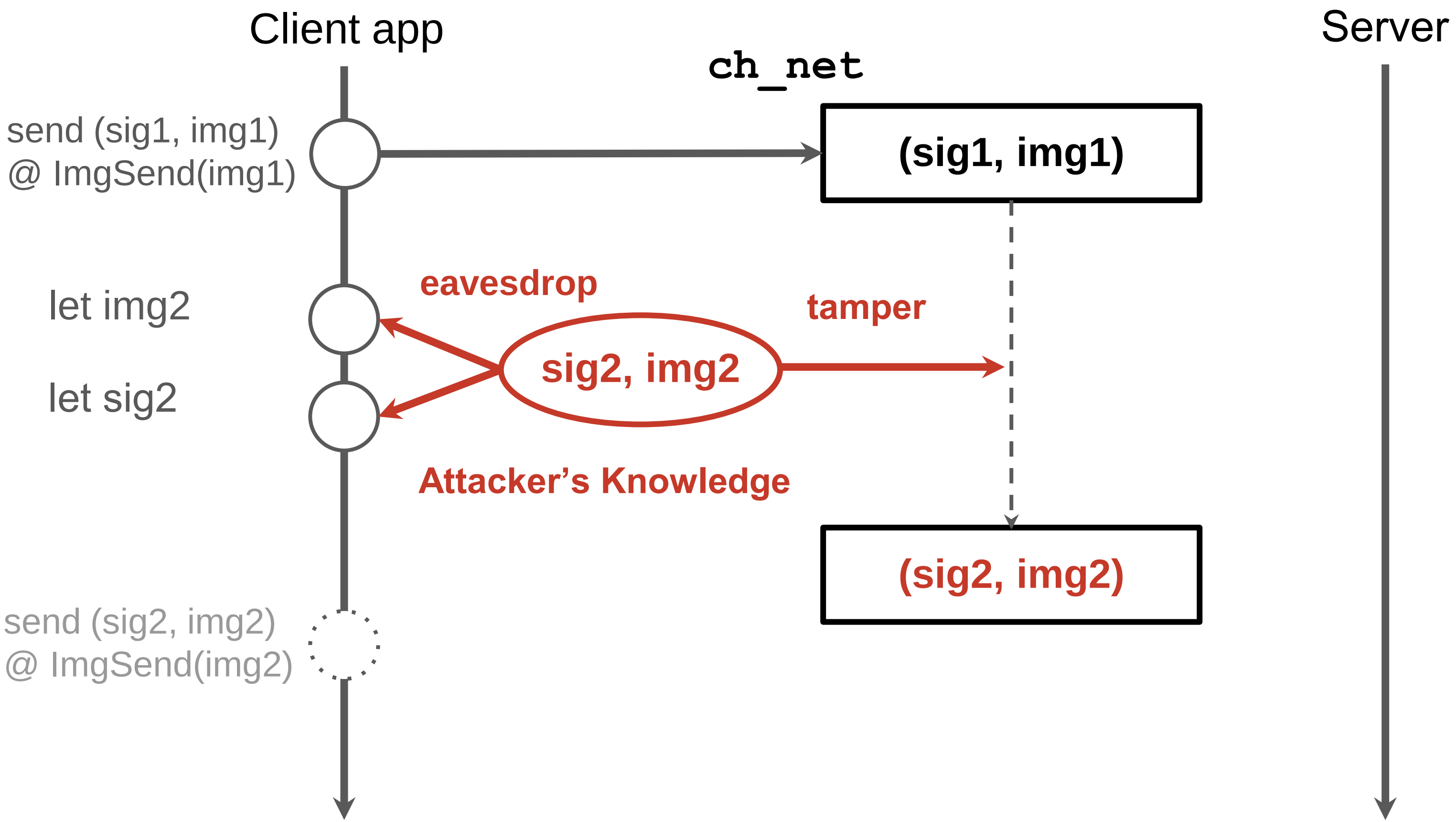
Automatically-found trace (falsified)





Automatically-found trace (falsified)





Client app

ch_net

Server

send (sig1, img1)
@ ImgSend(img1)

(sig1, img1)

eavesdrop

tamper

let img2

let sig2

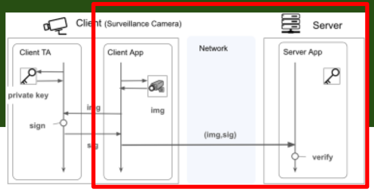
sig2, img2

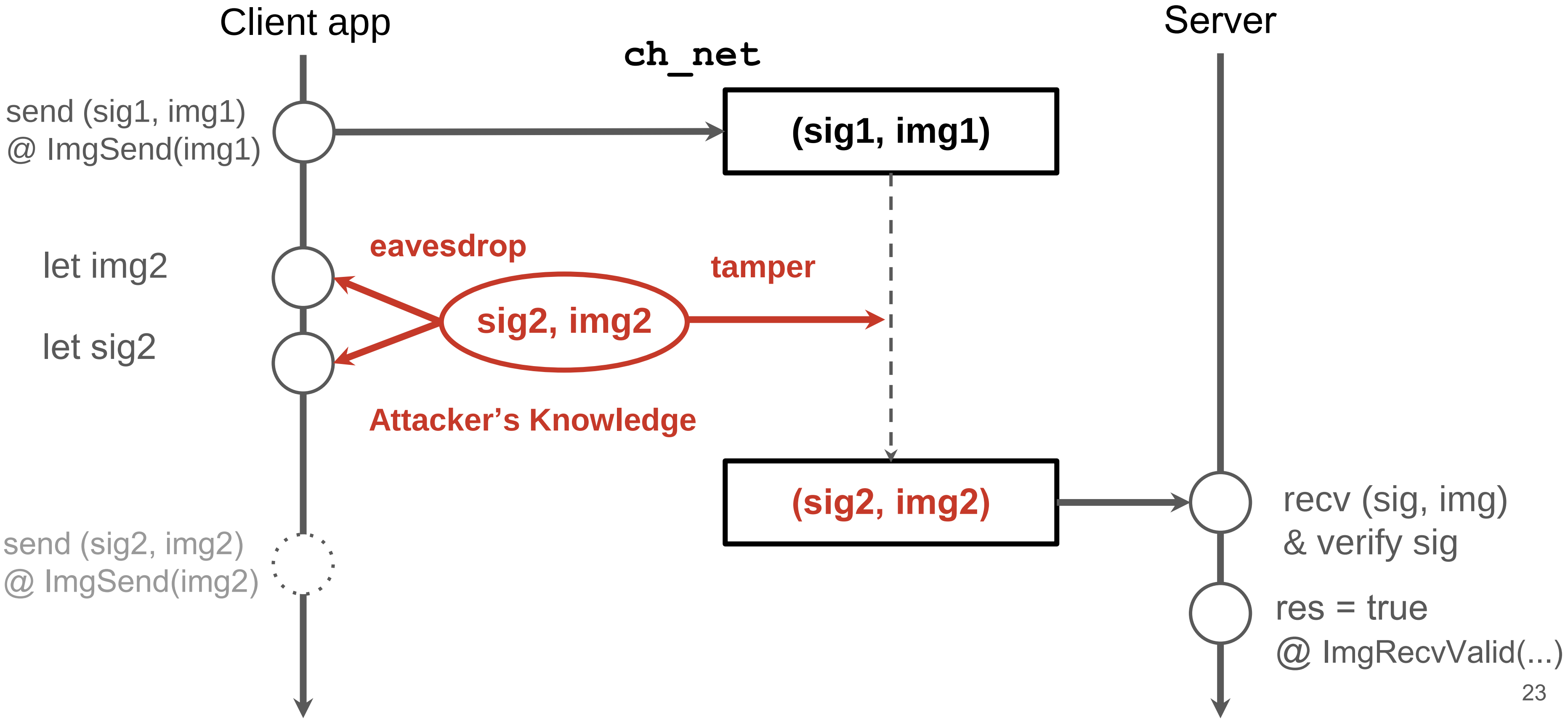
Attacker's Knowledge

(sig2, img2)

send (sig2, img2)
@ ImgSend(img2)

Automatically-found trace (falsified)





Client app

ch_net

Server

send (sig1, img1)
@ ImgSend(img1)

(sig1, img1)

let img2

let sig2

eavesdrop

tamper

sig2, img2

Attacker's Knowledge

send (sig2, img2)
@ ImgSend(img2)

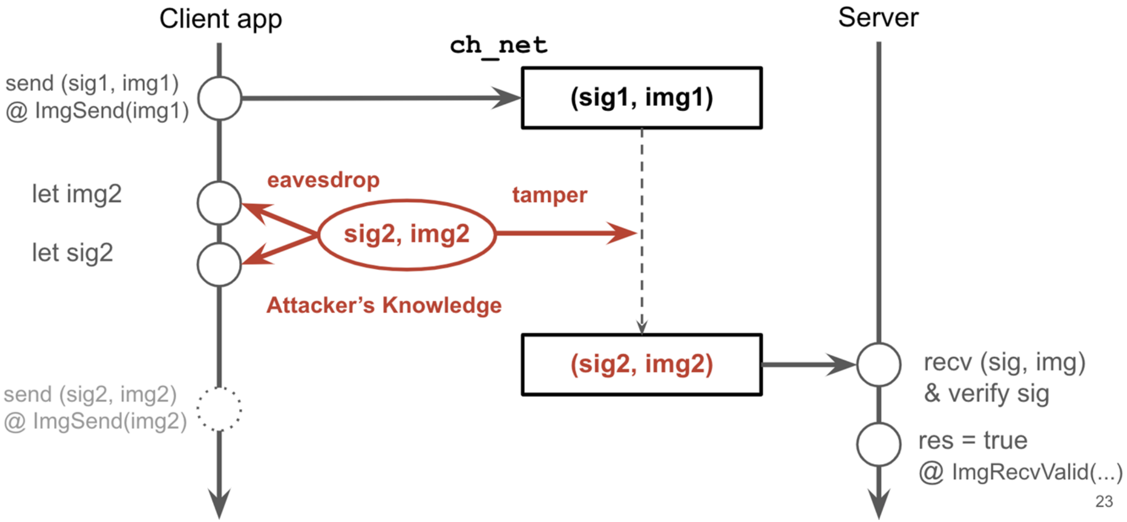
(sig2, img2)

recv (sig, img)
& verify sig

res = true
@ ImgRecvValid(...)

Discussion

- The falsified path implies the ability to automatically detect a nontrivial attack scenario (a path where the violation of security property occurs).
- It should be noted the falsified path is very rare case. In reality, adversary usually attacks parts of a system, but is not able to perform a variety of attacks.



Future Directions

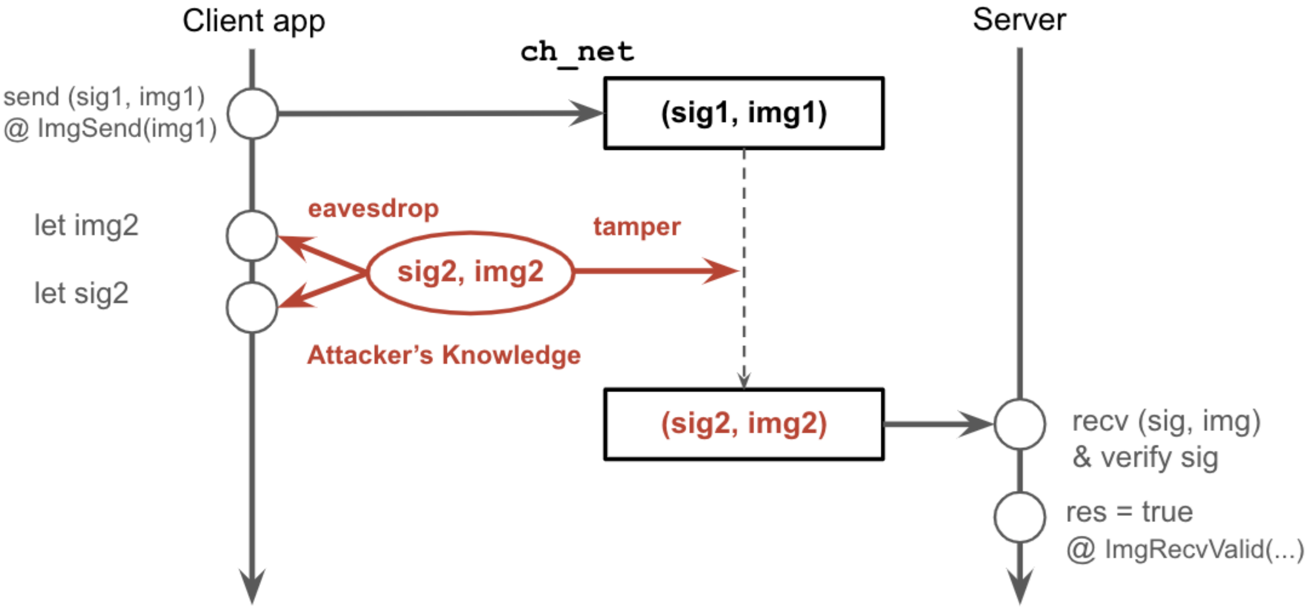
- On modeling, possible directions are
    - to support other classes of objects, such as a shared memory, and
    - to support other operations on objects, such as forking processes or executing files
- On verification, possible directions are
    - to develop an automatic translator, and
    - to improve encoding strategy in Tamarin

Thank you

Appendix

Enhancement of the protocol

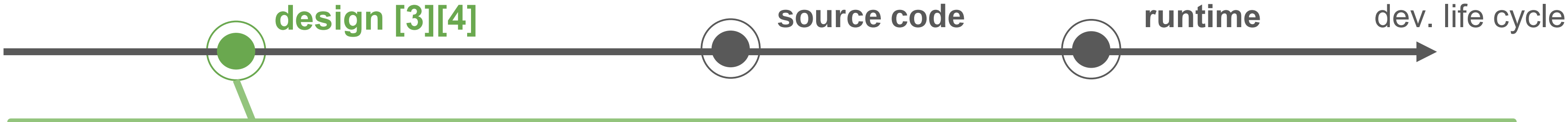
- Enhancing the protocol can be either to attach a sequence number to image data or to introduce nonce.



Threat Modeling on IoT Systems

In Internet of Things (IoT), **security** is one of the top priorities [1][2].

There are various approaches to fortify system security:

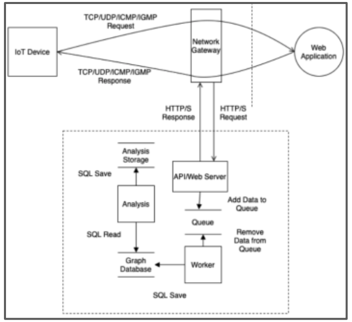


Threat Modeling

- model a system and identify potential threats
- identify security flaw before they become real

data-flow level

Confidentiality,
Integrity,
Authenticity, etc.



---

[1] Al-Fuqaha, ….: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, IEEE Communications Surveys & Tutorials, Vol. 17, No. 4, pp. 2347–2376 (2015).
[2] Kumar, S., Tiwari, P. and Zymbler, M. L.: Internet of Things is a revo- lutionary approach for future technology enhancement: a review, J. Big Data, Vol. 6, p. 111 (2019).
[3] Sion, L., Yskout, K., Van Landuyt, D., van den Berghe, A. and Joosen, W.: Security Threat Modeling: Are Data Flow Diagrams Enough?, Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW’20, New York, NY, USA, Association for Computing Machinery, pp. 254–257 (2020). event-place: Seoul, Republic of Korea.
[4] Union, E.: Regulation (EU) 2016/679, Official Journal of the European Union, Vol. 59, No. 119, pp. 1–88 (2016).

Formal Verification

To thoroughly investigate data flow, *formal verification* is a promising approach.
→ It can verify a property in a **mathematically rigorous** and **provable** way.

Ex: the Tamarin prover (Basin et al. 2015, [5])

```
rule Client_1:
  [ Fr(~k) , !Pk($S, pkS) ] → [ Client_1( $S, ~k ), Out( aenc(~k, pkS) ) ]
rule Serv_1:
  [ !Ltk($S, ~ltkS), In( request ) ]
  --[ AnswerRequest($S, adec(request, ~ltkS)) ]->[ Out( h(adec(request, ~ltkS))) ]
```

😕 Not easy to use for those who do not have expertise of the tool.
→ We want a tool that is **friendly for system programmers.**

---

[5] Schmidt, B., Meier, S., Cremers, C. and Basin, D.: Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties, 2012 IEEE 25th Computer Security Foundations Symposium, pp. 78–94 (2012).

Rabbit Language

Rabbit is a language for both **modeling** and **verification** of data-flow security.

Contributions

- Friendly modeling for system programmers
- Supports primitives for IoT security requirements [6]
    - secure execution environment
    - cryptography
    - access control
- Flexible specification of attacker models

```
process Client(ch_net, ch_rpc) with client_t {
  …
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  …
```

```
allow server_t server_file_t [read, write]
allow client_t client_net_t [send, recv]
```

```
allow attacker_t server_file_t [ eavesdrop ]
allow attacker_t chan_net_t [ eavesdrop, tamper, drop ]
```

---

[6] T. A. Ahanger and A. Aljumah, "Internet of Things: A Comprehensive Study of Security Issues and Defense Mechanisms," in IEEE Access, vol. 7, pp. 11020-11028, 2019, doi: 10.1109/ACCESS.2018.2876939.

Comparison of Rabbit and Other Tools

	Familiar to system programmer	IoT Security Solutions	Flexibility of Attacker models	Other features
Rabbit	◎	○	○	–
Tamarin prover	×	△	○	Unbounded Sessions
SAPIC+ [7]	△	○	△	Reducible to many verification tools
PSec [8]	△	○	△	Programming language
UML-based Tools [9]	○	△	△	Visualizer

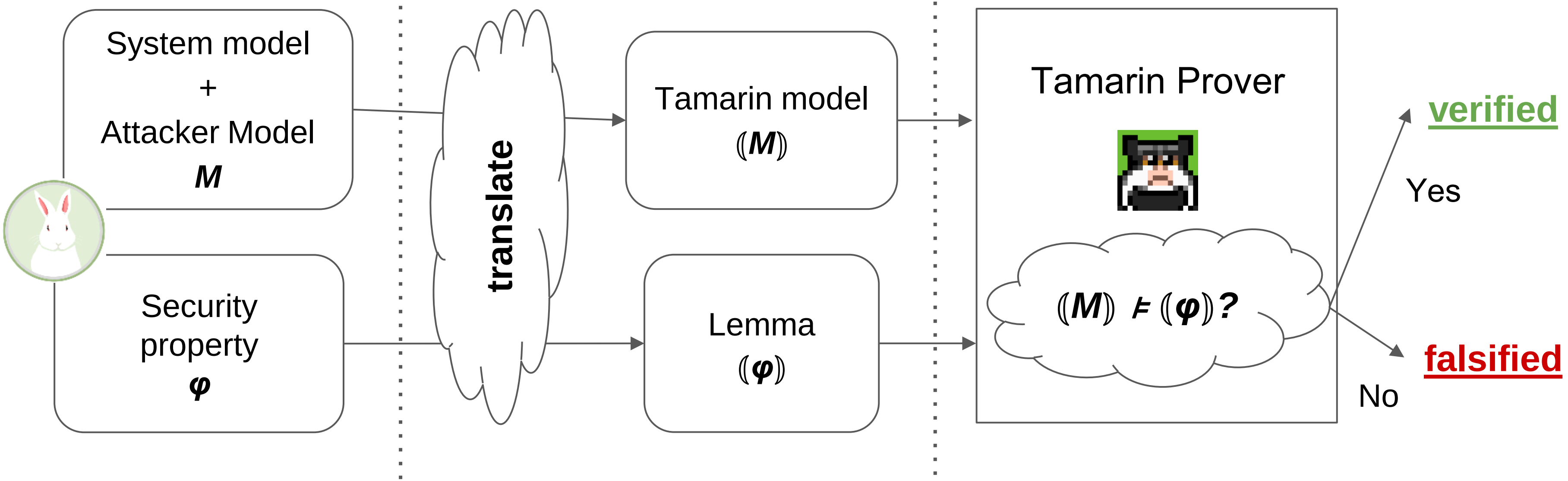
[7] Kremer, S. and K¨unnemann, R.: Automated analysis of security protocols with global state, J. Comput. Secur., Vol. 24, No. 5, pp. 583–616 (2016).

[8] Kushwah, S., Desai, A., Subramanyan, P. and Seshia, S. A.: PSec: Programming Secure Distributed Systems Using Enclaves, Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, ASIA CCS ’21, New York, NY, USA, Association for Computing Machinery, pp. 802–816 (2021). event-place: Virtual Event, Hong Kong.

[9] Ju¨rjens, J.: UMLsec: Extending UML for Secure Systems Develop- ment, UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany, September 30 - October 4, 2002.

Verification with Rabbit

Rabbit verifies a security property by **translating Rabbit model into Tamarin**.



System model
+
Attacker Model
M

Security property
φ

translate

Tamarin model
$(\!(\boldsymbol{M})\!)$

Lemma
$(\!(\boldsymbol{\varphi})\!)$

Tamarin Prover

$(\!(\boldsymbol{M})\!) \vDash (\!(\boldsymbol{\varphi})\!)$ **?**

Yes

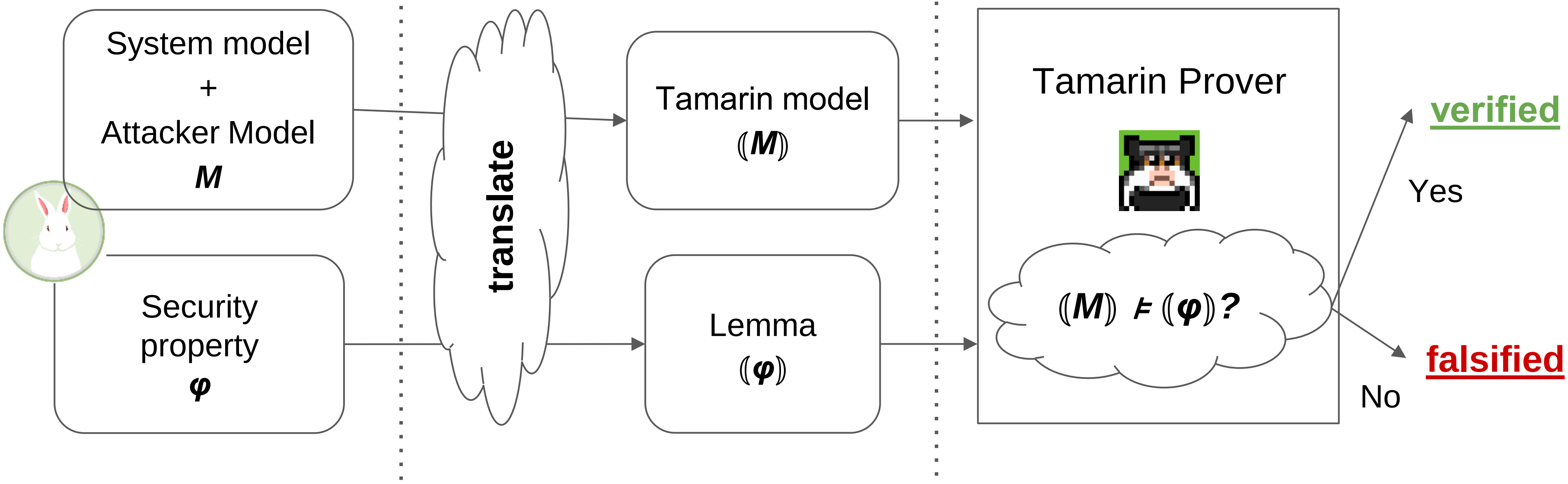
<u>**verified**</u>

No

<u>**falsified**</u>

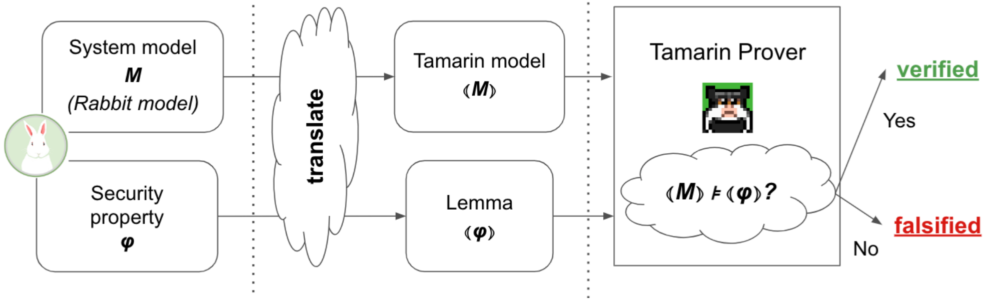
Verification with Rabbit

- Systematic (but not automated) translation
- Security properties are directly written in Rabbit



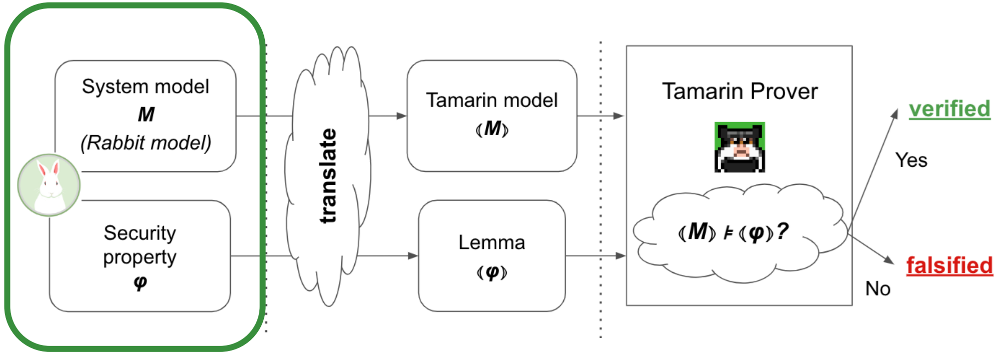
Outline

- Rabbit Language
    - Modeling Overview
    - Rabbit Program
- Translation to Tamarin
- Experiments
    - Experiment 1: Reachabitlity
    - Experiment 2: Authenticity
- Conclusions



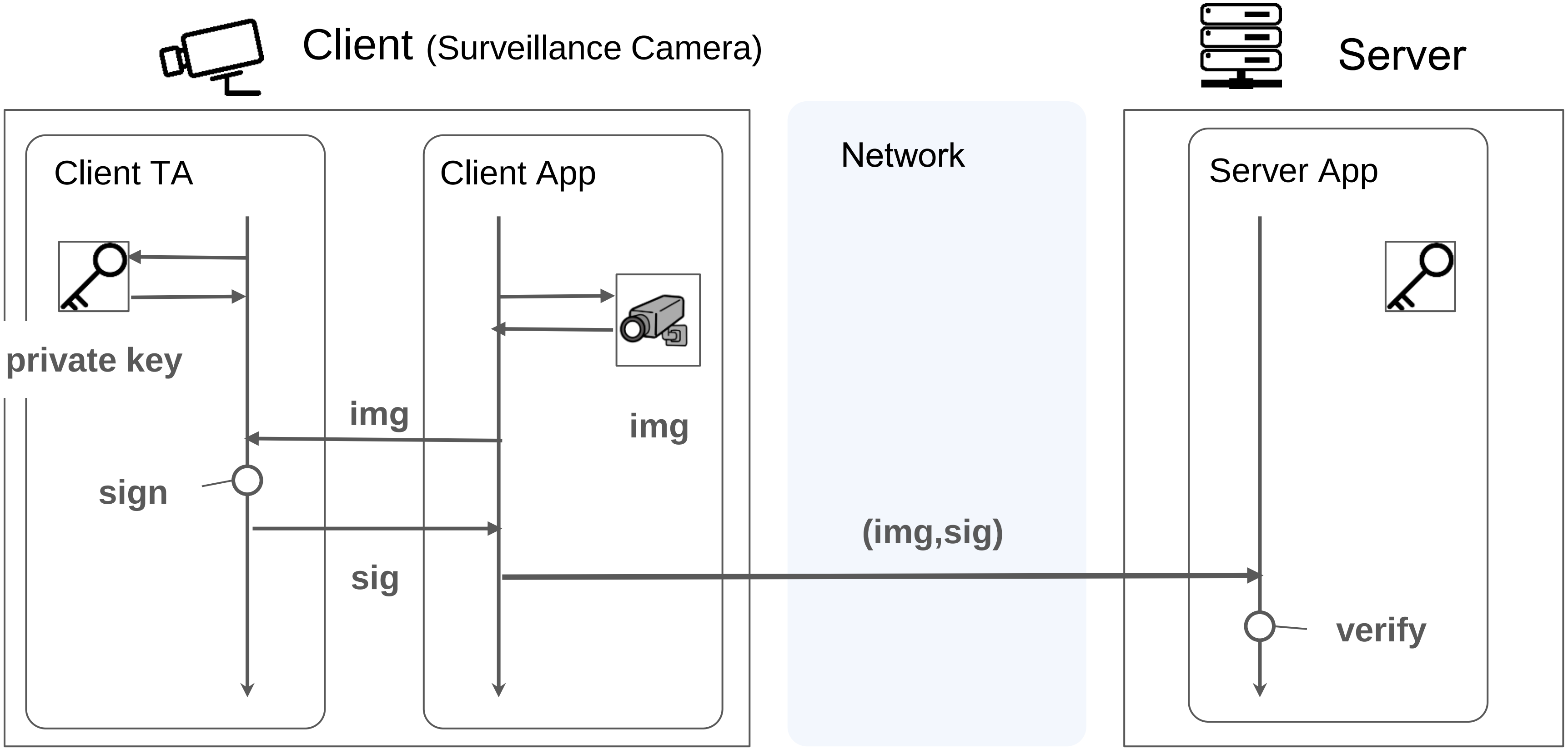
Outline

- **Rabbit Language**
  - **Modeling Overview**
  - **Rabbit Program**
- Translation to Tamarin
- Experiments
  - Experiment 1: Reachabitlity
  - Experiment 2: Authenticity
- Conclusions



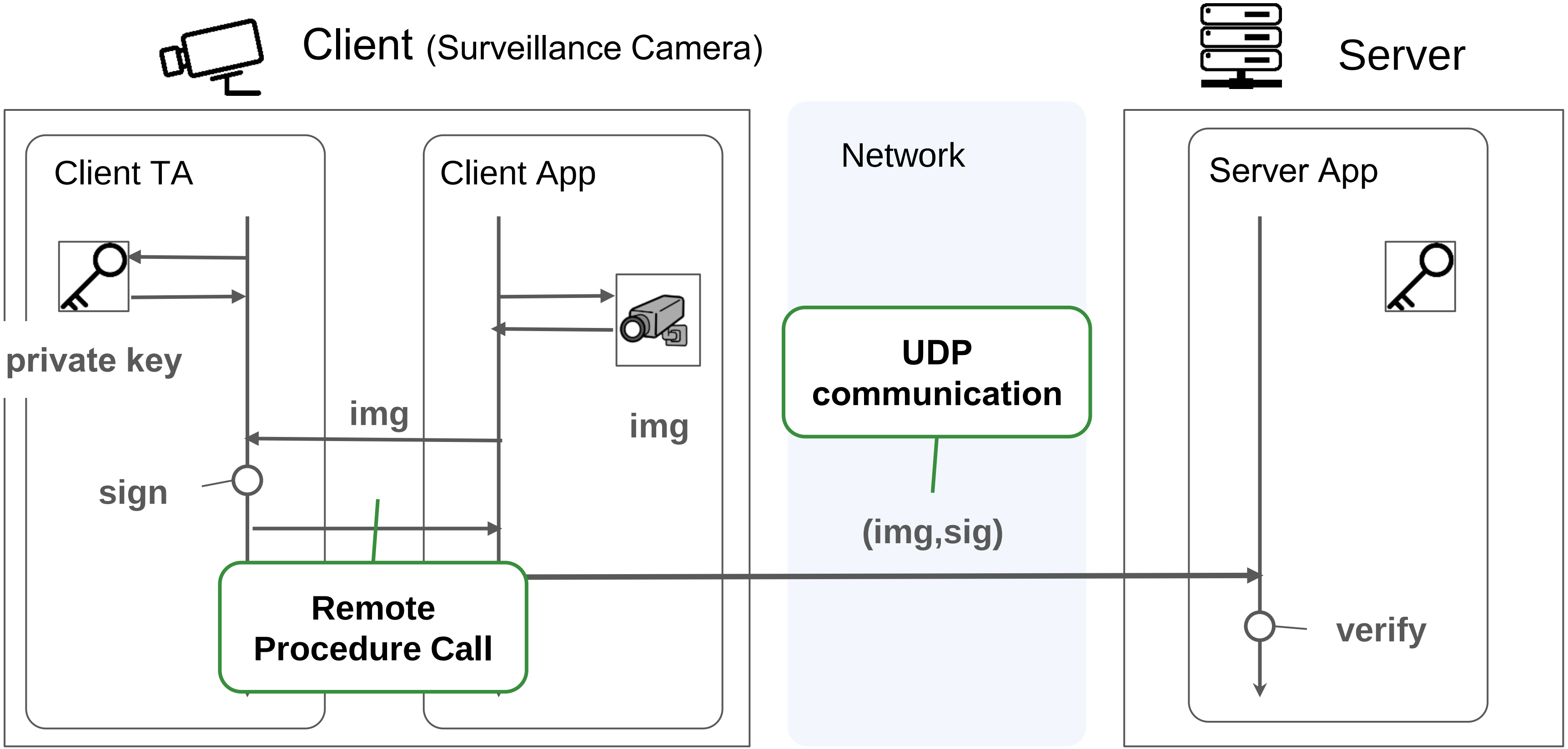
The Cam-Image System

TA ⋯ Trusted Application
(running in a secure execution environment)



The Cam-Image System

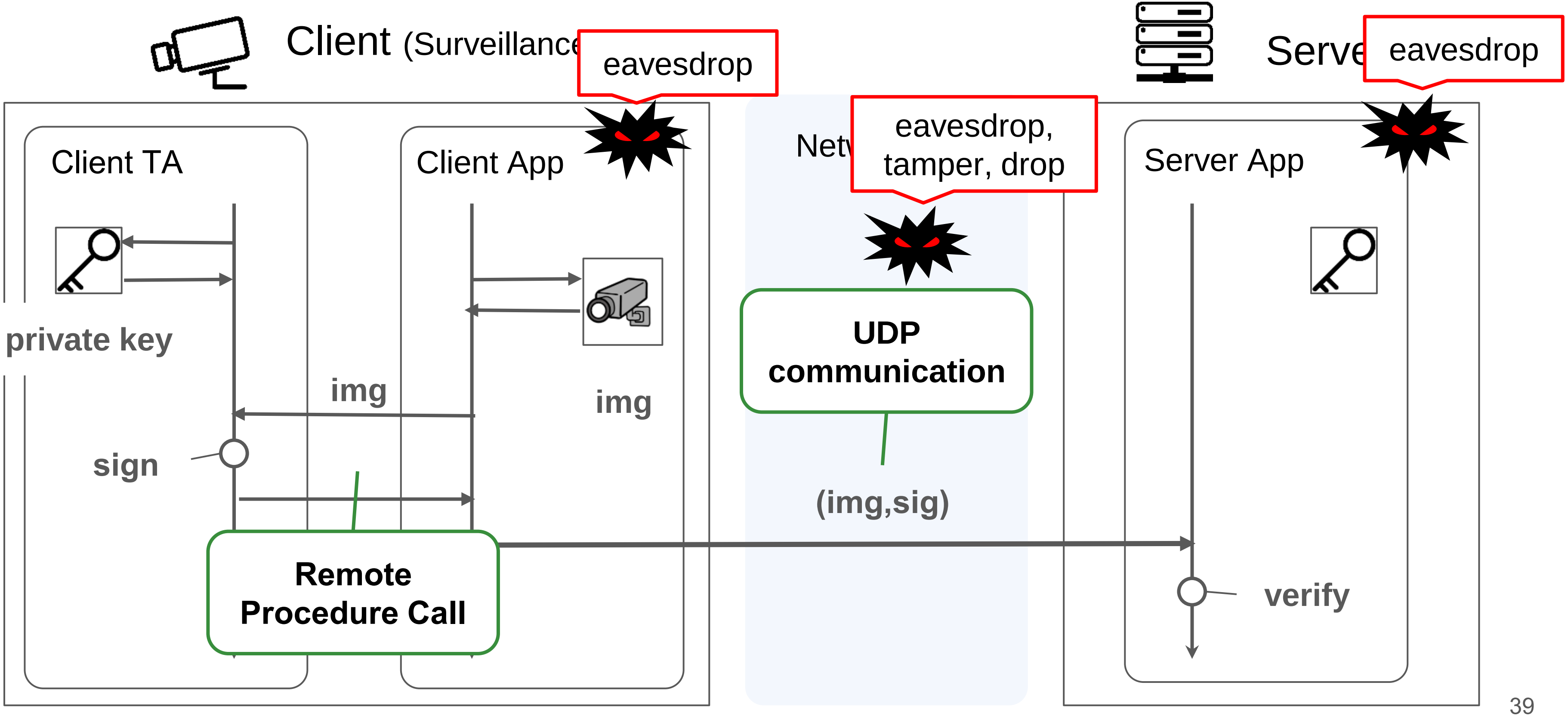
TA ⋯ Trusted Application
(running in a secure execution environment)



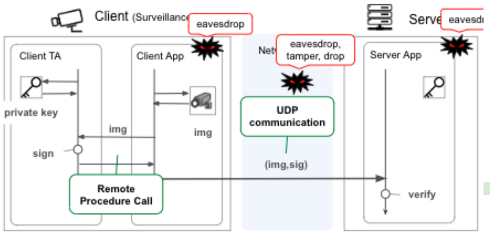
Rabbit Language - Modeling Overview

The Cam-Image System

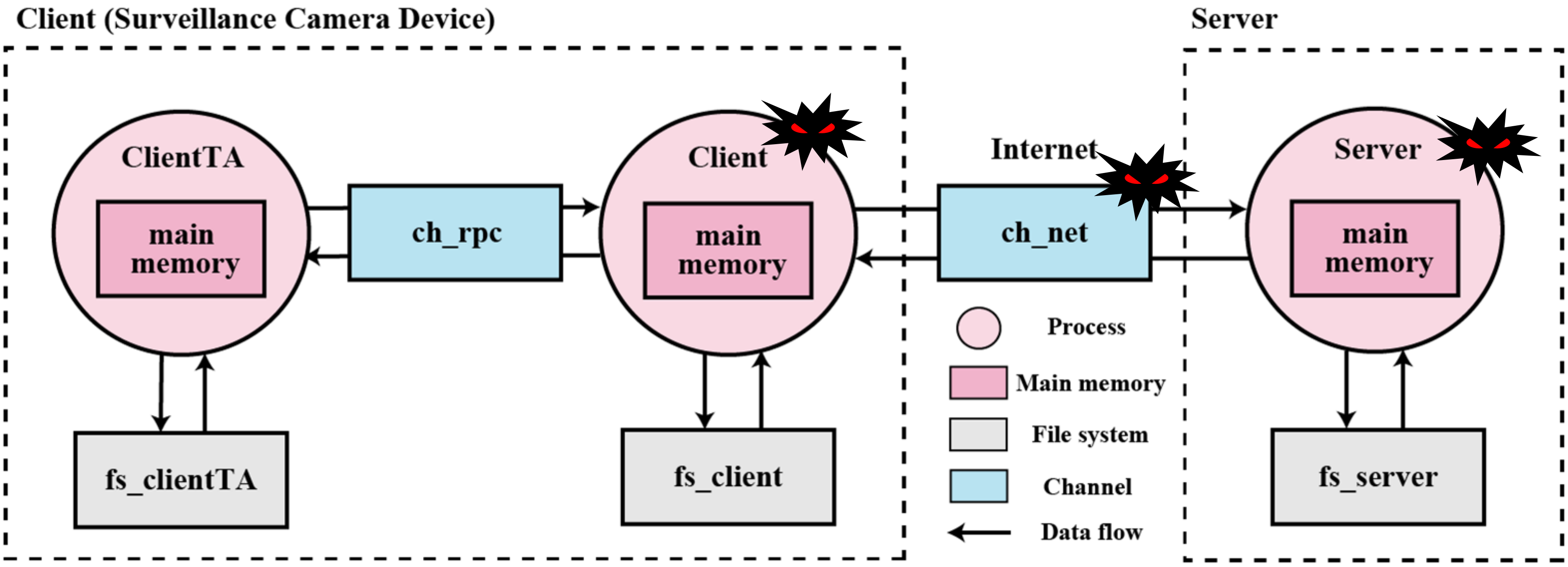
TA ··· Trusted Application
(running in a secure execution environment)



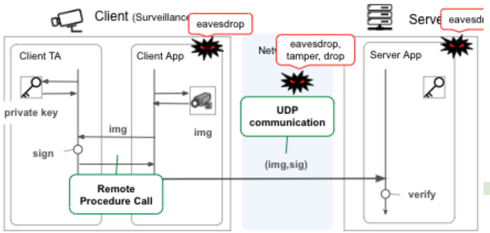
Rabbit Model of the Cam-Image System



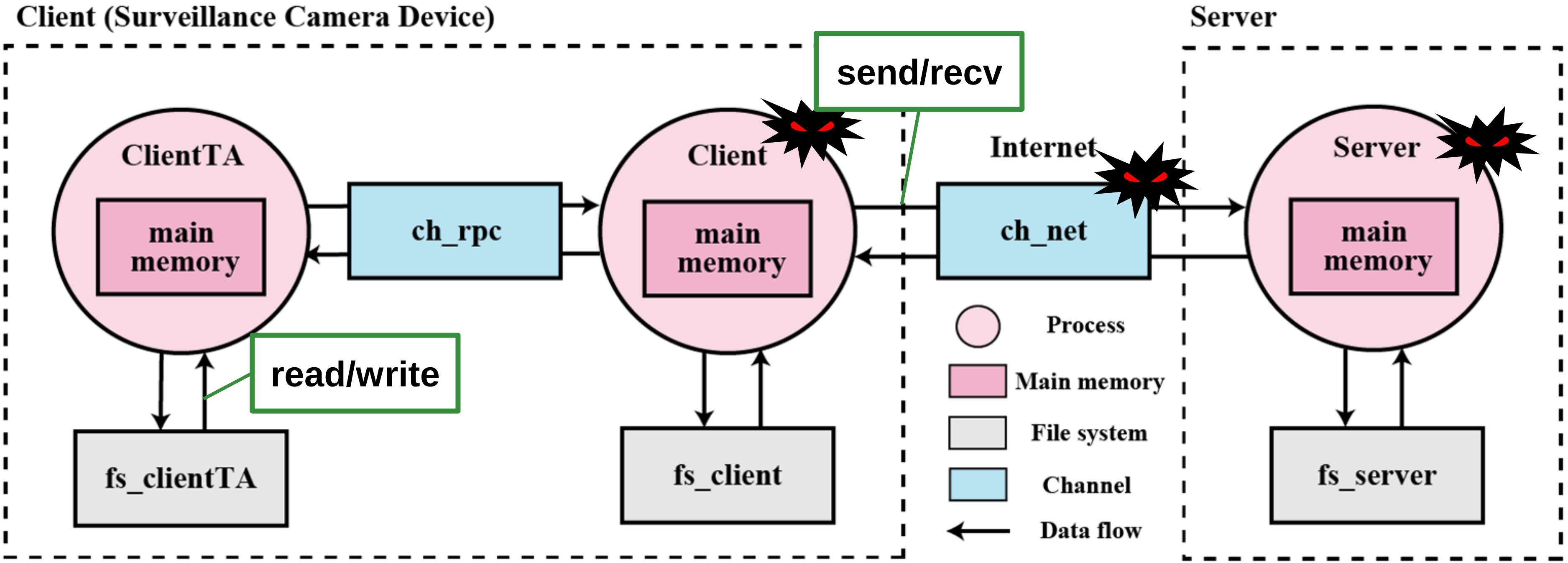
Rabbit components consists of **processes, file systems, channels**



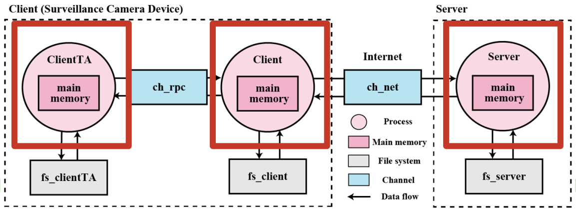
Rabbit Model of the Cam-Image System



Access control is configured by *MAC* (Mandatory Access Control)



Process Declaration



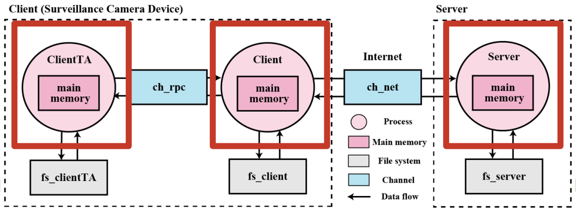
- Library functions like system calls in Linux
- Control Structures like for/if statements

```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

  main {
    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  }
}
```

```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

```
process Server(ch_net) with server_t {
  …
     let res = verify(p.fst, p.snd, pubkey);
     if (res) {
       skip @ ImgRecvValid(p.snd);
     }
  …
}
```

Process Declaration



- Library functions like system calls in Linux
- Control Structures like for/if statements

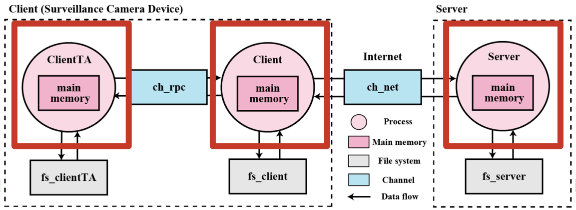
```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

  main {
    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  }
}
```

open system call

```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

```
process Server(ch_net) with server_t {
  …
     let res = verify(p.fst, p.snd, pubkey);
     if (res) {
       skip @ ImgRecvValid(p.snd);
     }
  …
}
```

Process Declaration



- Library functions like system calls in Linux
- Control Structures like for/if statements

```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  }
}
```

for statement

open system call

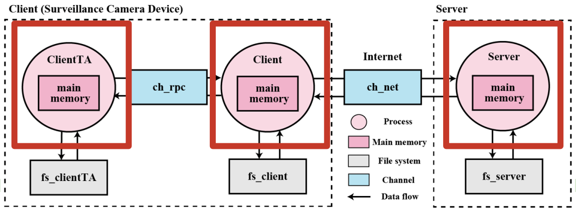
```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

```
process Server(ch_net) with server_t {
  …
     let res = verify(p.fst, p.snd, pubkey);
     if (res) {
       skip @ ImgRecvValid(p.snd);
     }

}
```

if statement

Rabbit Language - Rabbit Program

Process Declaration



- Library functions like system calls in Linux
- Control Structures like for/if statements

```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_ne (sig, image)) @ ImgSend(image);
    }
  }
}
```

for statement

open system call

rpc communication

```
process ClientTA(ch_rpc) with clientTA_t {
  func sign_image(image, privkey_path) {
    let sig = sign(image, privkey0);
    return sig;
  } …
}
```

digital signature

```
process Server(ch_net) with server_t {
  …
    let res = verify(p.fst, p.snd, pubkey);
    if (res) {
      skip @ ImgRecvValid(p.snd);
    }

}
```

if statement

Comparison with Tamarin

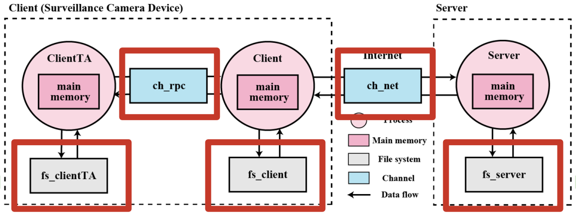
- Library functions like system calls in Linux
- Control Structures like for/if statements

```
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera"; …

  main {
    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
      let sig = invoke(ch_rpc, invoke_func, …);
      send(ch_net, (sig, image)) @ ImgSend(image);
    }
  }
}
```

```
rule Rule_ClientTA_76_1 :
  [
    F_Proc_ClientTA_75_1(
      called('1')
      , <'0', image_init_0, image_now_0>
      , <'0', privkey_path_init_0, privkey_path_now_0>
      , <'0', fek_init_0, fek_now_0>
      , <'1', fek_init_1, fek_now_1>
    )
  ]
  -->
  [
    F_Proc_ClientTA_76_1(
      called('1')
      , <'0', fd(privkey_path_now_0),
fd(privkey_path_now_0)>
      , <'0', image_init_0, image_now_0>
      , <'0', privkey_path_init_0, privkey_path_now_0>
      , <'0', fek_init_0, fek_now_0>
      , <'1', fek_init_1, fek_now_1>
    )
  ]
```

Rabbit Language - Rabbit Program

File System & Channel



```
filesys Client_FS = [
  { path: "/dev/camera", data: dont_care, type: client_file_t }
]
filesys Server_FS = [
  { path: "/secret/pub", data: pk(priv_k), type: server_file_t }
]
filesys ClientTA_FS = [
  { path: "/secret/priv", data: enc(priv_k, sym_k), type: clientTA_file_t }
]
```

```
channel ch_net = { connection: datagram, type: chan_net_t }

channel ch_rpc = { connection: stream, type: chan_rpc_t }
```

Access Control Policy

Access control policies ··· Configuration of permissions from subject to objects

a type for subject a type for object permissions

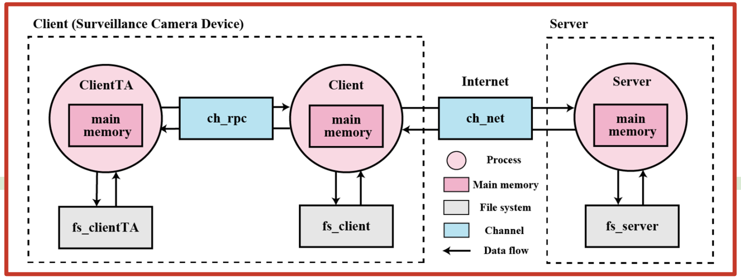
```
allow   server_t   server_file_t   [read, write]
allow   client_t   chan_net_t      [send, recv]
```

Attacker is also modeled by allow rules.

allow **attacker_t** client_t **[eavesdrop]**
allow **attacker_t** chan_net_t **[eavesdrop, tamper, drop]**

Rabbit Language - Rabbit Program

System Instantiation



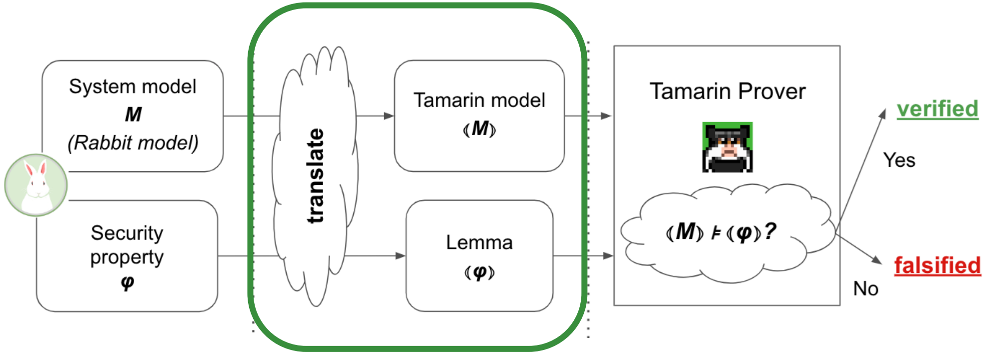
```
system
  Client([ch_net, ch_rpc]) with ClientFS
  || ClientTA([ch_rpc]) with ClientTA_FS
  || Server([ch_net]) with ServerFS
requires [
  "lemma Authenticity :
    all-traces
    "All image #i. ImgRecvValid(image) @ #i
              ==> Ex #j . ImgSend(image) @ #j & #j < #i"
  "]
```

Process Instantiation

Properties to verify

Outline

- Rabbit Language
  - Modeling Overview
  - Rabbit Program
- **Translation to Tamarin**
- Experiments
  - Experiment 1: Reachabitlity
  - Experiment 2: Authenticity
- Conclusions



Translated Tamarin Model

A statement in Rabbit is translate into one or more rules in Tamarin

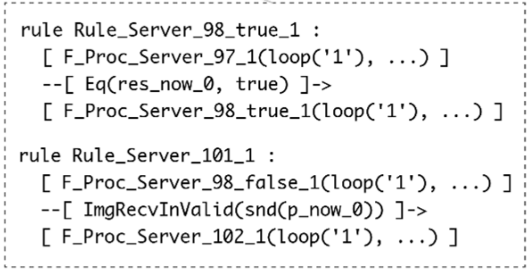
- 30 lines of statements → 99 rules in Tamarin

```
process Client(ch_net, ch_rpc) with
client_t {
  main {
    let image_fd = open(dev_path);
    for i in range(1, 4) {
      let image = read(image_fd);
    }
  }
}
```

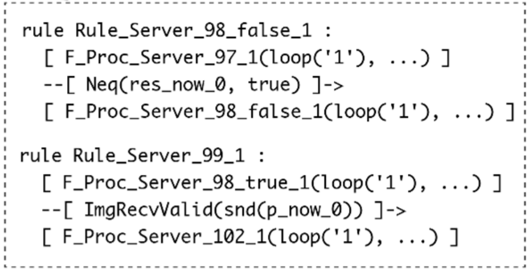
```
system
  Client([ch_net, ch_rpc]) with ClientFS
  || ClientTA([ch_rpc]) with
ClientTA_FS
  || Server([ch_net]) with ServerFS
requires [
  "lemma Authenticity : … "
]
```

LOC: 123

```
// let image = read(image_fd);
rule Rule_Client_63_1 :
  [ F_Proc_Client_62_1(loop('1'), ...)
    , File($Client, 'devCamera', data)
   , Fr(~image) ]
  --[ AttackerEavesdrop(~image) ]->
  [ F_Proc_Client_63_1(loop('1'), <'0', ~image, ~image>, ...)
    , File($Client, 'devCamera', ~image)
    , Out(~image) ]
```



```
rule Rule_Server_98_true_1 :
  [ F_Proc_Server_97_1(loop('1'), ...) ]
  --[ Eq(res_now_0, true) ]->
  [ F_Proc_Server_98_true_1(loop('1'), ...) ]

rule Rule_Server_101_1 :
  [ F_Proc_Server_98_false_1(loop('1'), ...) ]
  --[ ImgRecvInValid(snd(p_now_0)) ]->
  [ F_Proc_Server_102_1(loop('1'), ...) ]
```

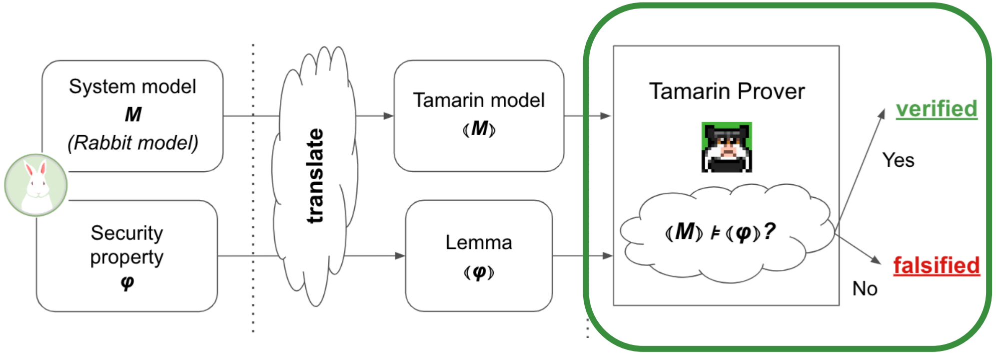


```
rule Rule_Server_98_false_1 :
  [ F_Proc_Server_97_1(loop('1'), ...) ]
  --[ Neq(res_now_0, true) ]->
  [ F_Proc_Server_98_false_1(loop('1'), ...) ]

rule Rule_Server_99_1 :
  [ F_Proc_Server_98_true_1(loop('1'), ...) ]
  --[ ImgRecvValid(snd(p_now_0)) ]->
  [ F_Proc_Server_102_1(loop('1'), ...) ]
```

LOC: 1467

Outline

- Rabbit Language
    - Modeling Overview
    - Rabbit Program
- Translation to Tamarin
- **Experiments**
    - **Experiment 1: Reachabitlity**
    - **Experiment 2: Authenticity**
- Conclusion



Experiments Overview

Experiment 1

Check the validity of translation

- the Cam-Image system
- No attacks

Experiment 2

Check how different attacker models affect results

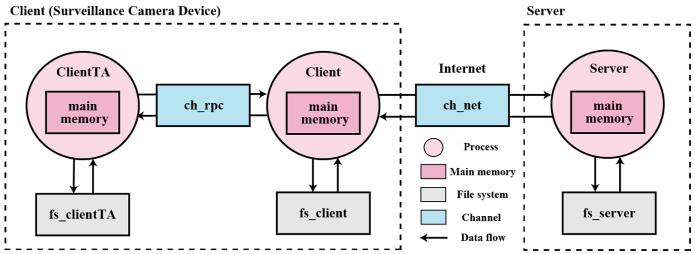
- the Cam-Image system
- various attacks on network

Machine specification

- OS: Ubuntu 22.04, Memory: 252 GB, CPU: Xeon E5-2687 W, 3.1G Hz, 8 core

Version of the Tamarin prover: 1.7.1

Experiment 1: Reachability



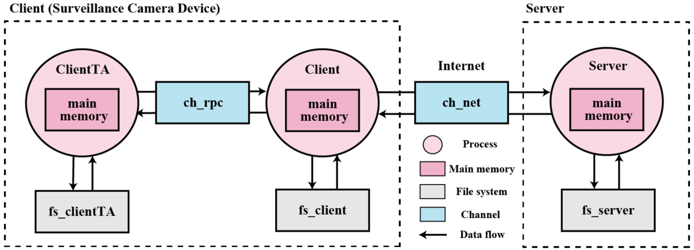
Target Model

- The same system model as the Cam-Image system.
- We consider no attacks on the system.

Property to verify

```
lemma Finish :
    exists-trace
    "Ex #i #j #k. ClientFin() @ #i & TAFin() @ #j & ServerFin() @ #k"
```

Experiment 1: Reachability



Target Model

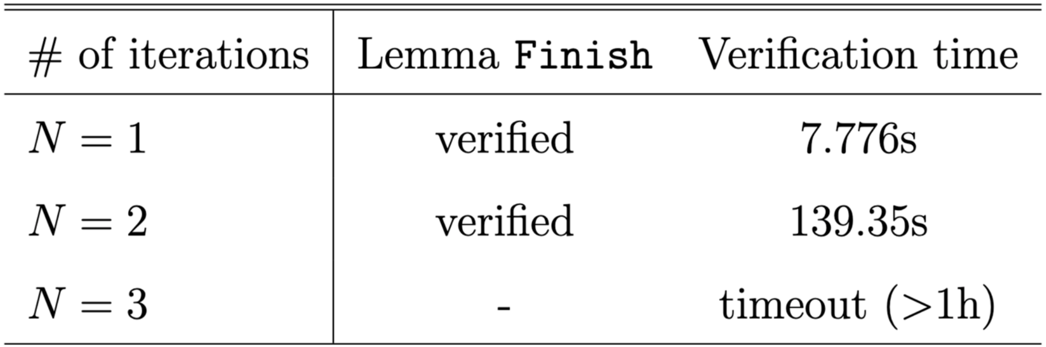
- The same system model as the Cam-Image system.
- We consider no attacks on the system.

Property to verify

Is there any trace where three processes finish their executions?

```
lemma Finish :
    exists-trace
    "Ex #i #j #k. ClientFin() @ #i & TAFin() @ #j & ServerFin() @ #k"
```

Experiment 1: Result



# of iterations	Lemma `Finish`	Verification time
$N = 1$	verified	7.776s
$N = 2$	verified	139.35s
$N = 3$	-	timeout (>1h)

The increase of N (N := # of iterations) largely affect the verification time.

→ Non-determinism in <u>conditional branching</u> and <u>random receptions of messages (in UDP)</u> increases.

```
// if (res)
rule Rule_Server_98_true_1 :
  [ F_Proc_Server_97_1(...) ]
  --[ Eq(res_now_0, true) ]->
  [ F_Proc_Server_98_true_1(...) ]

rule Rule_Server_98_false_1 :
  [ F_Proc_Server_97_1(...) ]
  --[ Neq(res_now_0, true) ]->
  [ F_Proc_Server_98_false_1(...) ]
```

```
rule Rule_Server_96_1 :
  [ F_Proc_Server_95_1(...)
   , Msg('ch_net', 's', i, p) ]
  -->
  [ F_Proc_Server_96_1(...) ]
```

Experiment 2: Authenticity with Different Attacker Models

Target Model

- The same system model as the Cam-Image system
- We consider an attacker that is capable of
  - eavesdropping on the main memory of the client & server app.
  - combination of attacks (eavesdrop, tamper, drop) on network messages.

Property to verify

```
lemma Authenticity :
  all-traces
  "All x #i . ImgRecvValid(x) @ #i ==> Ex #j . ImgSend(x) @ #j & #j < #i"
```

Experiment 2: Authenticity with Different Attacker Models

Target Model

- The same system model as the Cam-Image system
- We consider an attacker that is capable of
  - eavesdropping on the main memory of the client & server app.
  - combination of attacks (eavesdrop, tamper, drop) on network messages.

Property to verify

When the server verifies signature successfully, then the image is sent by the right client formerly (*correspondence assertion*).

```
lemma Authenticity :
  all-traces
  "All x #i . ImgRecvValid(x) @ #i ==> Ex #j . ImgSend(x) @ #j & #j < #i"
```

Experiment 2: Result

eavesdrop (on channel)

tamper

drop

eavesdrop & tamper & drop

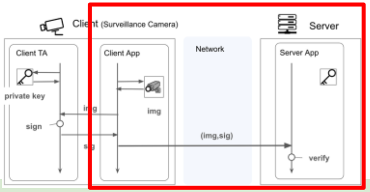
	`-`	`e`	`t`	`d`	`et`	`ed`	`td`	`etd`
N = *1*	6.538s	6.732s	7.890s	6.600s	8.150s	6.790s	8.020s	8.430s
N = *2*	125.622s	137.198s	525.380s	354.025s	510.905s	334.795s	1427.480s	1418.890s

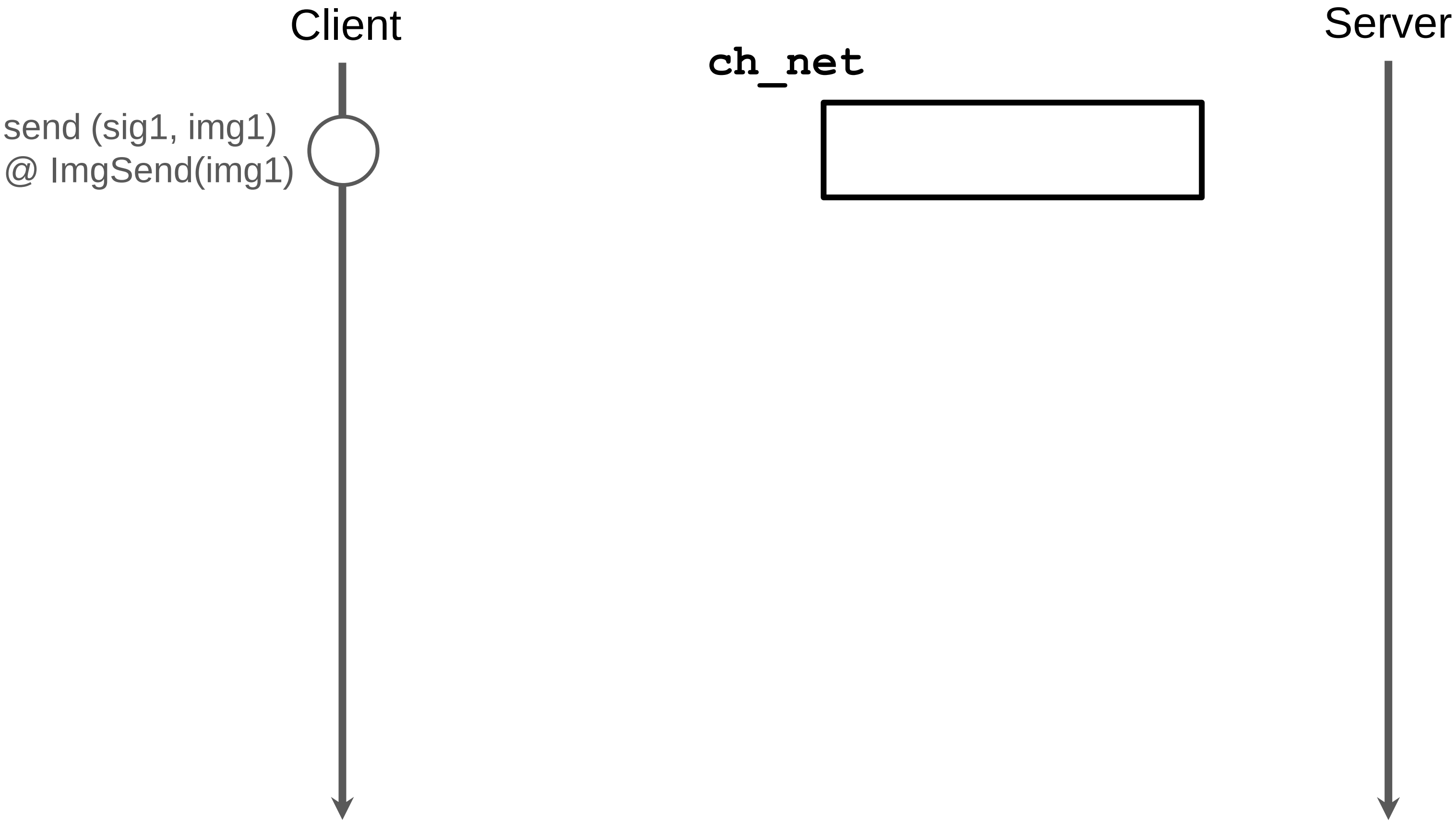
🟩 **verified** 🟥 **falsified**

Observation

- When the attacker is able to tamper messages, the authenticity is falsified.
- The increase of N largely affect the verification time.
- The verification time is large when we consider active attacks.

Experiment 2: Falsified Trace



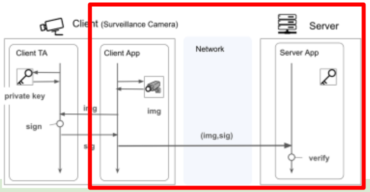

Client

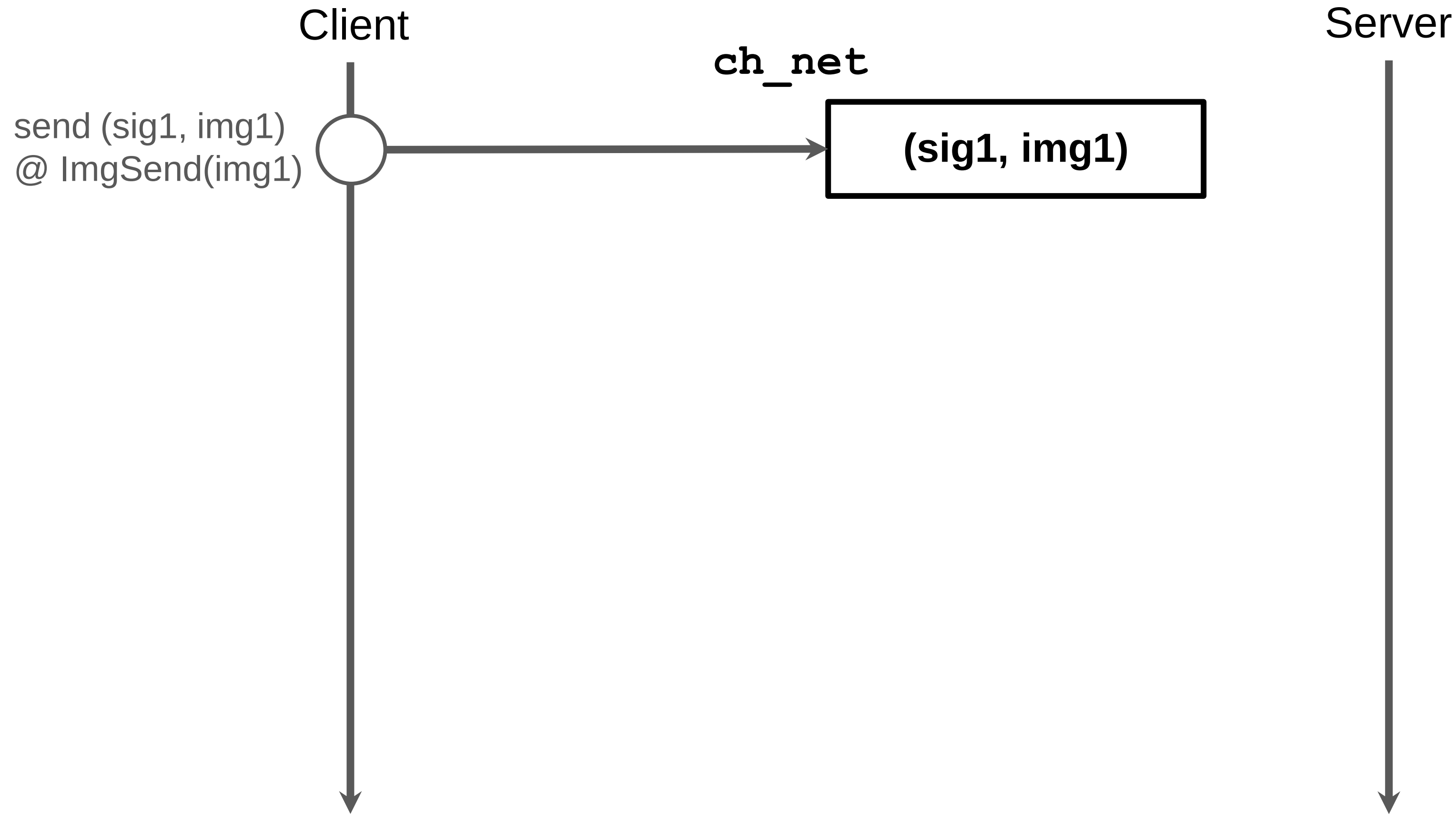
`ch_net`

Server

send (sig1, img1)
@ ImgSend(img1)

Experiment 2: Falsified Trace





Client

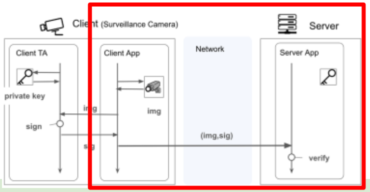
Server

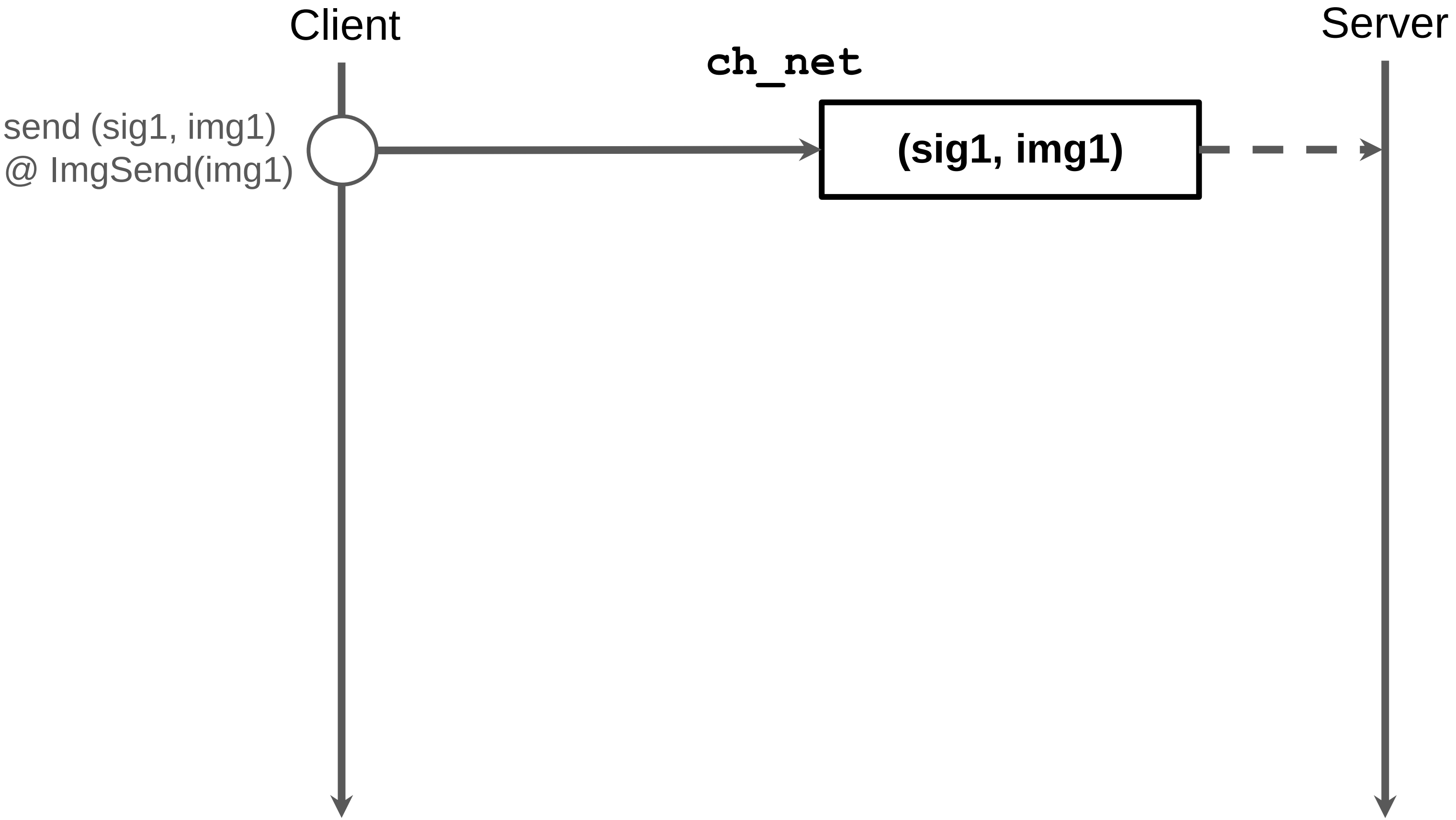
`ch_net`

send (sig1, img1)
@ ImgSend(img1)

(sig1, img1)

Experiment 2: Falsified Trace





Client

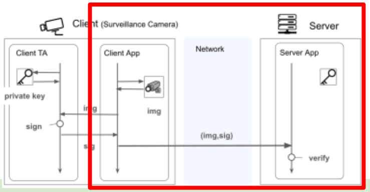
Server

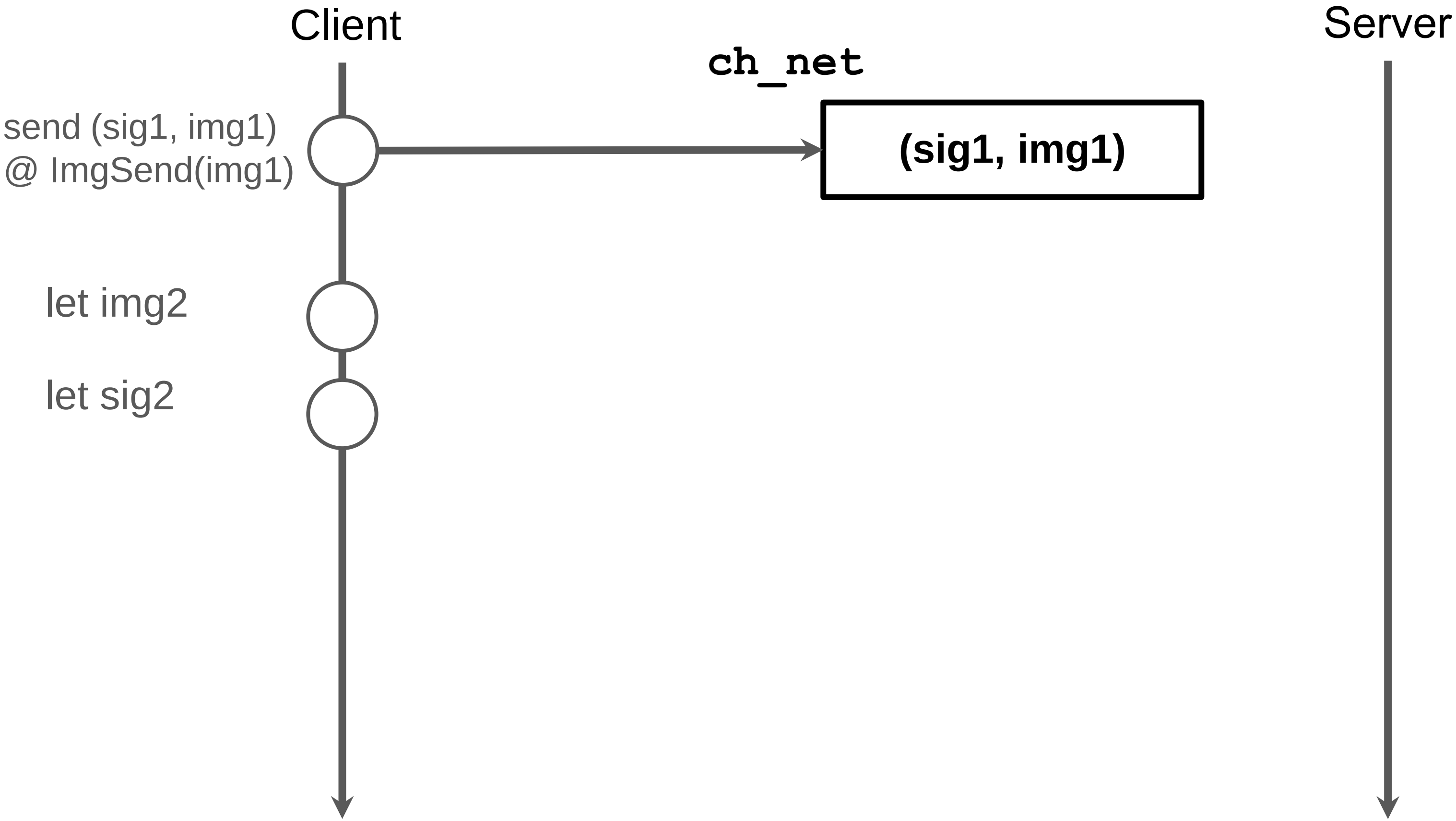
`ch_net`

send (sig1, img1)
@ ImgSend(img1)

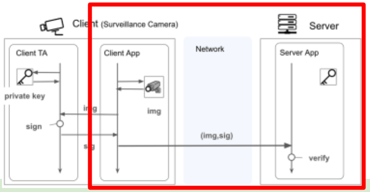
(sig1, img1)

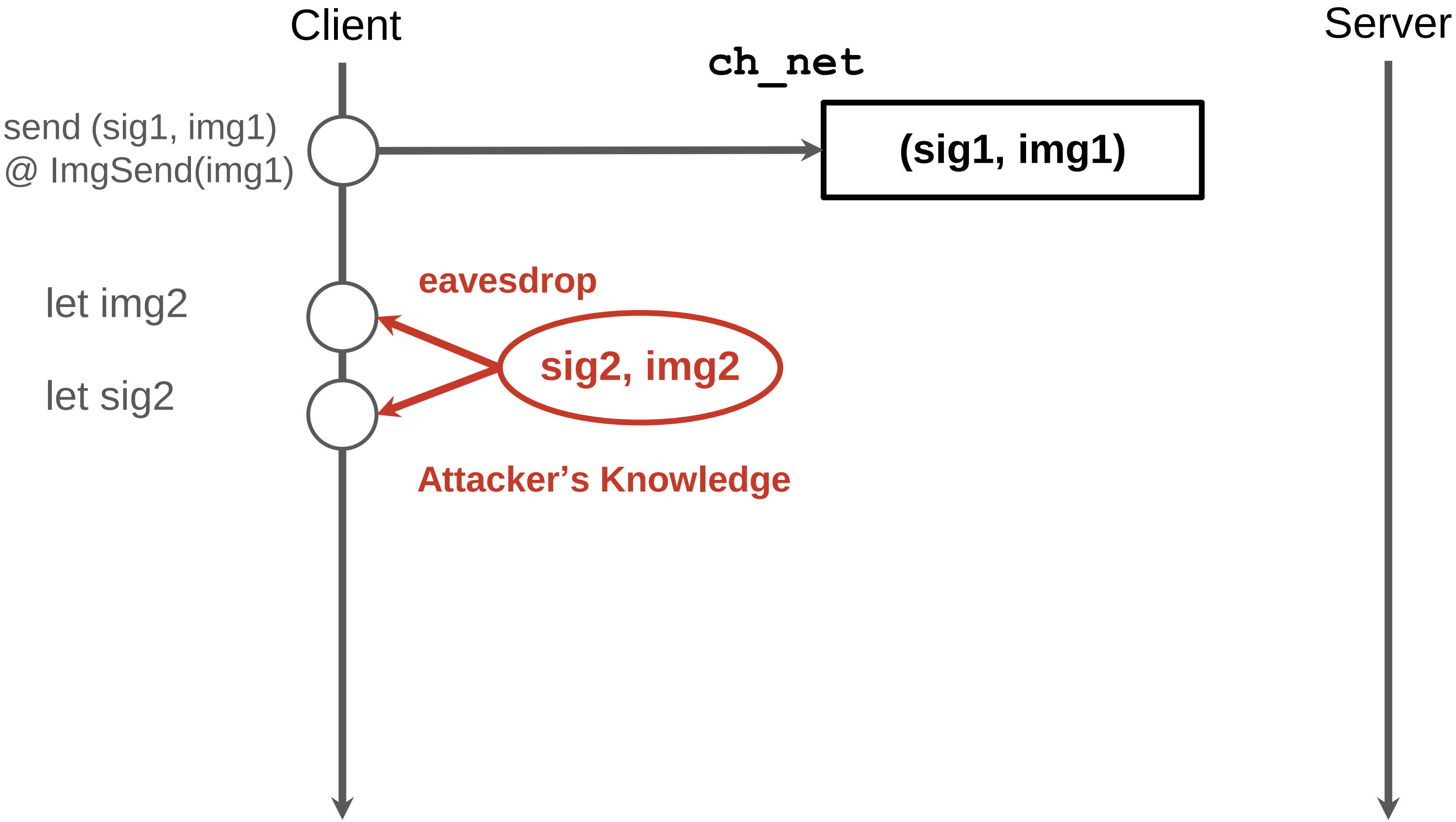
Experiment 2: Falsified Trace





Experiment 2: Falsified Trace





Client

Server

`ch_net`

send (sig1, img1)
@ ImgSend(img1)

(sig1, img1)

eavesdrop

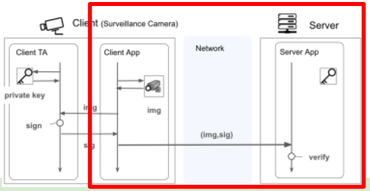
let img2

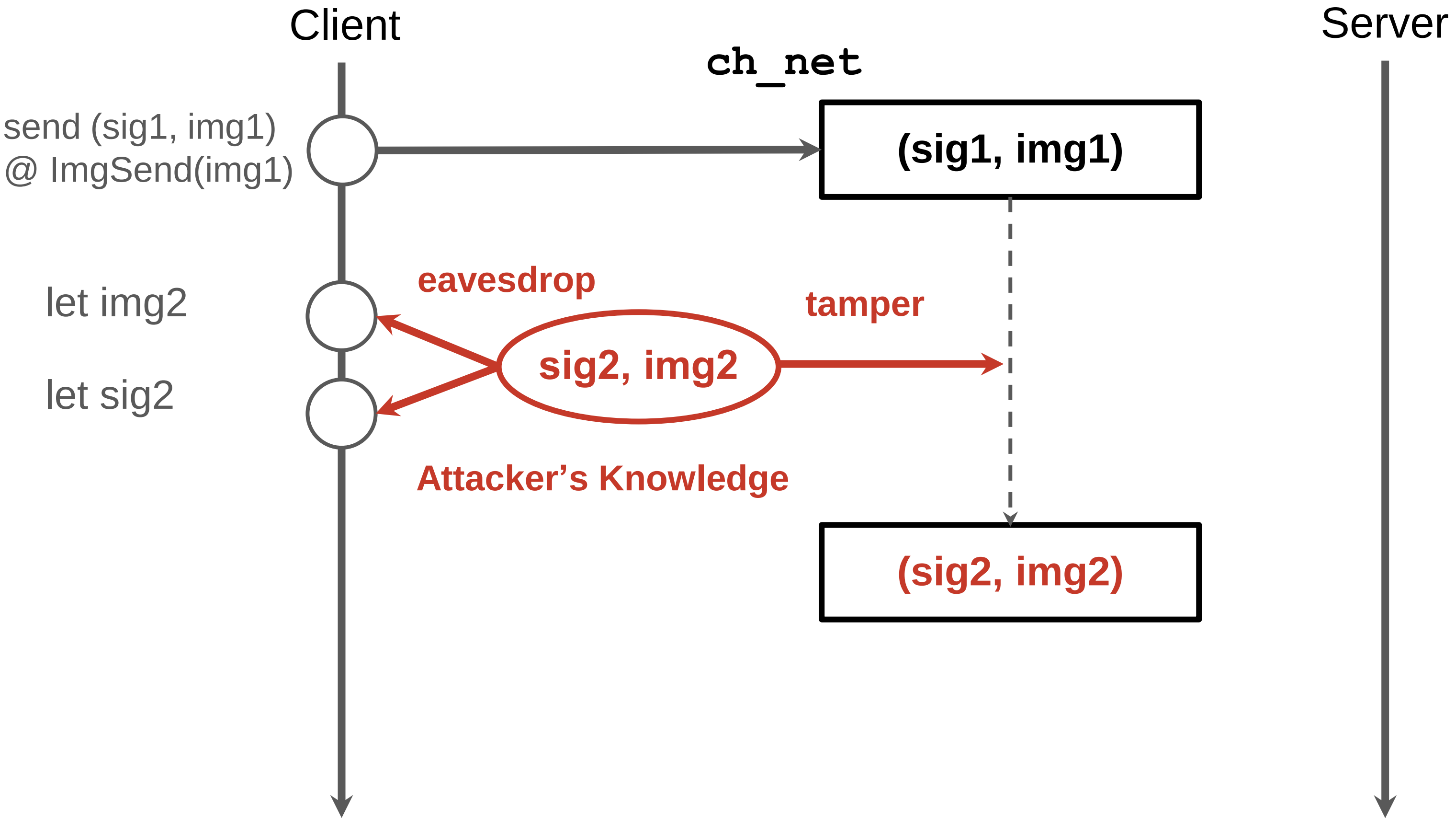
let sig2

sig2, img2

Attacker's Knowledge

Experiment 2: Falsified Trace





Client

Server

`ch_net`

send (sig1, img1)
@ ImgSend(img1)

(sig1, img1)

let img2

let sig2

eavesdrop

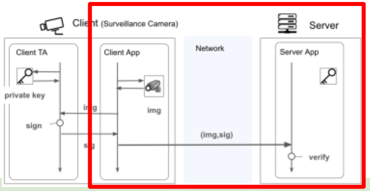
sig2, img2

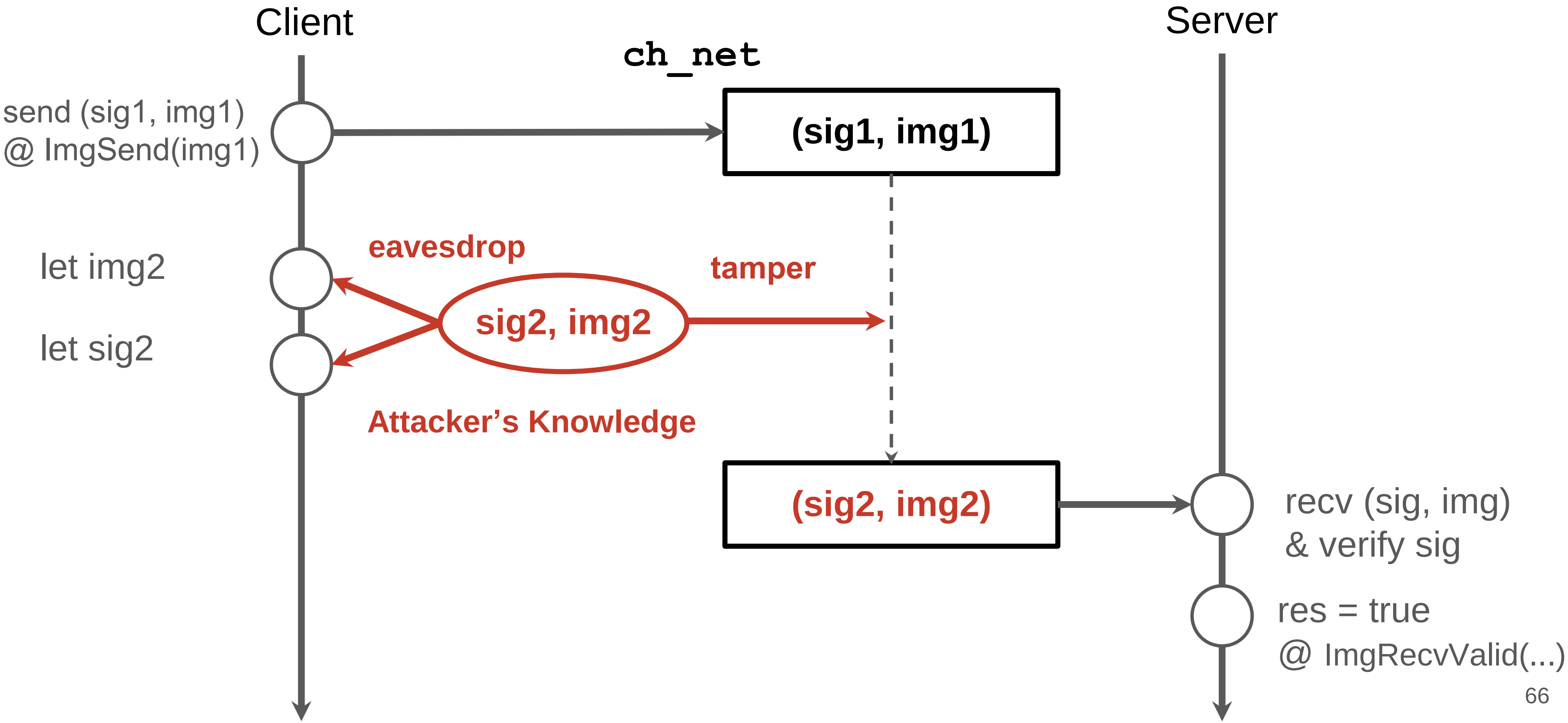
tamper

Attacker's Knowledge

(sig2, img2)

Experiment 2: Falsified Trace





Client

ch_net

Server

send (sig1, img1)
@ ImgSend(img1)

(sig1, img1)

let img2

let sig2

eavesdrop

sig2, img2

tamper

Attacker's Knowledge

(sig2, img2)

recv (sig, img)
& verify sig

res = true
@ ImgRecvValid(...)

Future Direction

On Modeling

- Other classes of objects, such as a shared memory
- Other permissions on objects, such as forking processes or executing files
- Dynamic update of access control policies
- Various communication protocols (e.g., TLS)

On Verification

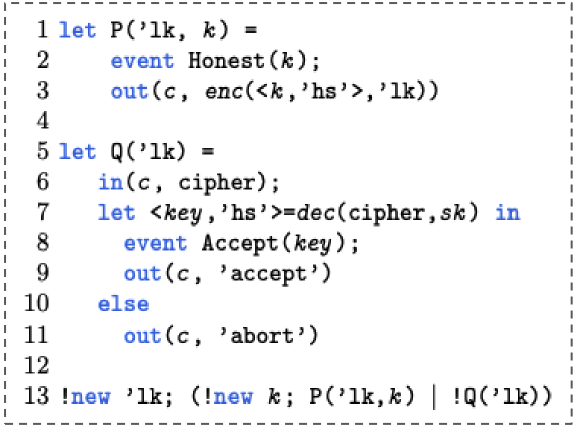
- Automatic translator
- Further encoding optimization using Tamarin’s advanced feature
- Conversion for multiple verifiers (c.f. SAPIC+)

Conclusions

- Rabbit, a modeling language for security verification on data flow
  - Easy to write for system programmers
  - Various IoT security solutions
  - Flexible specification of attacker models
- Rabbit’s Formal syntax and semantics
- Case study on a client-server system with TEE
- Validity of manual translation via experiments

Other Formal Verification Tools

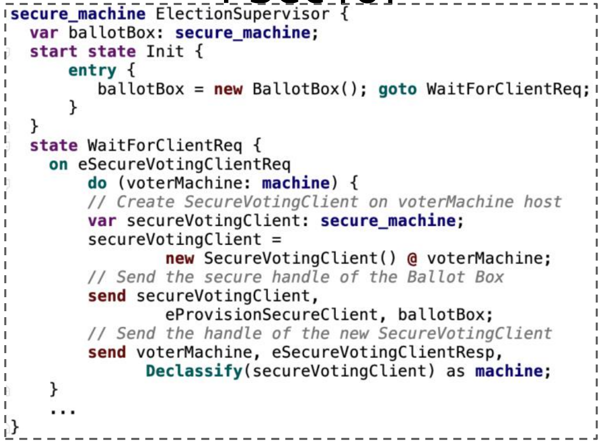
SAPIC+ [5]



```
let P(’lk, k) =
    event Honest(k);
    out(c, enc(<k,’hs’>,’lk))

let Q(’lk) =
   in(c, cipher);
   let <key,’hs’>=dec(cipher,sk) in
     event Accept(key);
     out(c, ’accept’)
   else
     out(c, ’abort’)

!new ’lk; (!new k; P(’lk,k) | !Q(’lk))
```

Limited attacker models, Unfamiliar styles for system programmers.

PSec [6]



```
secure_machine ElectionSupervisor {
  var ballotBox: secure_machine;
  start state Init {
      entry {
         ballotBox = new BallotBox(); goto WaitForClientReq;
      }
  }
  state WaitForClientReq {
    on eSecureVotingClientReq
        do (voterMachine: machine) {
        // Create SecureVotingClient on voterMachine host
        var secureVotingClient: secure_machine;
        secureVotingClient =
                new SecureVotingClient() @ voterMachine;
        // Send the secure handle of the Ballot Box
        send secureVotingClient,
                eProvisionSecureClient, ballotBox;
        // Send the handle of the new SecureVotingClient
        send voterMachine, eSecureVotingClientResp,
              Declassify(secureVotingClient) as machine;
    }
    ...
}
```

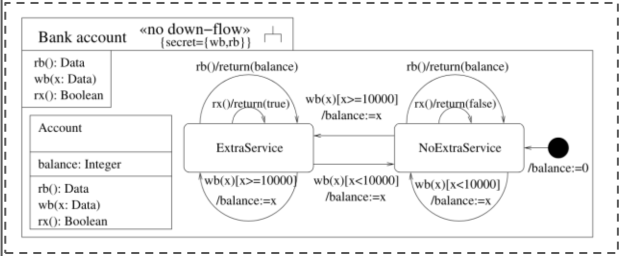
Limited attacker models

(Passive Network Observer, Active Man-in-the-Middle, Privileged Attacker)

UML-based Solutions

UML… Unified Modeling Language

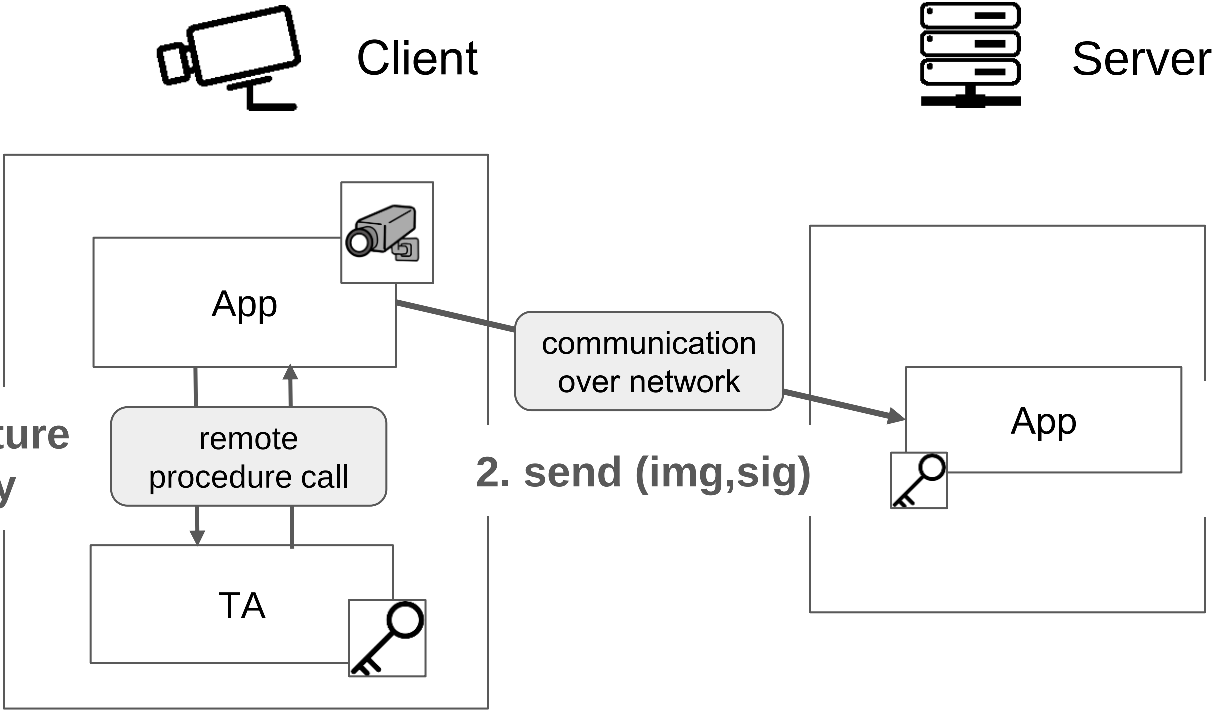
UMLSec [7]



No IoT solutions, Limited attacker models

[5] Kremer, S. and K¨unnemann, R.: Automated analysis of security protocols with global state, J. Comput. Secur., Vol. 24, No. 5, pp. 583–616 (2016).
[6] Kushwah, S., Desai, A., Subramanyan, P. and Seshia, S. A.: PSec: Programming Secure Distributed Systems Using Enclaves, Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, ASIA CCS ’21, New York, NY, USA, Association for Computing Machinery, pp. 802–816 (2021). event-place: Virtual Event, Hong Kong.
[7] Ju¨rjens, J.: UMLsec: Extending UML for Secure Systems Develop- ment, UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany, September 30 - October 4, 2002.

The Cam-Image System)



1. make signature with secret key

2. send (img,sig)

3.verify signature with public key

TA ··· Trusted Application (running in a secure execution environment)

Outline

- Rabbit Language
  - Modeling Overview
  - Rabbit Program
- Translation to Tamarin
- Experiments
  - Experiment 1: Reachabitlity
  - Experiment 2: Authenticity
- Conclusions

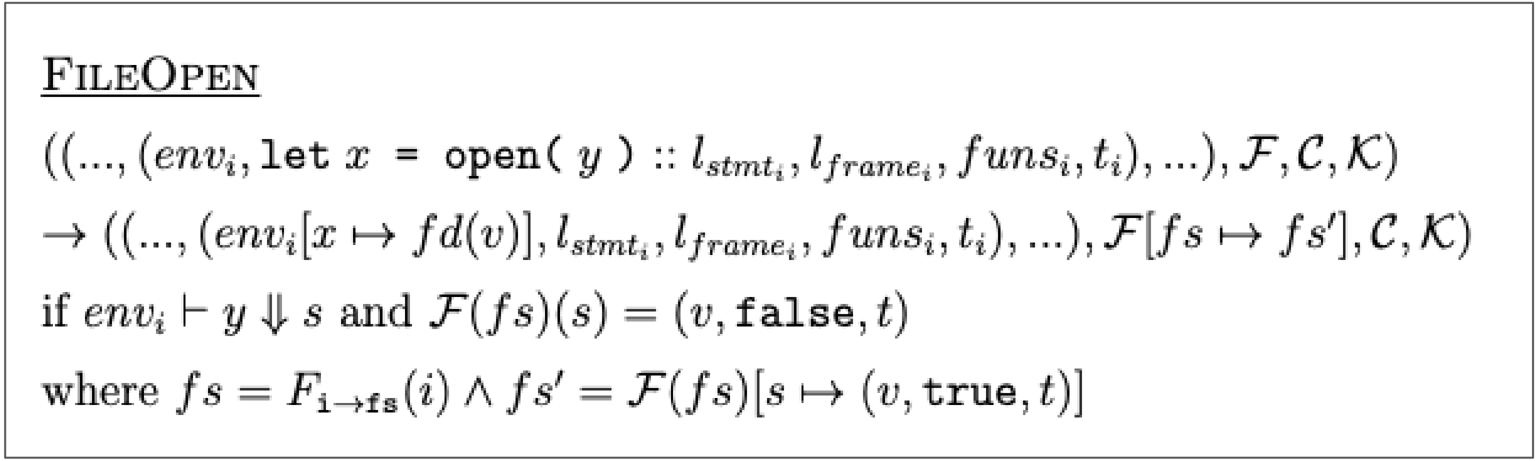
※ Explanations of Semantics, details of the Tamarin prover and Translation are omitted today.

Formal Semantics

We define semantics by state transitions caused by statements.

(*P, F, C, K*) ··· (*process state, file system state, channel state, attacker’s knowledge*)

Ex. **`open`** statement



FILEOPEN

$((\ldots,(env_i,\mathtt{let}\ x\ \mathtt{=}\ \mathtt{open(}\ y\ \mathtt{)} :: l_{stmt_i}, l_{frame_i}, funs_i, t_i),\ldots),\mathcal{F},\mathcal{C},\mathcal{K})$

$\rightarrow ((\ldots,(env_i[x \mapsto fd(v)], l_{stmt_i}, l_{frame_i}, funs_i, t_i),\ldots),\mathcal{F}[fs \mapsto fs'],\mathcal{C},\mathcal{K})$

if $env_i \vdash y \Downarrow s$ and $\mathcal{F}(fs)(s) = (v,\mathtt{false},t)$

where $fs = F_{\mathtt{i \rightarrow fs}}(i) \wedge fs' = \mathcal{F}(fs)[s \mapsto (v,\mathtt{true},t)]$

Overview of the Tamarin prover

The Tamarin prover [4] is a state-of-the-art tool for security of cryptographic protocols. It is used also for a real-world IoT system (Brun et al. 2023, [8])

A model in Tamarin is specified as **multiset rewriting rules**, which define a **labeled transition system**.

(an instance of) fact

```
rule ExampleRule :
  [ SomeFact(x), AnotherFact(y, z) ] --> [ NewFact(x, y, z) ]
```

Precondition **Postcondition**

[4] Schmidt, B., Meier,... : Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties, 2012 IEEE 25th Computer Security Foundations Symposium, pp. 78–94 (2012).
[8] Brun, L., Hasuo, I., Ono, Y. and Sekiyama, T.: Automated Security Analysis for Real-World IoT Devices, Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP ’23, New York, NY, USA, Association for Computing Machinery, pp. 29–37 (2023).

Overview of the Tamarin prover

The application of rules is recorded by adding instantiated ***action facts*** to the trace.

```
rule GenerateNonce:
  [ Fr(~n) ]
  --[ NonceGenerated(~n) ]->
  [ NonceAvailable(~n) ]
```

```
rule UseNonce:
  [ NonceAvailable(n) ]
  --[ NonceUsed(n) ]->
  [ NonceConsumed(n) ]
```

A ***lemma*** is a property to be verified in the system and is given on the trace.

```
lemma ExampleLemma :
  all-traces
  "All #i. NonceUsed(n) @ #i ==> Ex #j. NonceGenerated(n) @ #j"
```

Translation of Simple Statements

We manually translated the configuration & each statement in processes.

A statement is translated into multiple rules in Tamarin.

Ex. *send_datagram(ch_net, (sig, image)) @ ImgSend(image);* (at line 65)

```
rule Rule_Client_65_1 :
  [ F_Proc_Client_64_recv_1(...) ]
  --[ ImgSend(image_now_0) ]->
  [ F_Proc_Client_65_1(...)
    , Msg('ch_net', 's', '1', pair(sig_now_0, image_now_0)) ]
```

Translation of Simple Statements

We manually translated the configuration & each statement in processes.

A statement is translated into multiple rules in Tamarin.

Ex. *send_datagram(ch_net, (sig, image)) @ ImgSend(image);* (at line 65)

```
rule Rule_Client_65_1 :
  [ F_Proc_Client_64_recv_1(...) ]
  --[ ImgSend(image_now_0) ]->
  [ F_Proc_Client_65_1(...)
    , Msg('ch_net', 's', '1', pair(sig_now_0, image_now_0)) ]
```

Line number (program counter)

Translation of Simple Statements

We manually translated the configuration & each statement in processes.

A statement is translated into multiple rules in Tamarin.

Ex. *send_datagram(ch_net, (sig, image)) @ ImgSend(image);* (at line 65)

```
rule Rule_Client_65_1 :
  [ F_Proc_Client_64_recv_1(...) ]
  --[ ImgSend(image_now_0) ]->
  [ F_Proc_Client_65_1(...)
    , Msg('ch_net', 's', '1', pair(sig_now_0, image_now_0)) ]
```

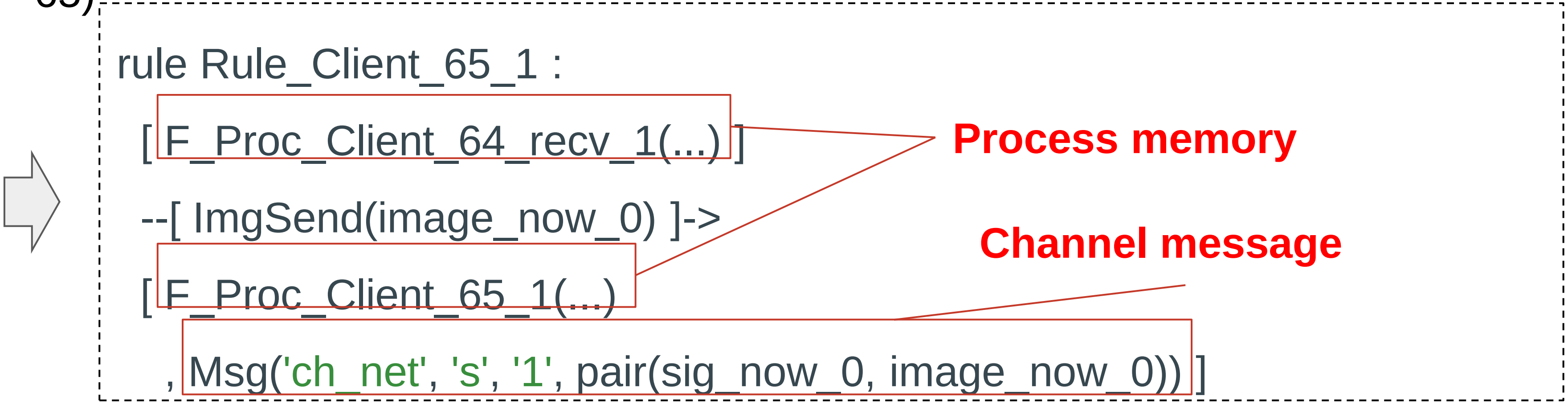
current iteration

Translation of Simple Statements

We manually translated the configuration & each statement in processes.

A statement is translated into multiple rules in Tamarin.

Ex. *send_datagram(ch_net, (sig, image)) @ ImgSend(image);* (at line 65)



```
rule Rule_Client_65_1 :
  [ F_Proc_Client_64_recv_1(...) ]
  --[ ImgSend(image_now_0) ]->
  [ F_Proc_Client_65_1(...)
    , Msg('ch_net', 's', '1', pair(sig_now_0, image_now_0)) ]
```

Process memory

Channel message

Translation of Simple Statements

We manually translated the configuration & each statement in processes.

A statement is translated into multiple rules in Tamarin.

Ex. *send_datagram(ch_net, (sig, image)) @ ImgSend(image);* (at line 65)

```
rule Rule_Client_65_1 :
  [ F_Proc_Client_64_recv_1(...) ]
  --[ ImgSend(image_now_0) ]->
  [ F_Proc_Client_65_1(...)
    , Msg('ch_net', 's', '1', pair(sig_now_0, image_now_0)) ]
```

Event (Action fact)

Translation of Attacker’s Behavior

Attacker receives any messages from the ***Out*** facts.

Ex. eavesdropping on memory

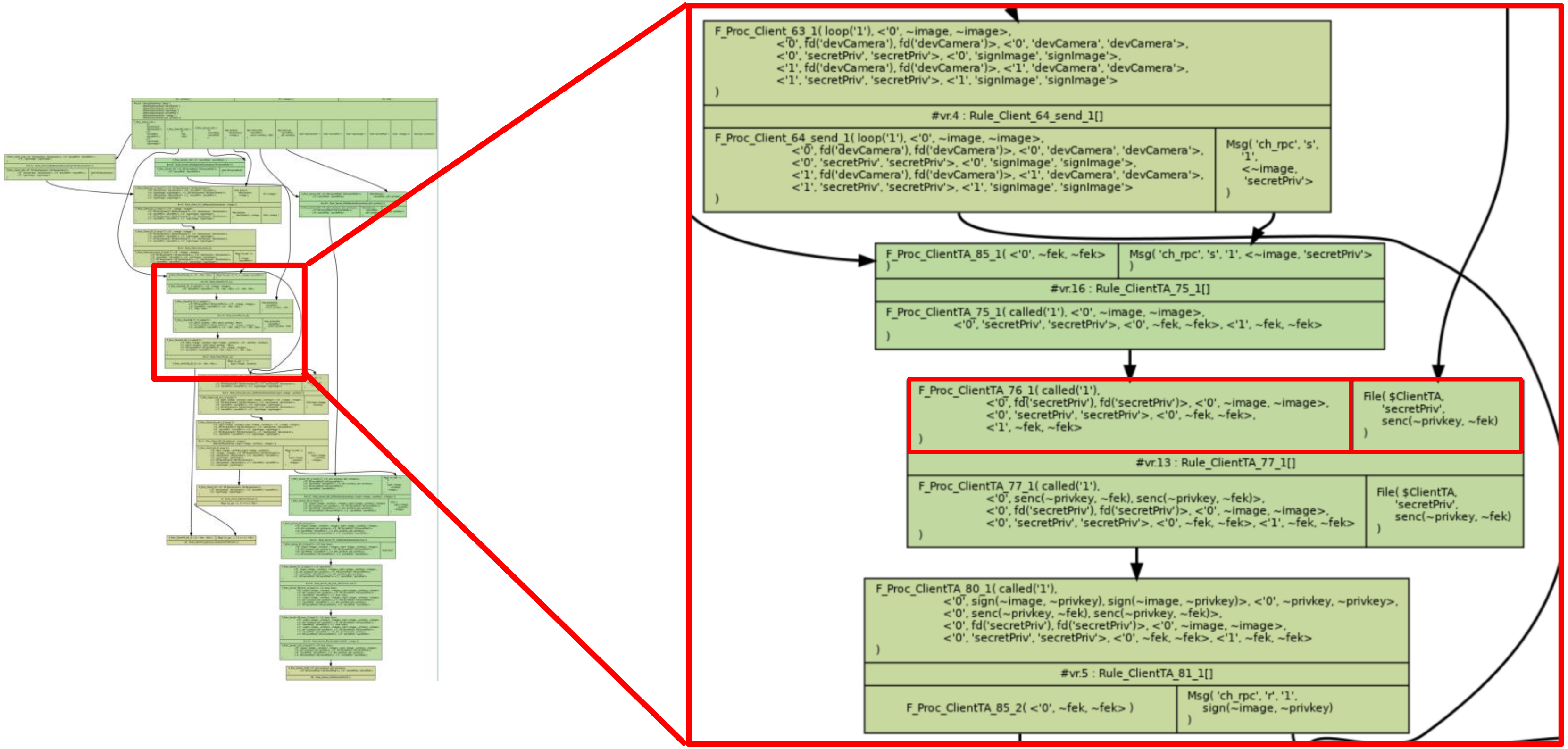
```
rule Rule_Server_93 :
  [ F_Proc_Server_init(...) ]-->
  [ F_Proc_Server_93(<'0', fd(pubkeyPath_now_0), …>)
    , Out(fd(pubkeyPath_now_0))]
```

Attacker generates the system inputs in the ***In*** facts.

Ex. tampering on channel messages (`Msg`)

```
rule Rule_Server_96 :
[ F_Proc_Server_95 (...), Msg(’ch_net’, ’s’, %1, pair(sig, image)) ] -->
[ F_Proc_Server_96(<’0’, pair(sig, image), pair(sig, image)>, ...) ]

rule Rule_Server_96_tampered :
[ F_Proc_Server_95 (...), Msg(’ch_net’, ’s’, %1, pair(sig, image)), In(x) ]
--> [ F_Proc_Server_96(<’0’, x, x>, ...) ]
```

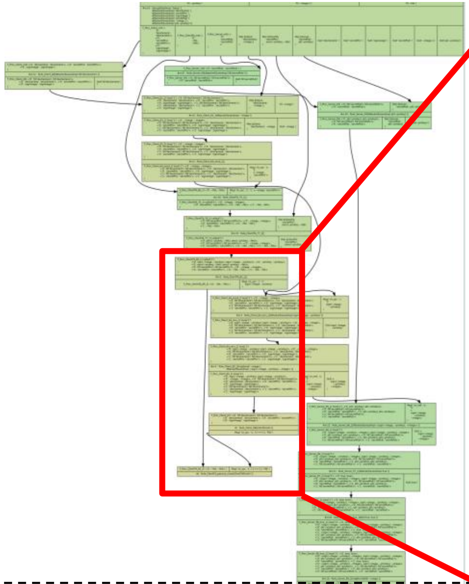
Visualization of Searching Algorithm

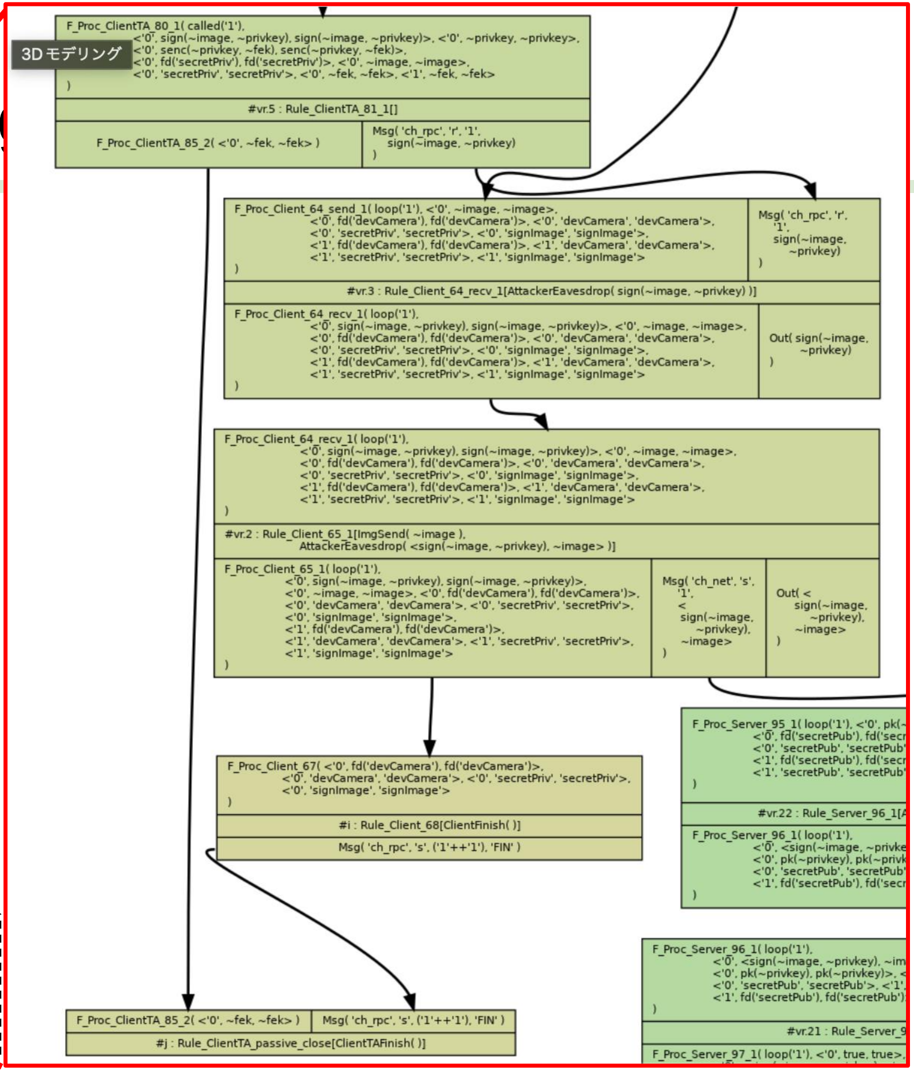


```
F_Proc_Client_63_1( loop('1'), <'0', ~image, ~image>,
            <'0', fd('devCamera'), fd('devCamera')>, <'0', 'devCamera', 'devCamera'>,
            <'0', 'secretPriv', 'secretPriv'>, <'0', 'signImage', 'signImage'>,
            <'1', fd('devCamera'), fd('devCamera')>, <'1', 'devCamera', 'devCamera'>,
            <'1', 'secretPriv', 'secretPriv'>, <'1', 'signImage', 'signImage'>
)

#vr.4 : Rule_Client_64_send_1[]

F_Proc_Client_64_send_1( loop('1'), <'0', ~image, ~image>,
            <'0', fd('devCamera'), fd('devCamera')>, <'0', 'devCamera', 'devCamera'>,
            <'0', 'secretPriv', 'secretPriv'>, <'0', 'signImage', 'signImage'>,
            <'1', fd('devCamera'), fd('devCamera')>, <'1', 'devCamera', 'devCamera'>,
            <'1', 'secretPriv', 'secretPriv'>, <'1', 'signImage', 'signImage'>
)

Msg( 'ch_rpc', 's',
   '1',
   <~image,
    'secretPriv'>
)

F_Proc_ClientTA_85_1( <'0', ~fek, ~fek>
)

Msg( 'ch_rpc', 's', '1', <~image, 'secretPriv'>
)

#vr.16 : Rule_ClientTA_75_1[]

F_Proc_ClientTA_75_1( called('1'), <'0', ~image, ~image>,
            <'0', 'secretPriv', 'secretPriv'>, <'0', ~fek, ~fek>, <'1', ~fek, ~fek>
)

F_Proc_ClientTA_76_1( called('1'),
            <'0', fd('secretPriv'), fd('secretPriv')>, <'0', ~image, ~image>,
            <'0', 'secretPriv', 'secretPriv'>, <'0', ~fek, ~fek>,
            <'1', ~fek, ~fek>
)

File( $ClientTA,
   'secretPriv',
   senc(~privkey, ~fek)
)

#vr.13 : Rule_ClientTA_77_1[]

F_Proc_ClientTA_77_1( called('1'),
            <'0', senc(~privkey, ~fek), senc(~privkey, ~fek)>,
            <'0', fd('secretPriv'), fd('secretPriv')>, <'0', ~image, ~image>,
            <'0', 'secretPriv', 'secretPriv'>, <'0', ~fek, ~fek>, <'1', ~fek, ~fek>
)

File( $ClientTA,
   'secretPriv',
   senc(~privkey, ~fek)
)

F_Proc_ClientTA_80_1( called('1'),
            <'0', sign(~image, ~privkey), sign(~image, ~privkey)>, <'0', ~privkey, ~privkey>,
            <'0', senc(~privkey, ~fek), senc(~privkey, ~fek)>,
            <'0', fd('secretPriv'), fd('secretPriv')>, <'0', ~image, ~image>,
            <'0', 'secretPriv', 'secretPriv'>, <'0', ~fek, ~fek>, <'1', ~fek, ~fek>
)

#vr.5 : Rule_ClientTA_81_1[]

F_Proc_ClientTA_85_2( <'0', ~fek, ~fek> )

Msg( 'ch_rpc', 'r', '1',
   sign(~image, ~privkey)
)
```

Translation from Rabbit to Tamarin

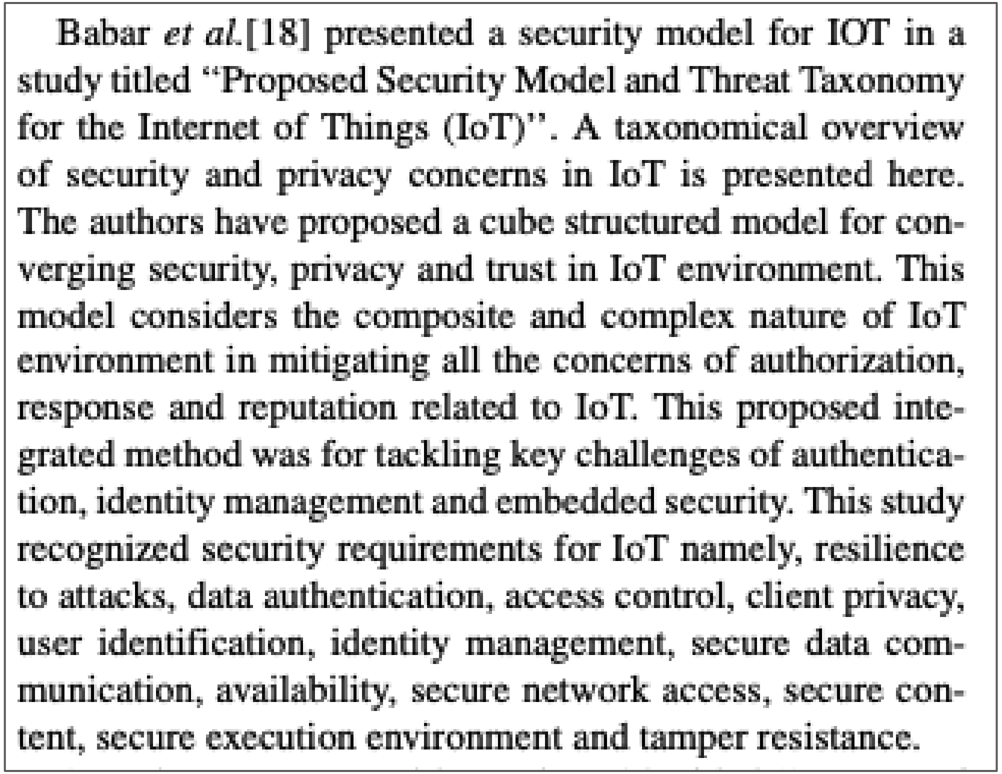
Visualization of Searching Al





lemma Finish :
  exists-trace
  "Ex #i #j #k. ClientFin() @ #i & ClinentTAFin() @ #j … “

Security Requirements for IoT



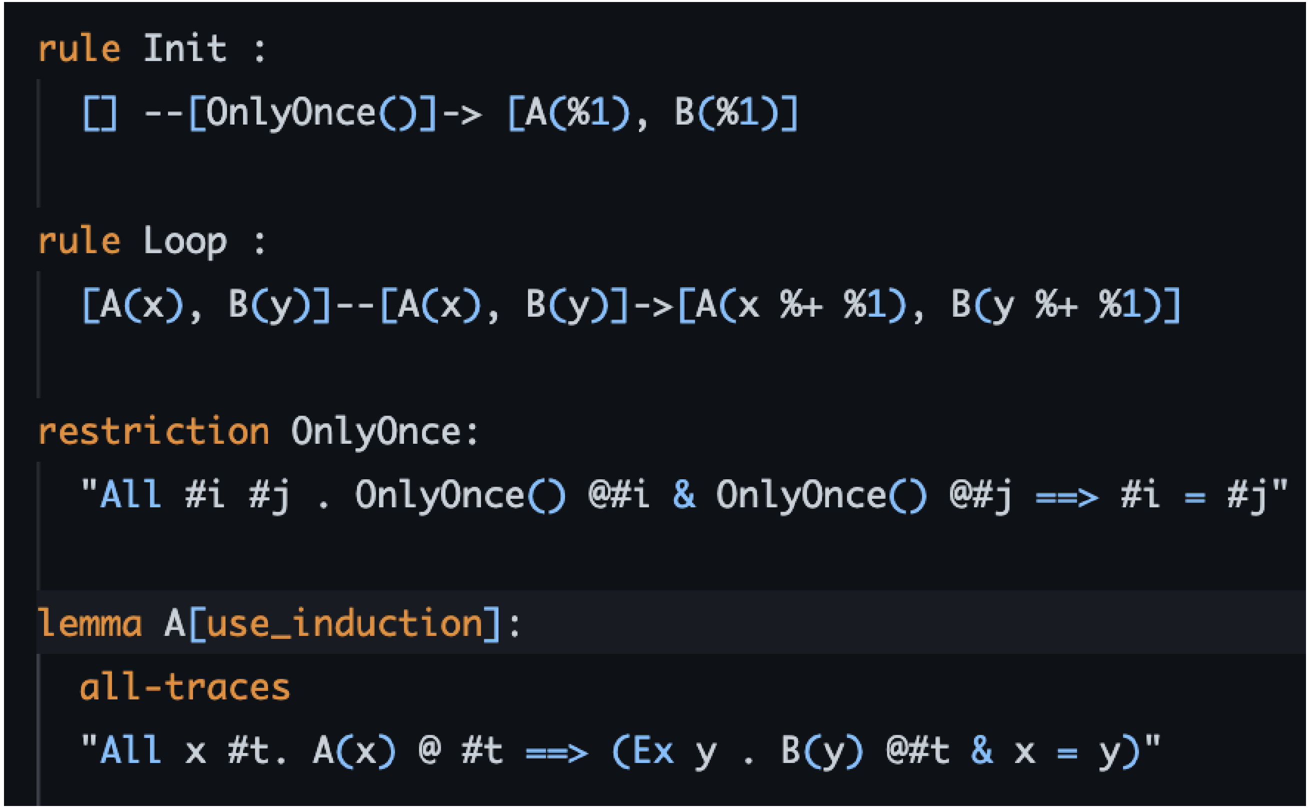
Babar *et al.*[18] presented a security model for IOT in a study titled "Proposed Security Model and Threat Taxonomy for the Internet of Things (IoT)". A taxonomical overview of security and privacy concerns in IoT is presented here. The authors have proposed a cube structured model for converging security, privacy and trust in IoT environment. This model considers the composite and complex nature of IoT environment in mitigating all the concerns of authorization, response and reputation related to IoT. This proposed integrated method was for tackling key challenges of authentication, identity management and embedded security. This study recognized security requirements for IoT namely, resilience to attacks, data authentication, access control, client privacy, user identification, identity management, secure data communication, availability, secure network access, secure content, secure execution environment and tamper resistance.

Security Requirements for IoT

- resilience to attacks
- data authentication
- access control
- secure data communication
- availability
- secure network access
- secure content
- secure execution environment
- tamper resistance

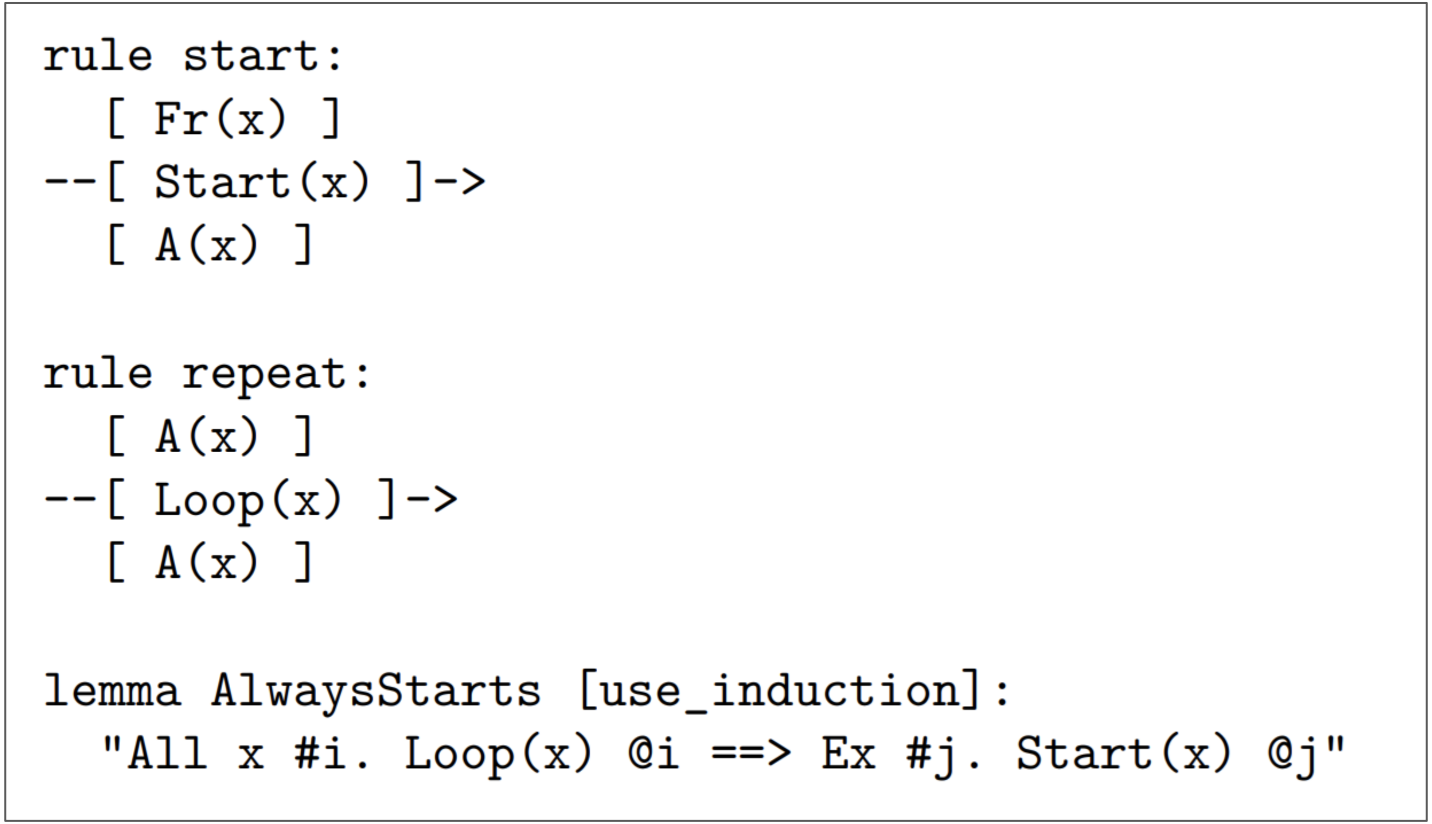
[6] T. A. Ahanger and A. Aljumah, "Internet of Things: A Comprehensive Study of Security Issues and Defense Mechanisms," in IEEE Access, vol. 7, pp. 11020-11028, 2019, doi: 10.1109/ACCESS.2018.2876939.

Never-Ending Example



```
rule Init :
  [] --[OnlyOnce()]-> [A(%1), B(%1)]

rule Loop :
  [A(x), B(y)]--[A(x), B(y)]->[A(x %+ %1), B(y %+ %1)]

restriction OnlyOnce:
  "All #i #j . OnlyOnce() @#i & OnlyOnce() @#j ==> #i = #j"

lemma A[use_induction]:
  all-traces
  "All x #t. A(x) @ #t ==> (Ex y . B(y) @#t & x = y)"
```

Induction in Tamarin



```
rule start:
  [ Fr(x) ]
--[ Start(x) ]->
  [ A(x) ]

rule repeat:
  [ A(x) ]
--[ Loop(x) ]->
  [ A(x) ]

lemma AlwaysStarts [use_induction]:
  "All x #i. Loop(x) @i ==> Ex #j. Start(x) @j"
```

References

- Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. and Ayyash, M.: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications, IEEE Communications Surveys & Tutorials, Vol. 17, No. 4, pp. 2347–2376 (2015).
- Kumar, S., Tiwari, P. and Zymbler, M. L.: Internet of Things is a revo- lutionary approach for future technology enhancement: a review, J. Big Data, Vol. 6, p. 111 (2019).
- Masys, A.: Security by Design: Innovative Perspectives on Complex Prob- lems, Advanced Sciences and Technologies for Security Applications, Springer International Publishing (2018).
- Schmidt, B., Meier, S., Cremers, C. and Basin, D.: Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties, 2012 IEEE 25th Computer Security Foundations Symposium, pp. 78–94 (2012).
- Kremer, S. and K¨unnemann, R.: Automated analysis of security protocols with global state, J. Comput. Secur., Vol. 24, No. 5, pp. 583–616 (2016).
- Kushwah, S., Desai, A., Subramanyan, P. and Seshia, S. A.: PSec: Programming Secure Distributed Systems Using Enclaves, Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, ASIA CCS ’21, New York, NY, USA, Association for Computing Machinery, pp. 802–816 (2021). event-place: Virtual Event, Hong Kong.
- Ju¨rjens, J.: UMLsec: Extending UML for Secure Systems Develop- ment, UML 2002 - The Unified Modeling Language, 5th International Conference, Dresden, Germany, September 30 - October 4, 2002, Pro- ceedings (Je´ze´quel, J.-M., Hußmann, H. and Cook, S., eds.), Lecture Notes in Computer Science, Vol. 2460, Springer, pp. 412–425 (on- line), https://doi.org/10.1007/3-540-45800-X_32 (2002).
- Brun, L., Hasuo, I., Ono, Y. and Sekiyama, T.: Automated Security Analysis for Real-World IoT Devices, Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP ’23, New York, NY, USA, Association for Computing Machinery, pp. 29–37 (2023).