データフローセキュリティの検証のための IoT システムモデリング言語 Rabbit

稲葉皓信*　五十嵐淳*　石川裕**　関山太朗**

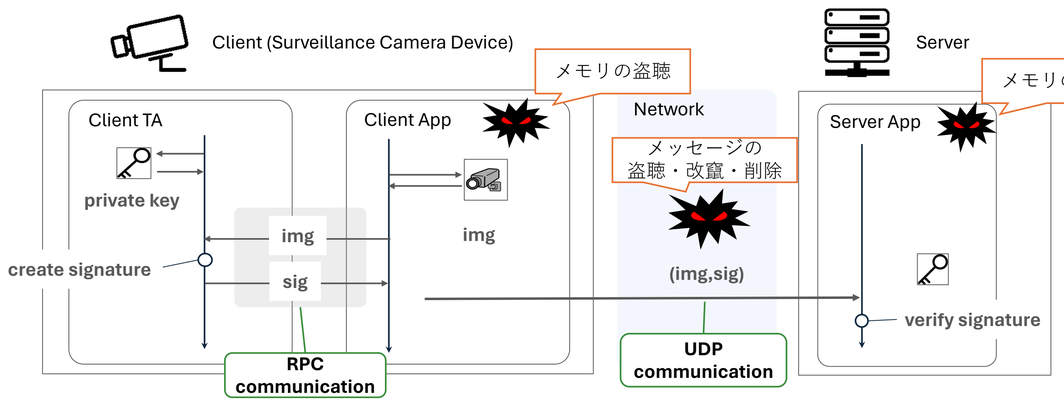
* 京都大学情報研究科　**国立情報学研究所

Rabbit 言語

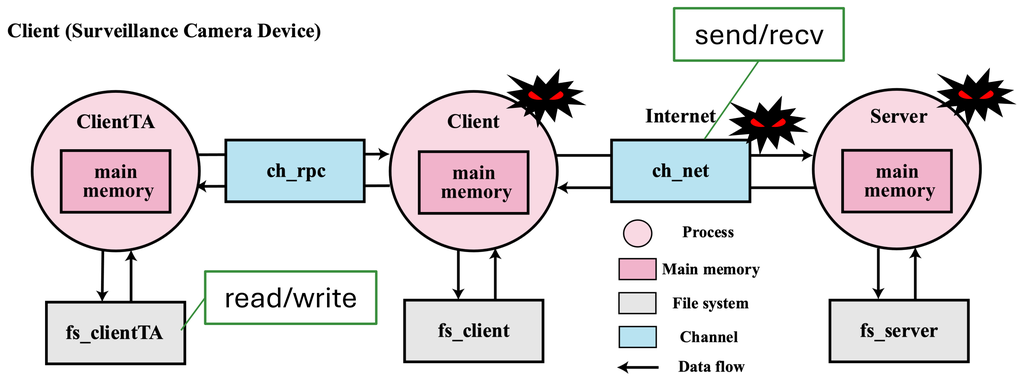
脅威分析では **データフローレベル** でシステムを抽象化し、Security Weaknessを洗い出す。
Rabbitはシステムの「**モデル化**」と調べたい性質の「**形式検証**」を一貫して行うモデリング言語。

性質を数学的に厳密に保証👍

プロセス等のシステムプログラミングの概念でシステムをモデル化　攻撃者は各コンポーネントに細かく指定



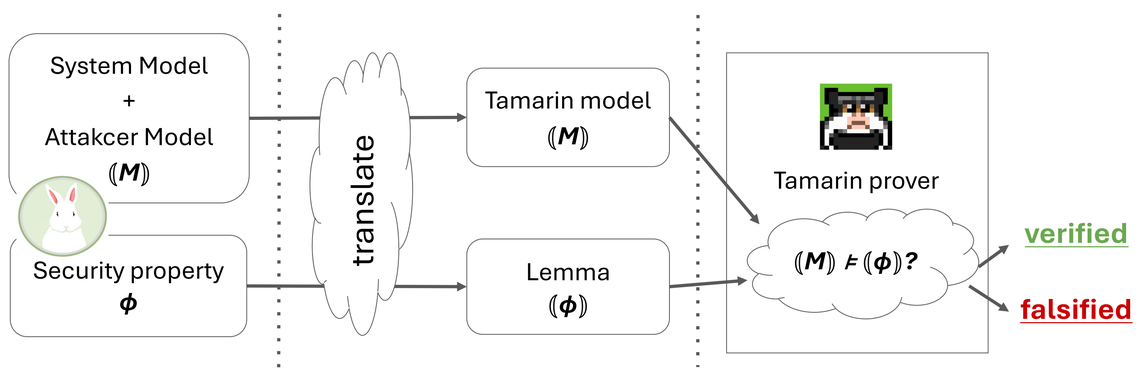
Model



システムの振る舞いを**システムプログラミング的に記述できる**他、
TEE・アクセス制御・暗号文のIoT要件を記述可能なのが特徴

	Familiar to system programmer	IoT Security Solutions	Flexibility of Attacker models	Other features
Rabbit	◎	○	○	–
Tamarin prover [1]	×	△	○	Unbounded Sessions
SAPIC+ [2]	△	○	△	Reducible to many verification tools
PSec [3]	△	○	△	Programming language
UML extensions [4]	○	△	△	Visualizer

Rabbitで記述したシステムモデル・攻撃者モデル・性質は
プロトコル検証器 Tamarin prover [1] のモデルに変換して検証



Rabbit プログラム

構文・意味論を形式的に定義

```
// access control type
type client_t:process
type clientTA_t:process
type server_t:process
type client_file_t:file
type clientTA_file_t:file
type server_file_t:file
type chan_net_t:channel
type chan_rpc_t:channel

// allow rules for regular processes
allow client_t client_file_t [read]
allow client_ta_t clientTA_file_t [read]
allow server_t server_file_t [read, write]
allow client_t client_net_t [send, recv]
allow server_t client_net_t [send, recv]
allow client_t client_rpc [send, recv]
allow clientTA_t clinet_rpc_t [send, recv]
```

```
// Allow rules for attacker process
allow attacker_t client_t [ eavesdrop ]
allow attacker_t server_t [ eavesdrop ]
allow attacker_t client_file_t [ eavesdrop ]
allow attacker_t server_file_t [ eavesdrop ]
allow attacker_t chan_net_t [ eavesdrop, tamper, drop ]

// file system declaration
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 declaration
channel ch_net = { connection: datagram, type: chan_net_t }
channel ch_rpc = { connection: stream, type: chan_rpc_t }
```

```
// process declaration
process Client(ch_net, ch_rpc) with client_t {
  let dev_path = "/dev/camera";
  let privkey_path = "/secret/priv";
  let invoke_func = "sign_image";

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

process ClientTA(ch_rpc) with clientTA_t {
  let fek = sym_kt;

  func sign_image(image, privkey_path) {
    let privkey_fd = open(privkey_path);
    let privkey = read(privkey_fd);
    let privkey0 = dec(privkey, fek);
    let sig = sign(image, privkey0);
    return sig;
  }

  main {
    let conn = accept(ch_rpc);
  }
}

process Server(ch_net) with server_t {
  let pubkey_path = "/secret/pub";

  main {
    let pubkey_fd = open(pubkey_path);
    let pubkey = read(pubkey_fd);
    for i in range(1, 4) {
      let p = recv(ch_net);
      let res = verify(p.fst, p.snd, pubkey);
      if (res) {
        skip @ ImgRecvValid(p.snd);
      } else {
        skip @ ImgRecvInvalid(p.snd);
      }
    }
  }
}
```

Linuxシステムコールを模倣したライブラリ関数

基本的な制御構文（For文・If文）

RPC通信用のライブラリ関数（invoke）

検証用の性質の指定に使うイベント発火

署名等の暗号プリミティブ

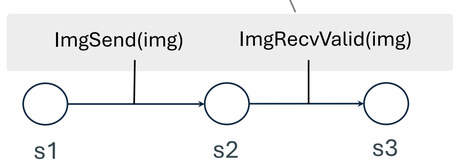
署名の検証（＋対応するイベント発火）

定義されたコンポーネントをインスタンス化（示したい性質も指定）

```
// 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 ""
]
```

Tamarinへの変換

Tamarinはシステムの状態遷移を項書き換え規則で逐一記述。
状態遷移時にイベントを発火し、時系列的なイベントの列について性質を論理式で指定・検証。



let image = read(image_fd); の規則

```
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>
  )
 ]
```

検証実験

上記のシステムモデルに加えて攻撃者モデルをパラメタライズして、
ある種の真正性（correspondence assertion）を検証。

Serverが署名に検証した画像は全てClientが事前に送信したものである

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

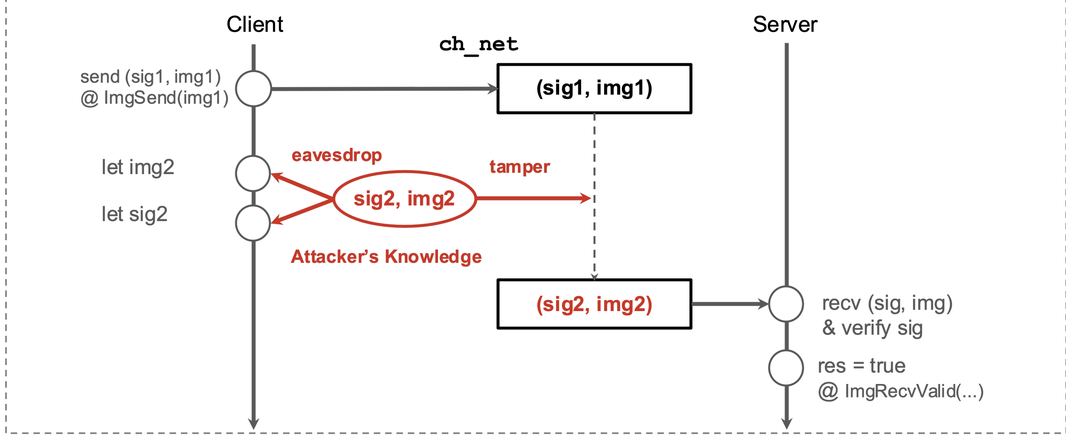
検証結果

	-	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

※N ... ループのイテレーション数
※e ... メッセージの盗聴, t ... メッセージの改竄, d ... メッセージの削除, etdなどはそのコンビネーション

verified　falsified

Tamarinが出した反例
→ Serverが署名の検証に成功したにも拘らずClientは画像を送信していない



今後の課題

- モデリング能力の向上：プロセス・チャネル・ファイルへの操作（exec, create等）
- 検証の高速化：Tamarinへのエンコーディングの最適化・別検証ツール入力への変換

[1] 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).
[2] Kremer, S. and Künnemann, R.: Automated analysis of security protocols with global state, J. Comput. Secur., Vol. 24, No. 5, pp. 583–616 (2016).
[3] 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.
[4] 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.