AVATAR: A SysML Environment for the Formal Verification of Safety and Security Properties

G. Pedroza L. Apvrille D. Knorreck

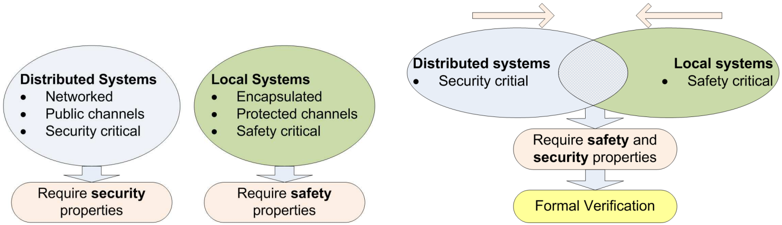
System-on-Chip laboratory (LabSoC), Institut Telecom
Telecom ParisTech, LTCI CNRS

9-12 May / NOTERE 2011

Outline

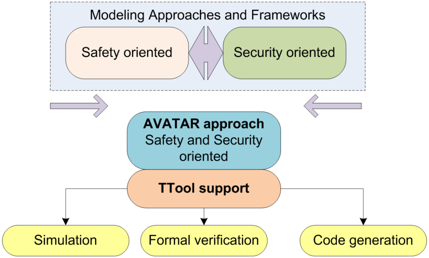
Context
main concerns

Tendency in embedded systems design



Context
main concerns

- Modeling approaches target either
  - safety properties or
  - security properties
- Safety and security models should be maintained
- Consistency problems



Methodology Phases

Overview

Requirements capture: Models requirements to be satisfied, e.g. with Requirement Diagrams

System analysis: Analysis of system behavior, e.g. using Sequence Diagrams

System design: Captures system behavior, e.g. with AVATAR profile

Property modeling: Captures properties to be verified, e.g. in TEPE Diagrams [1]

Formal verification: Formal proves over targeted properties

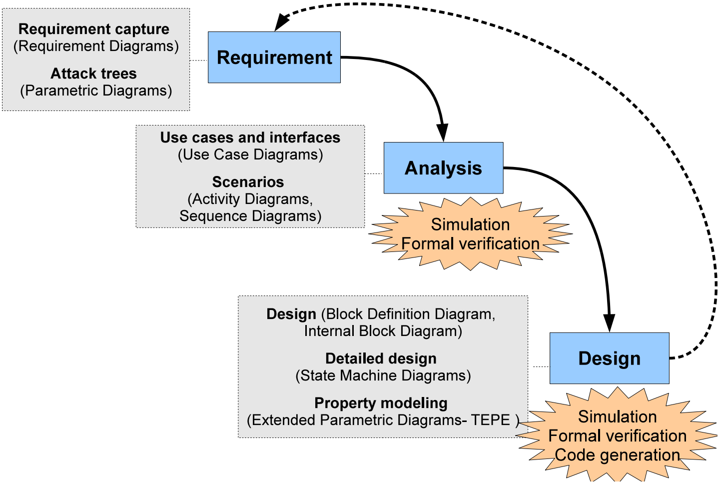
Refinement: Repeat previous stages adding system elements up to final design

---

[1] See [1]

Methodology Phases

Implementation

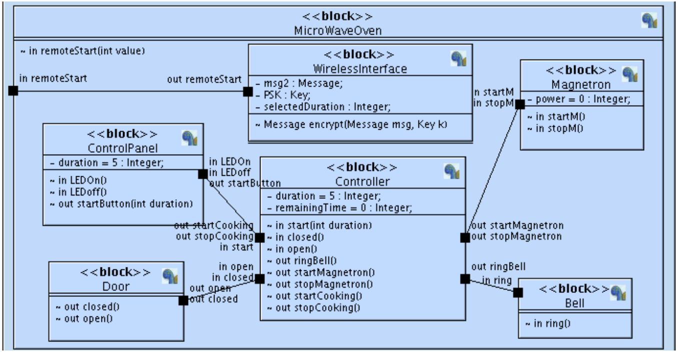


AVATAR
In a Nutshell

- SysML environment supporting all methodological phases
- Graphical capture of properties
- Integrated simulation
- Safety and **security** proofs at the push of a button
- C-POSIX code generation

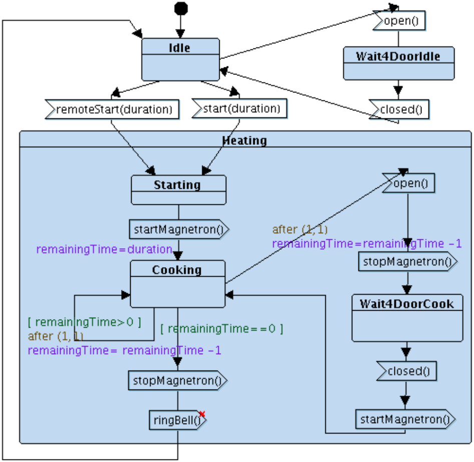
Design: Architecture

- SysML Block Definition and Internal Block Diagrams
- Block = attributes, methods, in/out signals, behaviour



Detailed Design

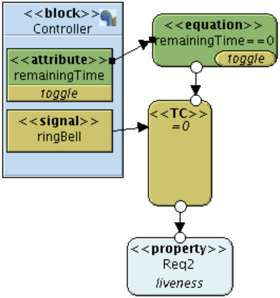
- Block's behaviour is described in terms of SysML State Machine Diagrams
- Non deterministic choices
- Non deterministic temporal operators



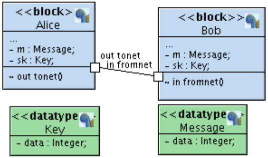
Property Modeling

Security properties

- Customized Parametric Diagrams (TEPE)



Profile limitations
to address security



Initial knowledge: No way to preshare data between blocks

Cryptographic functions: Not predefined and should be modelled

Communication Architecture: Channels can not be eavesdropped

Attacker model: Not included and not easily representable

Security properties: Not easily representable

AVATAR for security

implemented extensions

Initial knowledge: Introduced as a common knowledge by the pragma:
#InitialCommonKnowledge *Alice.sk Bob.sk*

Cryptographic functions: Predefined in each crypto block: *MAC(), encrypt(), decrypt(), sign(), verifyMAC(), verifySign()...*

Communication Architecture: Common broadcast channels can be defined in blocks. Attackers can eavesdrop public ones

Attacker model: Taken from the underlying security framework *ProVerif*[2]

2

AVATAR for security
implemented extensions

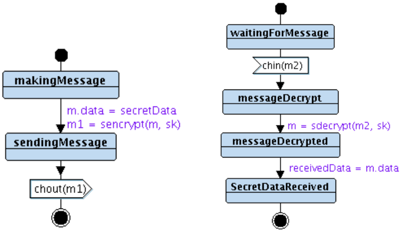
Security properties: **Confidentiality** of *data* intended to be secret is captured in the pragma:

***#Confidentiality** Alice.sk*

Authenticity of a block exchange is captured in the pragma:

***#Authenticity** Alice.sendingMessage.m1*

Bob.messageDecrypted.m2



a – Alice   b – Bob

Why ProVerif?

as underlying formal framework

- Proverif is [3] ...
  - quite generic: targets communicating systems modeling in general
  - completely automated
  - well suited for communicating entities (CEs) modeling:
    - based upon process algebras
    - CEs represented as pi-processes
  - oriented to prove security properties:
    - confidentiality
    - authenticity
  - endowed with an attacker targeting security properties
  - supported by a rigorous formal approach
  - implemented with a resolution algorithm

[3] See [2]

Processes
in ProVerif

Syntax of the process calculus [4]

`M,N::=`		terms
	`x,y,z`	variables
	`a,b,c,k`	names
	`f(`$M_1 \ldots M_n$`)`	constructor application
`P,Q::=`		processes
	`out(c,M); P`	outputs *M* in *c* then *P*
	`in(c,M); P`	inputs *M* from *c* then *P*
	`new a; P`	defines *a* restricted to *P*
	`event myEvt(x); P`	executes an event *myEvt(x)* then *P*
	`let x=g(`$M_1 \ldots M_n$`) in P else Q`	destructor application
	`if M=N then P else Q`	conditional
	`P|Q`	parallel composition of processes *P*, *Q*
	`!P`	infinite replication of process *P*
	`0`	null process

[4] See [3]

Properties
in ProVerif

Defined as queries [5]:

Confidentiality: can the attacker disclose secret data -*mySecret*?

```
query attacker:mySecret
```

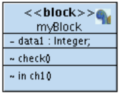
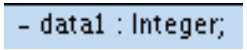
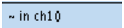
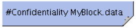
Authenticity: can the attacker break the receiver-sender correspondence?

```
query evinj:eventReceiveM(x)==>evinj:eventSendM(x)
```

[5] See [3]

AVATAR block diagram to ProVerif

Translation rules

AVATAR	ProVerif	Semantics
Block declaration 	`let` $myBlock_0$=	The initial process $myBlock_0$
Block data types 	`new` *`data1`*`;`	The new name *`data1`*
Block input signal 	`free` *`ch1`*.	The input channel *`ch1`*. Quite similar rule for *`output`* or *`private`* channel
Common knowledge *val* in blocks $\{B_i\}$ 	`new` *`val`*`;`	The static variable *`val`* is only known by the processes $\{Block_i\}$, *i=1...n*
Confidentiality pragma 	`query attacker:` *`data`*.	*`data`* confidentiality will be proved

Formal Proves

Security Properties

Performed at a push of a button!

Here the process[6]:

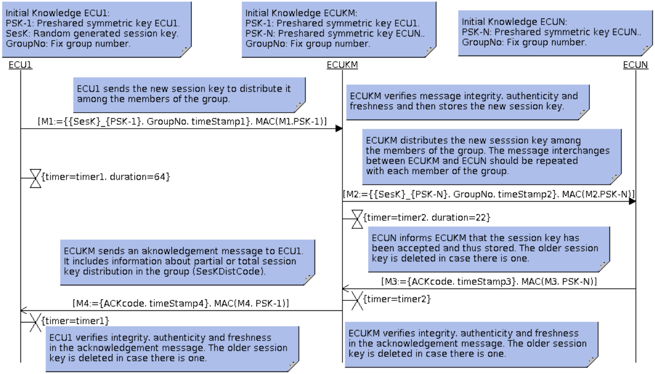
1. Translation of AVATAR model to ProVerif
2. Gained attacker knowledge in form of Horn clauses
3. Confidentiality; for each `query attacker:mySecret` the attacker:
   - builds a finite inference space of horn clauses
   - searches whether `mySecret` can be inferred
4. Authenticity; for each `query evinj:eventReceiveM(x)==>evinj:eventSendM(x)` the attacker:
   1. test all input channels
   2. acts on behalf of sender - or receiver - in replicated sessions
   3. builds a finite space of horn clauses
   4. proves sender-receiver correspondence

[6] See [4]

Keying Protocol

EVITA project

Security Goal: distribute a new secret key amongst members of a group of in-car Electronic Control Units (ECUs).[7]

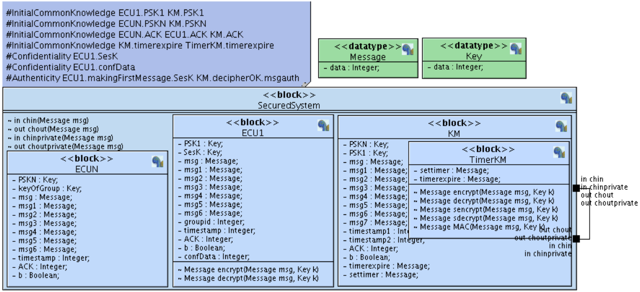


[7] See [5], [6]

AVATAR model of Keying Protocol

EVITA project

- Each ECU associated to a crypto block
- Public input and output channels
- Predefined crypto functions are used: *sencrypt(), sdecrypt, MAC(), etc.*
- Confidentiality of *SesK* and mutual authentication (`<auth>`) of ECUs is required



Results
Security proofs

Keying Protocol verification results[8]

Verification Scheme	AVATAR model and pragmas
Modeled Blocks	*ECU1, ECUKM, ECUN*
Verified confidential data	*SesK, PSK-1, PSK-N*
Verified authenticity correspondences	*ECU1*<auth>*ECUKM*, *ECUKM*<auth>*ECUN*
RESULT #*Confidentiality* B.dat	True for each verified data *dat*
RESULT #*Authenticity* B_d.Send.Mx B_o.Valid.Mx	True for each correspondence and message *Mx*
Observations	Keying Protocol preserves data confidentiality and authenticity.

8 See [7]

Conclusions

- AVATAR:
  - eases embedded systems modeling
  - easily proves safety and security properties
  - avoids models consistency maintainability
  - is fully supported by TTool[9]
  - has been tested in industrial projects
- However:
  - only targets confidentiality and authenticity
  - not suitable for temporal analyses
  - richer notions of attackers are required

[9] See [8]

Next steps
to do

- introduce temporal analyses capabilities
- introduce richer notions of attackers:
  - to prove message integrity
  - to prove message freshness
- automatic code generation from models
- code maintainability
- addapt tool support

Thanks

References

D. Knorreck, L. Apvrille, and P. D. Saqui-Sannes, “TEPE: A SysML language for timed-constrained property modeling and formal verification,” in *Proceedings of the UML&Formal Methods Workshop (UML&FM)*, (Shanghai, China), November 2010.

B. Blanchet, “Proverif automatic cryptographic protocol verifier user manual,” tech. rep., CNRS, Département d’Informatique École Normale Supérieure, Paris, July 2010.

B. Blanchet, “From Secrecy to Authenticity in Security Protocols,” in *9th International Static Analysis Symposium (SAS’02)* (M. Hermenegildo and G. Puebla, eds.), vol. 2477 of *Lecture Notes on Computer Science*, (Madrid, Spain), pp. 342–359, Springer Verlag, Sept. 2002.

B. Blanchet, “Automatic verification of correspondences for security protocols,” *Journal of Computer Security*, vol. 17, pp. 363–434, July 2009.

“The EVITA european project.”
http://www.evita-project.org/.

H. Schweppe, M. S. Idrees, Y. Roudier, B. Weyl, R. E. Khayari, O. Henniger, D. Scheuermann, G. Pedroza, L. Apvrille, H. Seudié, H. Platzdasch, and M. Sall, “Secure on-board protocols specification,” Tech. Rep. Deliverable D3.3, EVITA Project, 2010.

A. Fuchs, S. Gürgens, L. Apvrille, and G. Pedroza, “On-Board Architecture and Protocols Verification,” Tech. Rep. Deliverable D3.4.3, EVITA Project, 2010.

LabSoc, “TTool,” in *http://labsoc.comelec.enst.fr/turtle/ttool.html*.