

Extending Spi2Java with Nested Types

Alex Busenius

Bachelor Seminar
Advisor: Cătălin Hrițcu
Supervisor: Prof. Dr. Michael Backes

July 1, 2008

Outline

Introduction

- Motivation
- Code Generation

Spi2Java

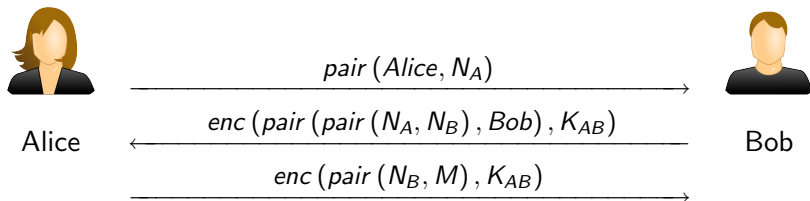
- Features
- Example

Term Types

- Current Type System
- Nested types

Thesis Goals

Mutual Authentication Protocol Model



Spi with Constructors and Destructors (ConSpi)

- ▶ Extension of Spi Calculus
- ▶ Processes and Terms
 - ▶ $out(c, enc(pair(a, b), k)); in(c, x), \dots$
- ▶ Constructors and Destructors
 - ▶ Cryptographic primitives and data structures
 - ▶ Constructors (terms):
 $enc(m, k), pair(a, b), true, \dots$
 - ▶ Destructors (terms list \rightarrow term):
 $dec(enc(m, k), k) \Downarrow m, first(pair(a, b)) \Downarrow a, eq(x, x) \Downarrow true, \dots$
- ▶ Security can be verified by various tools
 - ▶ e.g. ProVerif, type systems, etc.

Reality



Alice

0000416c 69636500 2badf00d

$enc(pair(pair(N_A, N_B), Bob), K_{AB})$

$enc(pair(N_B, M), K_{AB})$



Bob

Reality

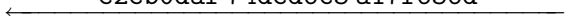


Alice

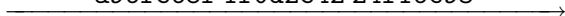
0000416c 69636500 2badf00d



e2eb0daf 74ded6c3 a17f636d



a96fe68f 1f0d2e42 24f16c93



Bob

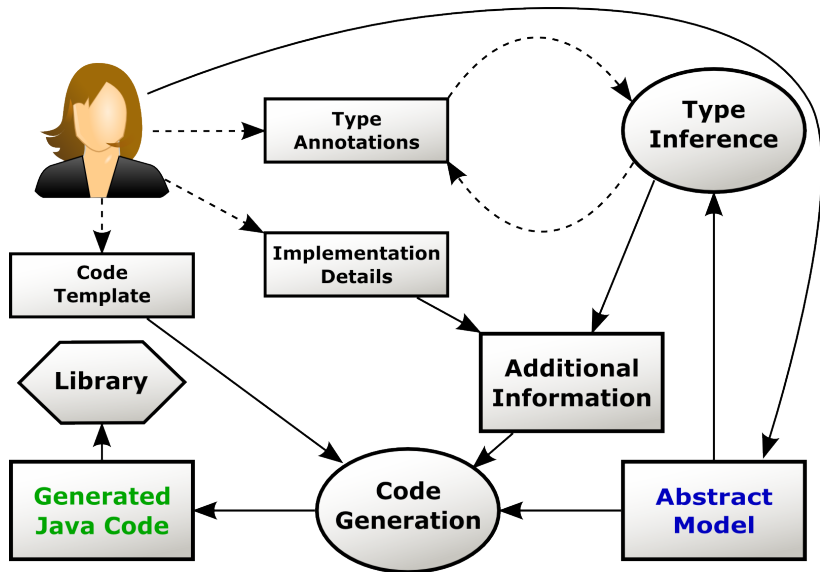
Automatic Code Generation

- ▶ Write abstract model
- ▶ Make sure the model is secure
 - ▶ Tools for (automatic) verification
- ▶ Implement
 - ▶ By hand → Civitas (months)
 - ▶ Generate code automatically (1 click)
- ▶ Why generate?
 - ▶ Model once - use everywhere
 - ▶ Reduce time and costs
 - ▶ Increased confidence in the correctness
 - ▶ Would like to have stronger guarantees (actual proofs)
- ▶ Spi2Java

Spi2Java Features

- ▶ Original version from D. Pozza, A. Pironti and R. Sisto
- ▶ ConSpi → Java implementation
 - ▶ Java API
- ▶ Customizable
 - ▶ Code templates
 - ▶ Library
- ▶ Extensible
 - ▶ Term types
 - ▶ Cryptographic primitives
- ▶ Interoperable
 - ▶ Previous case study: SSH client
- ▶ Free software (GPLv3)

Spi2Java Workflow



What was done so far

- ▶ Extended the original Spi2Java to use ConSpi as input language
 - ▶ Type inference
 - ▶ Code generation
- ▶ Code cleanup and refactoring
- ▶ Tested on simple protocols

Example: Protocol in ConSpi

```
...
free c, Alice, Bob.

let testA = ...

let testB =
  (* A -- pair(Alice, Na) --> B *)
  in(c, tmp);
  let name = first(tmp) in
    let dummy = eq(name, Alice) in
      let Na = second(tmp) in
        new Nb;
        (* A <-- enc(pair(...), k) -- B *)
        out(c, enc(pair(pair(Na, Nb), Bob), k));
        ...

process
  new k; new M;
  (testA | testB)
```

Example: Type Inference

```
...
<term id="92" name="Bob" type="Identifier">
  <codify>IdentifierSR</codify>
</term>
<term id="93" name="pair(pair(Na, Nb), Bob)" type="Pair">
  <codify>PairSR</codify>
</term>
...
<term id="95" name="enc(pair(pair(Na, Nb), Bob), k)"
  type="Shared Key Ciphered">
  <codify>SharedKeyCipheredSR</codify>
  <parameters>
    <param name="algorithm" type="const">DES</param>
    <param name="iv" type="const">12345678</param>
    <param name="mode" type="const">CBC</param>
    <param name="padding" type="const">PKCS5Padding</param>
    <param name="provider" type="const">SunJCE</param>
  </parameters>
</term>
...
```

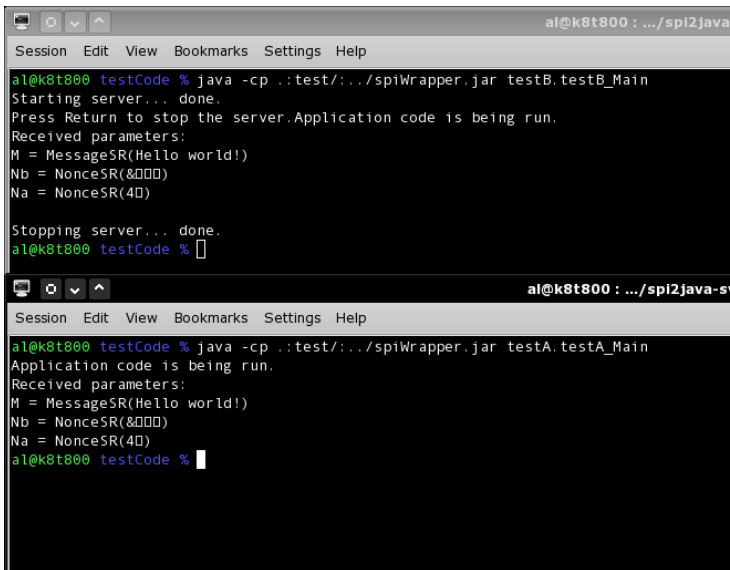
Example: Generated Code

```
/* RestrictedProcess :: new Nb; OutProcess ... */
// "Nb" id=90
Nb = new NonceSR("4");
_return.put("Nb", Nb);

/* OutProcess :: out(c, enc(pair(pair(Na, Nb), Bob), k)); ... */
Pair pair_91_Na_Nb_ = (Pair) PairSR.pair(Na, Nb);
Pair pair_93_pair_91_Na_Nb_Bob = PairSR.pair(pair_91_Na_Nb, Bob);
SharedKeyCiphered enc_95_pair_93_pair_91_Na_Nb_Bob_k_
    = SharedKeyCipheredSR.enc(pair_93_pair_91_Na_Nb_Bob,
                              k, "DES",
                              "12345678", "CBC",
                              "PKCS5Padding", "SunJCE");

// sending message
c.send(enc_95_pair_93_pair_91_Na_Nb_Bob_k);
```

Example: Protocol Run



```
al@k8t800 : .../spi2java
Session Edit View Bookmarks Settings Help
al@k8t800 testCode % java -cp ../spiWrapper.jar testB.testB_Main
Starting server... done.
Press Return to stop the server.Application code is being run.
Received parameters:
M = MessageSR>Hello world!
Nb = NonceSR(&000)
Na = NonceSR(40)

Stopping server... done.
al@k8t800 testCode %

al@k8t800 : .../spi2java-s
Session Edit View Bookmarks Settings Help
al@k8t800 testCode % java -cp ../spiWrapper.jar testA.testA_Main
Application code is being run.
Received parameters:
M = MessageSR>Hello world!
Nb = NonceSR(&000)
Na = NonceSR(40)
al@k8t800 testCode %
```

Long Term Goals

- ▶ Vision: Automatically implement large protocols using zero-knowledge
 - ▶ Generate an implementation of Civitas with 1 click
- ▶ Some necessary steps
 - ▶ Another type system
 - ▶ Adapt infrastructure for specifying the low-level details
 - ▶ Library for implementing symbolic zero-knowledge
 - ▶ Code generation for zero-knowledge terms
- ▶ Correctness proofs

Term Types

- ▶ `in(c, x)`
- ▶ `x = c.receive();`
- ▶ Term representation in generated code
 - ▶ Serialization/Deserialization
- ▶ Currently: simple type system
 - ▶ Channel, Message, Nonce, Key, ...
 - ▶ DH Hashing <: Hashing <: Abstract Message
- ▶ Can be used in correctness proofs
 - ▶ Type system for security of implementations (Thorsten)

Type Annotations and Type Inference

- ▶ No types in ConSpi
- ▶ Optional annotations
 - ▶ Comments
 - ▶ Can still be parsed by ProVerif

```
▶ free c (*: Channel <Nonce> *).  
...  
process  
  new N (*: Nonce *);  
  out(c, N); ...
```

- ▶ Types of some terms can be inferred
 - ▶ Inference rules
 - ▶ By usage
 - ▶ Subterms

Nested Types

- ▶ Represent types of terms and subterms
 - ▶ Integer, Boolean, Nonce, ...
 - ▶ Channel<T>, Pair<T, U>, Key<T>, ...
- ▶ More expressive (trusted channels, specialized encryptions, etc.)
- ▶ Helps type inference (less discarded information)
- ▶ Formalize type system
- ▶ Realization
 - ▶ Java generics

```
▶ class Pair<T, U> extends Message {  
    private T first = null;  
    private U second = null;  
  
    public Pair(T first, U second) {  
        this.first = first;  
        this second = second;  
    }  
    ...  
}
```

Type System for Security

- ▶ Subtyping relation captures security
 - ▶ Untrusted type: Un
 - ▶ T public if $T <: Un$
 - ▶ T tainted if $Un <: T$
- ▶ Types can contain logical formulas
 - ▶ $Pair(x, y) : \langle x: Un, y: Un \rangle \{x = enc(y)\}$
- ▶ Can deal with zero-knowledge
- ▶ Translation is non-trivial
 - ▶ Find representation in Java hierarchy
 - ▶ Need structural subtyping relation
 - ▶ Provide low-level details
 - ▶ Stay customizable and extensible
 - ▶ Leverage existing type checker?

Thesis Goals

- ▶ New type system
 - ▶ Nested types
- ▶ Preserve Spi2Java advantages
 - ▶ Customizability
 - ▶ Extensibility
 - ▶ Interoperability
 - ▶ At least the same amount of type inference
- ▶ Try to reuse existing implementation

Further Extensions

- ▶ Many interesting directions
 - ▶ Handling zero-knowledge
 - ▶ Formalizing target calculus (fragment of Java)
 - ▶ Proving code generation correct
- ▶ Usability improvements
 - ▶ User-friendliness
 - ▶ Nicer error messages
 - ▶ Graphical user interface

Thank You

Questions?

Example: Typed ProVerif Interface (TPVI)

```
(* constructors *)  
fun pk(KeyPair) : PublicKey.  
fun sk(KeyPair) : PrivateKey.  
fun h(Message) : Hashing.  
fun enc(Message, SharedKey) : SharedKeyCiphered.  
fun sign(Message, PrivateKey) : PrivateKeyCiphered.  
fun enca(Message, PublicKey) : PublicKeyCiphered.  
fun pair(Message, Message) : Pair.  
fun True : Boolean.  
fun False : Boolean.  
fun zero : Integer.  
fun succ(Integer) : Integer.  
  
(* destructors *)  
reduc dec(enc(x, y), y) = x.  
reduc deca(enca(x, pk(y)), sk(y)) = x.  
reduc msg(sign(x, y)) = x.  
reduc ver(sign(x, sk(k)), pk(k)) = True.  
reduc first(pair(x, y)) = x.  
reduc second(pair(x, y)) = y.  
reduc eq(x, x) = True.  
reduc pre(succ(x)) = x;  
pre(zero) = zero.
```