On the Development and Formalization of an Extensible Code Generator for Real Life Security Protocols

Michael Backes[1,2] Alex Busenius[1] Cătălin Hrițcu[1,3]

[1]Saarland University [2]MPI-SWS [3]University of Pennsylvania

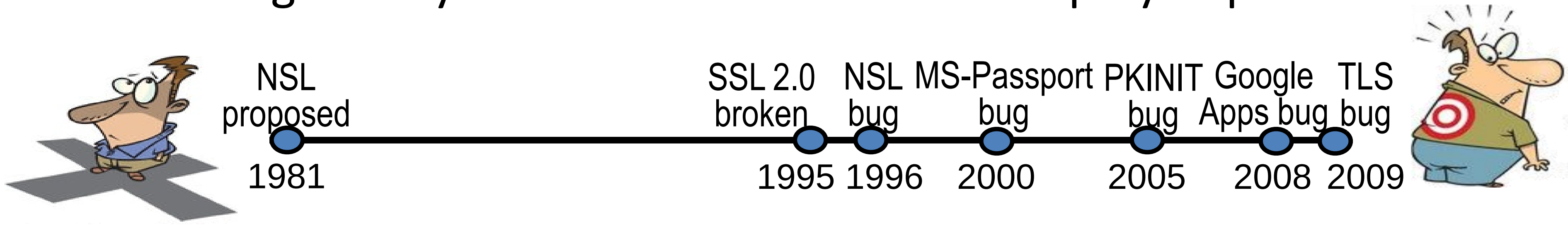
Security protocols

- security protocols are **ubiquitous**

TLS protocol

Security protocols

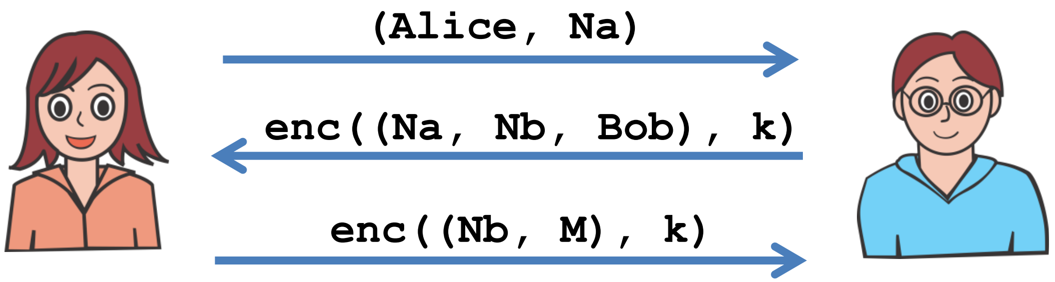
- security protocols are **ubiquitous**
- security protocols are **very easy to get wrong**
  - long history of flaws in standards and deployed products



  - bugs very hard to find manually
  - might be too late to fix; update would “break the Internet”
- a decade of research provides **solutions**:
  - protocol analysis tools (ProVerif, AVISPA, Cryptyc,...)

Why is this not enough?

- most protocol analysis tools work on **abstract protocol models** (formal specifications)



- even if protocol model / spec proved secure
  - models often not executable – can't test interoperability (might be secure but completely dysfunctional)
  - model can abstract away important details
- what does it mean for protocol implementation?
  - verifying model doesn't prevent implementation bugs

OpenSSL Security Advisory [07-Jan-2009]

Incorrect checks for malformed signatures

==

Several functions inside OpenSSL incorrectly checked the result after calling the EVP_VerifyFinal function, **allowing a malformed signature to be treated as a good signature rather than as an error**. This issue affected the signature checks on DSA and ECDSA keys used with SSL/TLS.

One way to exploit this flaw would be for a remote attacker who is in control of a malicious server or who can use a **'man in the middle' attack** to present a malformed SSL/TLS signature from a certificate chain to a vulnerable client, **bypassing validation**.

This vulnerability is tracked as CVE-2008-5077.

The OpenSSL security team would like to thank the Google Security Team for reporting this issue.

OpenSSL Security Advisory [01-Jun-2010]

Invalid Return value check in pkey_rsa_verifyrecover
==

When verification recovery fails for RSA keys an uninitialised buffer with an undefined length is returned instead of an error code (CVE-2010-1633).

This bug is only present in OpenSSL 1.0.0 and only affects applications that call the function EVP_PKEY_verify_recover(). As this function is not present in previous versions of OpenSSL and not used by OpenSSL internal code very few applications should be affected. The OpenSSL utility application "pkeyutl" does use this function.

Affected users should update to 1.0.0a which contains a patch to correct this bug.

Thanks to Peter-Michael Hager for reporting this issue.

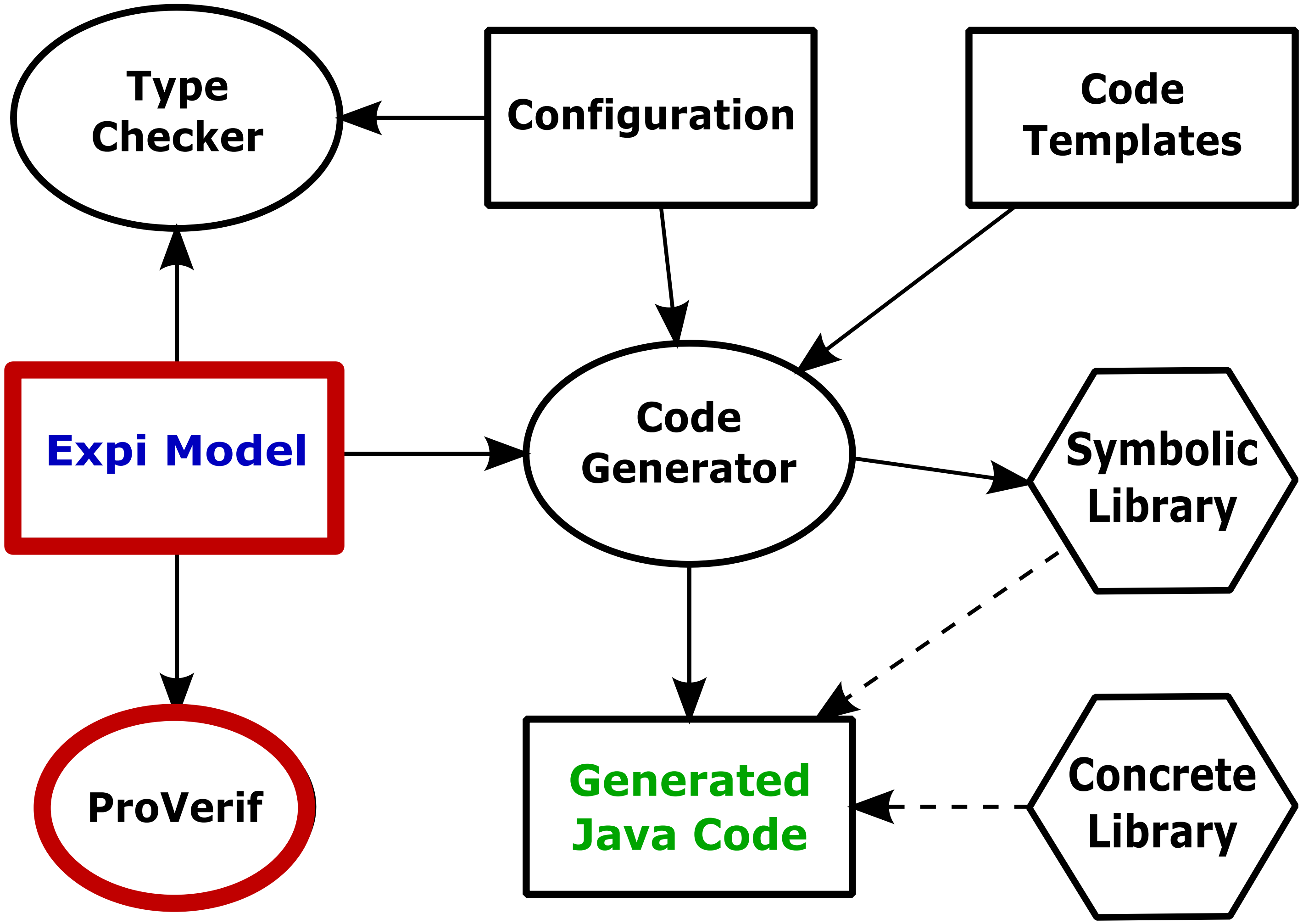
Use model for code generation

1. **write formal model** of protocol (e.g. from RFC)
2. **check security of model** using analysis tool(s)
3. **automatically generate secure implementation**
   - existing code generators:
     - AGVI toolkit (2001), CIL2Java (2001), SPEAR II (2003), Sprite (2004), CPPL compiler (2007)
     - Spi2Java (2004-) first tool to generate **interoperable** implementation (simple SSH client)
   - Spi2Java was starting point for Expi2Java, our **extensible** code generator for security protocols

Expi2Java

- **brings generators for protocols closer to reality**
- works for **complex real-life protocols**
  - case study: interoperable **TLS 1.0 client and server** generated from verified formal model
- highly **extensible** and **customizable**
  - e.g. in TLS we support 3 different encryption algorithms (AES, RC4 and 3DES) with different key lengths
- **useful** and **usable** tool, but also **quite complex**
  - 16.000 lines of Java, most of this in the TCB
  - **formalized** translation algorithm in Coq
  - **proved** that generated Java code is always well-typed

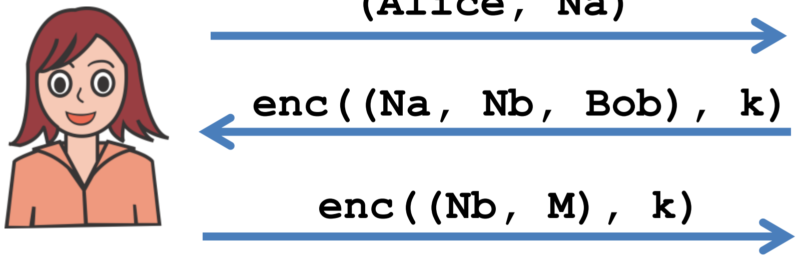
Expi2Java workflow



Expi Model

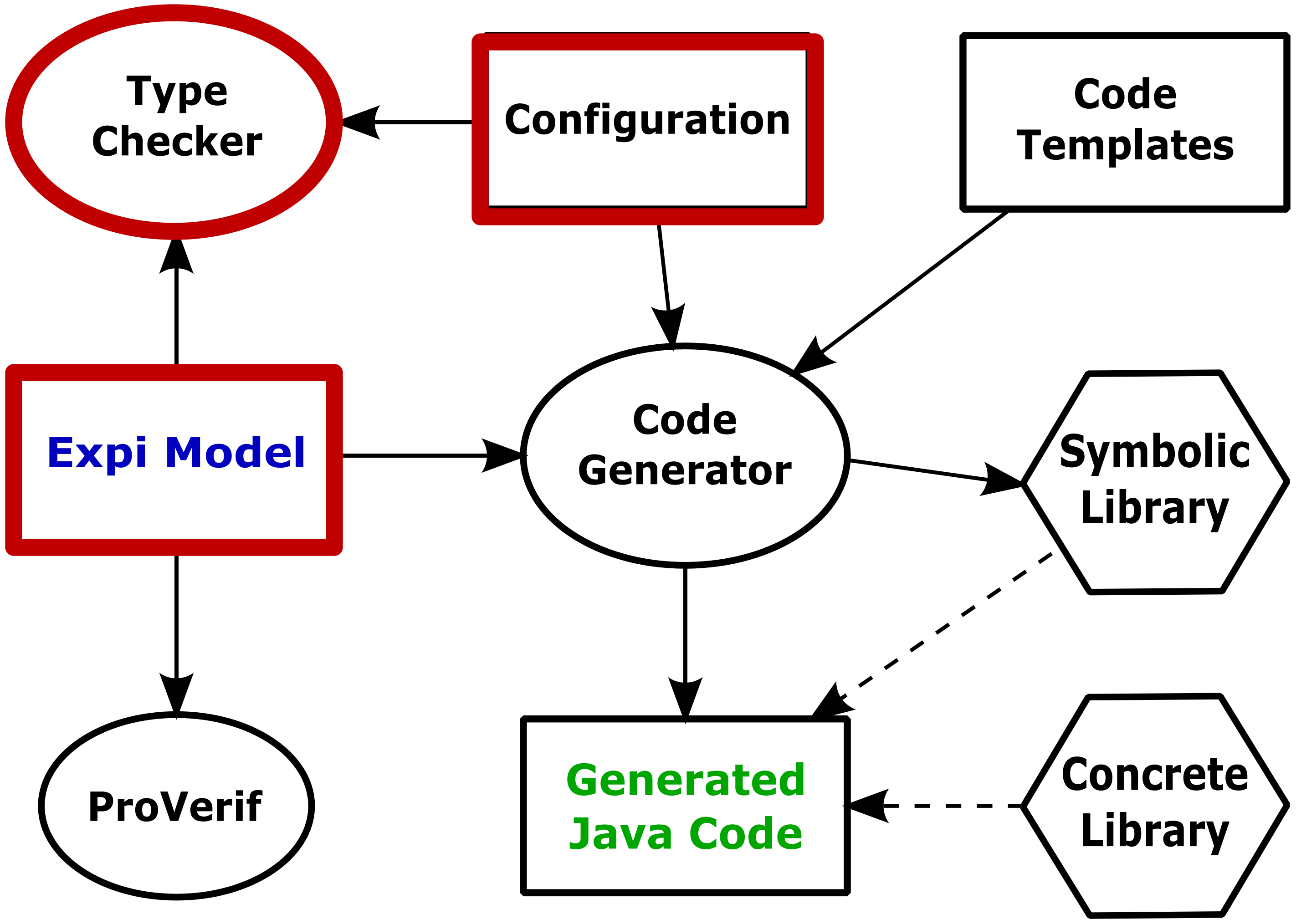
- Using **Extensible Spi calculus** (Expi)
  - very simple DSL for writing security protocols
  - Spi calculus with constructors (pair,enc) and destructors (fst,dec) [Abadi & Gordon, '97] [Abadi and Blanchet, '05]
  - extended with **types + configurations**
  - compatible with ProVerif analysis tool [Blanchet et al.]

```
let AliceP =
    in(cin, M);
    new Na : $Nonce;
    out(c1, (Alice, Na));
    in(c2, emsg);
    let (Na1,Nb,B1) = dec(emsg, k) in
    if Na = Na1 and B1 = Bob then
        out(c3, enc((Nb, M), k));
        out(cout, "message sent").
let BobP = ...
query attacker (M). query attacker (k).
process (AliceP | BobP)
```



Perrig-Song protocol

Types and configurations



The need for Expi types

1. can’t synthesize Java **type annotations** “out of thin air”
   – user has to provide type annotations on model
   – we translate them automatically to Java annotations
2. good to **detect errors early**; before code generation
   – even if user makes mistake, we don’t want error messages in terms of generated Java code user doesn’t understand
   – **Expi type-checker gives errors in terms of model**
   – if model is well-typed, generated code is well-typed Java
3. bonus: **Expi type-checker prevents misconfigurations**

Types and configuration

```
free Alice : $Identifier.
free Bob : $Identifier.
private free k : $KeyAB;
private free cin : $STDIN;
private free cout : $STDOUT;
free c1 : Channel@Ch<$Msg1>;
free c2 : Channel@Ch<$Msg2>;
free c3 : Channel@Ch<$Msg3>;
```

```
let AliceP =
    in(cin, M);
    new Na : $Nonce;
    out(c1, (Alice, Na));
    in(c2, emsg);
    let (Na1,Nb,B1) = dec(emsg, k) in
    if Na = Na1 and B1 = Bob then
        out(c3, enc((Nb, M), k));
        out(cout, "message sent").
let BobP = ...
query attacker (M). query attacker (k).
process (AliceP | BobP)
```

Types and configuration

```
free Alice : $Identifier.
free Bob : $Identifier.
private free k : $KeyAB;
private free cin : $STDIN;
private free cout : $STDOUT;
free c1 : Channel@Ch<$Msg1>;
free c2 : Channel@Ch<$Msg2>;
free c3 : Channel@Ch<$Msg3>;
```

```
(* cryptographic primitive types *)
type SymEnc<T0>.
type generative SymKey<T->.
(* data types *)
type generative Int.
type Pair@PairCfg<X+, Y+>.
(* constructors *)
fun enc: [T].(T,SymKey<T>)->SymEnc<T>.
(* destructors *)
reduc dec(enc[T](x, y), y) = x
      :[T].(SymEnc<T>,SymKey<T>)->T.
```

```
include "default.exdef"
config Ch extends TcpIpChCfg_(
    host        = "localhost",
    timeout_ms  = "20000",
    port        = "2121").

typedef $Msg1   = Pair<$Identifier, $Nonce>.
typedef $Msg2   = SymEnc@AES<Pair<Pair<$Nonce, $Nonce>, $Identifier>>.
typedef $Msg3   = SymEnc@AES<Pair<$Nonce, ExpiString>>.

typedef $KeyAB  = SymKey@AES<Pair<Top, Top>>.
```

Types and configuration

```
free Alice : $Identifier.
free Bob : $Identifier.
private free k : $KeyAB;
private free cin : $STDIN;
private free cout : $STDOUT;
free c1 : Channel@Ch<$Msg1>;
free c2 : Channel@Ch<$Msg2>;
free c3 : Channel@Ch<$Msg3>;
```

```
config IntCfg(
    size             = "4",
    value            = "0",
    type Int(class = "SignedInt")).

config NonceCfg extends IntCfg(
    size             = "8",
    type Int(class = "Nonce")).

typedef $Nonce = Int@NonceCfg.
```

```
include "default.exdef"
config Ch extends TcpIpChCfg_(
    host       = "localhost",
    timeout_ms = "20000",
    port       = "2121").

typedef $Msg1   = Pair<$Identifier, $Nonce>.
typedef $Msg2   = SymEnc@AES<Pair<Pair<$Nonce, $Nonce>, $Identifier>>.
typedef $Msg3   = SymEnc@AES<Pair<$Nonce, ExpiString>>.

typedef $KeyAB  = SymKey@AES<Pair<Top, Top>>.
```

Types and configuration

```
free Alice : $Identifier.
free Bob : $Identifier.
private free k : $KeyAB;
private free cin : $STDIN;
private free cout : $STDOUT;
free c1 : Channel@Ch<$Msg1>;
free c2 : Channel@Ch<$Msg2>;
free c3 : Channel@Ch<$Msg3>;
```

```
config ConsoleChCfg(
type Channel(class="ConsoleChannel")).

(* standard input / output *)
typedef $STDOUT =
     Channel@ConsoleChCfg<Top>.
typedef $STDIN  =
     Channel@ConsoleChCfg<ExpiString>.
```

```
include "default.exdef"
config Ch extends TcpIpChCfg_(
    host         = "localhost",
    timeout_ms   = "20000",
    port         = "2121").

typedef $Msg1    = Pair<$Identifier, $Nonce>.
typedef $Msg2    = SymEnc@AES<Pair<Pair<$Nonce, $Nonce>, $Identifier>>.
typedef $Msg3    = SymEnc@AES<Pair<$Nonce, ExpiString>>.

typedef $KeyAB   = SymKey@AES<Pair<Top, Top>>.
```

Types and configuration

```
free Alice : $Identifier.
free Bob : $Identifier.
private free k : $KeyAB;
private free cin : $STDIN;
private free cout : $STDOUT;
free c1 : Channel@Ch<$Msg1>;
free c2 : Channel@Ch<$Msg2>;
free c3 : Channel@Ch<$Msg3>;
```

```
config AES(
    algorithm  = "AES",
    keylength  = "256",
    provider   = "SunJCE",
    type SymEnc(
       class    = "SymEnc",
       mode     = "CTR",
       padding = "PKCS5Padding"),
    type SymKey(
       class    = "SymKey",
       iv       = "1234567887654321")).
```

```
include "default.exdef"
config Ch extends TcpIpChCfg_(
    host          = "localhost",
    timeout_ms    = "20000",
    port          = "2121").

typedef $Msg1    = Pair<$Identifier, $Nonce>.
typedef $Msg2    = SymEnc@AES<Pair<Pair<$Nonce, $Nonce>, $Identifier>>.
typedef $Msg3    = SymEnc@AES<Pair<$Nonce, ExpiString>>.

typedef $KeyAB   = SymKey@AES<Pair<Top, Top>>.
```

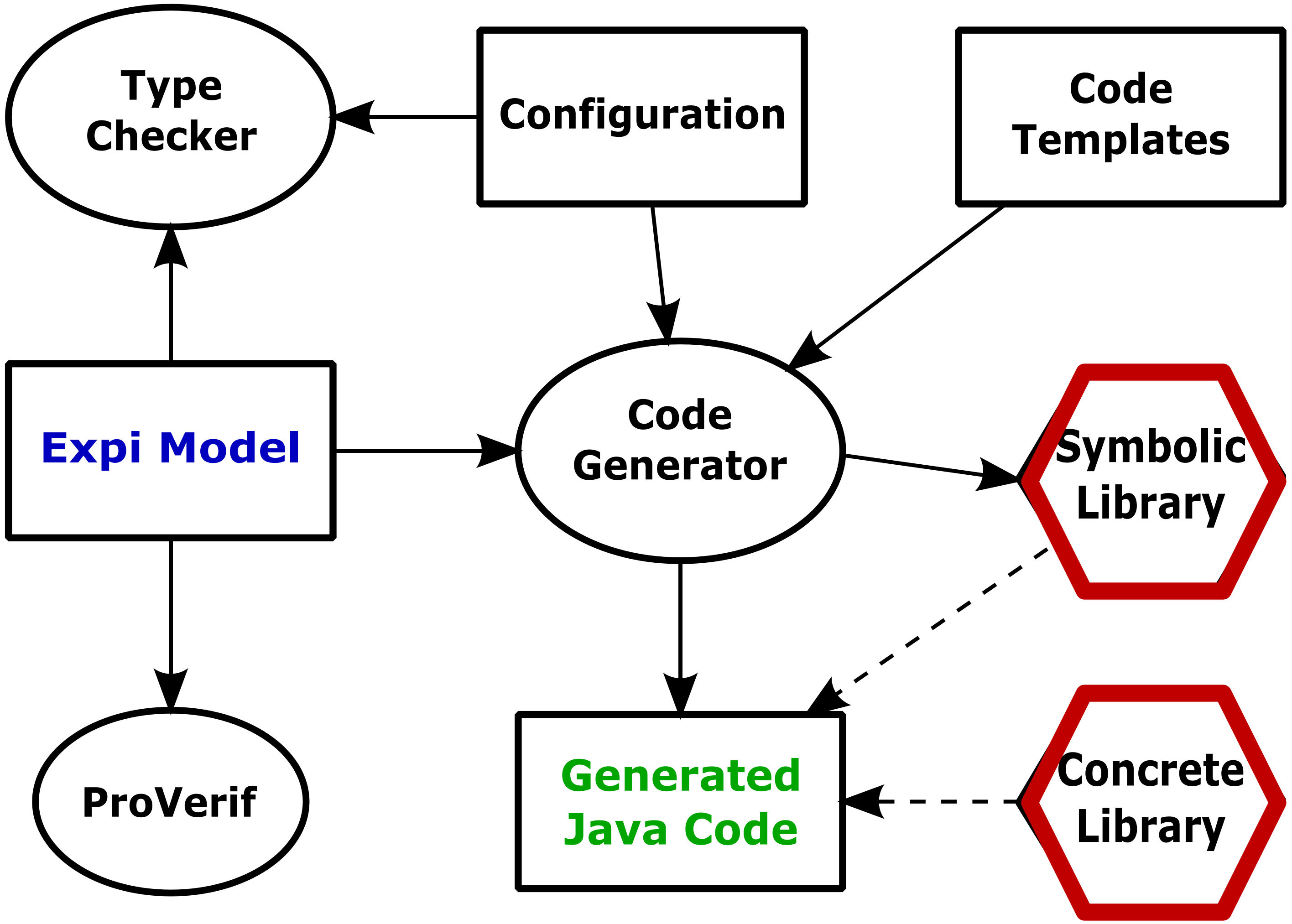
Configuration: nothing “hard coded”

- Everything is **easily extensible & configurable**:
  - crypto primitives (enc, dec)
  - types (Int, Nonce, Pair, SymKey, SymEnc)
  - user has full control over all low-level details
    - choosing crypto algorithm (e.g. AES)
      - precise variant (e.g. CTR encryption mode)
    - size of messages, keys and nonces (e.g. 256bit keys)
    - endianness of integer nonces (big endian / little endian)
    - padding and bit level encoding (e.g. PKCS5Padding)
    - others parameters (e.g. iv = "1234567887654321")

Expi type-checker

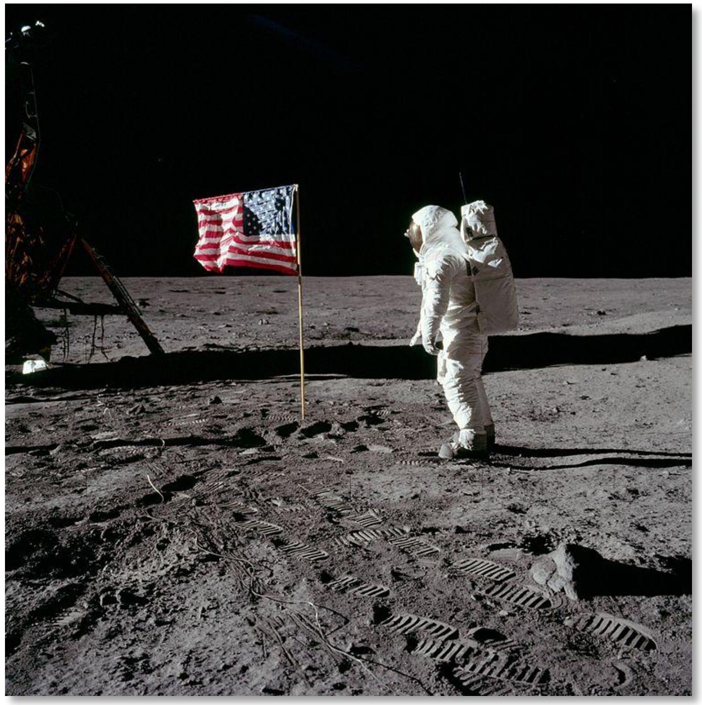
- reflects Java types to the model level
- features **subtyping** and **polymorphism** (generics)
  - nested types capture structure of crypto terms
- precise unification-based **type inference**
  - decreases annotation burden + increases readability
- type-checker **prevents misconfiguration**
  - types, ctors, & dtors parametrized by crypto algorithm
  - e.g. can’t decrypt AES ciphertext with DES key

Concrete/Symbolic Crypto Library



Concrete/Symbolic Crypto Library

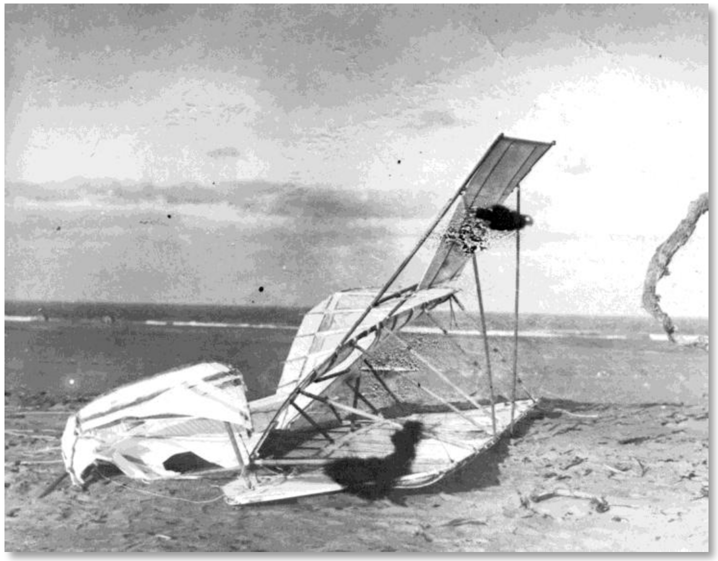
- generated code relies on crypto library
- 2 interchangeable versions:
- **concrete library** (what we deploy)
  - does **real networking and real crypto**
  - uses the standard Java cryptographic providers
- **symbolic library** (what we formalize and prove)
  - symbolic crypto (Java encoding of Expi terms)
  - networking abstracted as inter-thread communication
  - automatically generated from configuration
  - **main goal is to make proofs easier / possible**
  - also useful for testing and debugging locally



TLS 1.0 CASE STUDY

TLS 1.0 case study

- **fully functional** and **interoperable TLS 1.0 client & server**, with AES & SNI extensions (RFCs 3268 & 4366)
- input: 850 lines of model + 625 lines of config
  - **less configuration than model** (ignoring default config)
  - Spi2Java SSH client: 250 lines of model + 1390 lines of config
  - **model verified** with ProVerif (15 queries in 9 mins)
- dynamically negotiates 1 of **6 supported cipher suites**
  - AES, RC4, and 3DES encryption; different key lengths
  - SHA1 or MD5 HMACs
  - RSA key exchange
  - using parametrized processes this is only a few lines
- code generation time: **12 seconds**
- **tested interoperability** with common browsers and servers
- unsupported features: session resumption (vulnerable), client-authenticated handshake, record fragmentation



COQ FORMALIZATION

Coq formalization

Coq +
Ott +
LNGen

- defined Expi calculus + type system
  - + machine-checked proof of **“destructor consistency”**
- defined Java fragment + type system (VPJ)
  - – “Jinja with Threads” [Klein & Nipkow, ’06], [Lochbihler ’08, ’10] extended with variant parametric types [Igarashi & Viroli, ’06]
- defined Expi2Java translation algorithm
  - + machine-checked proof that **generated Java code is well-typed if original model is well-typed**
- first steps towards formal correctness proof
  - – preservation of security properties
- no existing tool had a thorough or mechanized formalization or an interesting proof

Lessons learned

- **formalization harder than we originally expected**
- we target **Java** (most code generators for protocols do):
  - advantages for **usability** and **security**:
    - type and memory safety
    - extensive standard library
    - cryptographic service provider architecture
    - easy to integrate generated code into existing apps
  - **Java is very complex** (7kLOC out of 17kLOC just for defining VPJ)
  - **Java lacks functional features** needed for symbolic terms:
    - immutable data structures, structural equality, and pattern matching
    - Scala or F# would be better suited for implementing symbolic library
    - no formalizations for comprehensive subsets of Scala or F#

Looking for users

- URL: http://www.infsec.cs.uni-saarland.de/projects/expi2java (or just google “Expi2Java”)
- everything linked from there:
  - implementation
  - documentation (user manual, tutorial, and JavaDoc)
  - samples (including TLS 1.0)
  - all source code under GPL v3
  - Coq formalization and proofs (LGPL v3)
  - online appendix of NFM 2012 paper
  - mailing list
- if this sounds interesting then please give it a try!
- if you like it you can make it your own! ☺

BACKUP SLIDES

Variant Parametric Jinja (VPJ)

- **Java fragment based on “Jinja with Threads”** [Klein & Nipkow, ’06], [Lochbihler ’08, ’10]
  - most comprehensive formalized Java fragment:
  - classes, methods, fields, inheritance, casting, **concurrency** and thread **synchronization**, **exceptions**, …
  - ported it from Isabelle/HOL to Coq
- **extended with variant parametric types** [Igarashi & Viroli, ’06]
  - Java 5 generics and wildcards (List<? extends T>)

Looking for new maintainer(s)

- we’re working on Expi2Java since 2008
- Alex has put a lot of effort into engineering, documenting, scaling it up to real-world examples, ...
- we kept working on this even if it wasn’t always very fulfilling (no users, little scientific impact)
- but now we need to move on with our lives

“Lesson #5: When you lose interest in a program, your last duty to it is to hand it off to a competent successor.”
– The Cathedral and the Bazaar

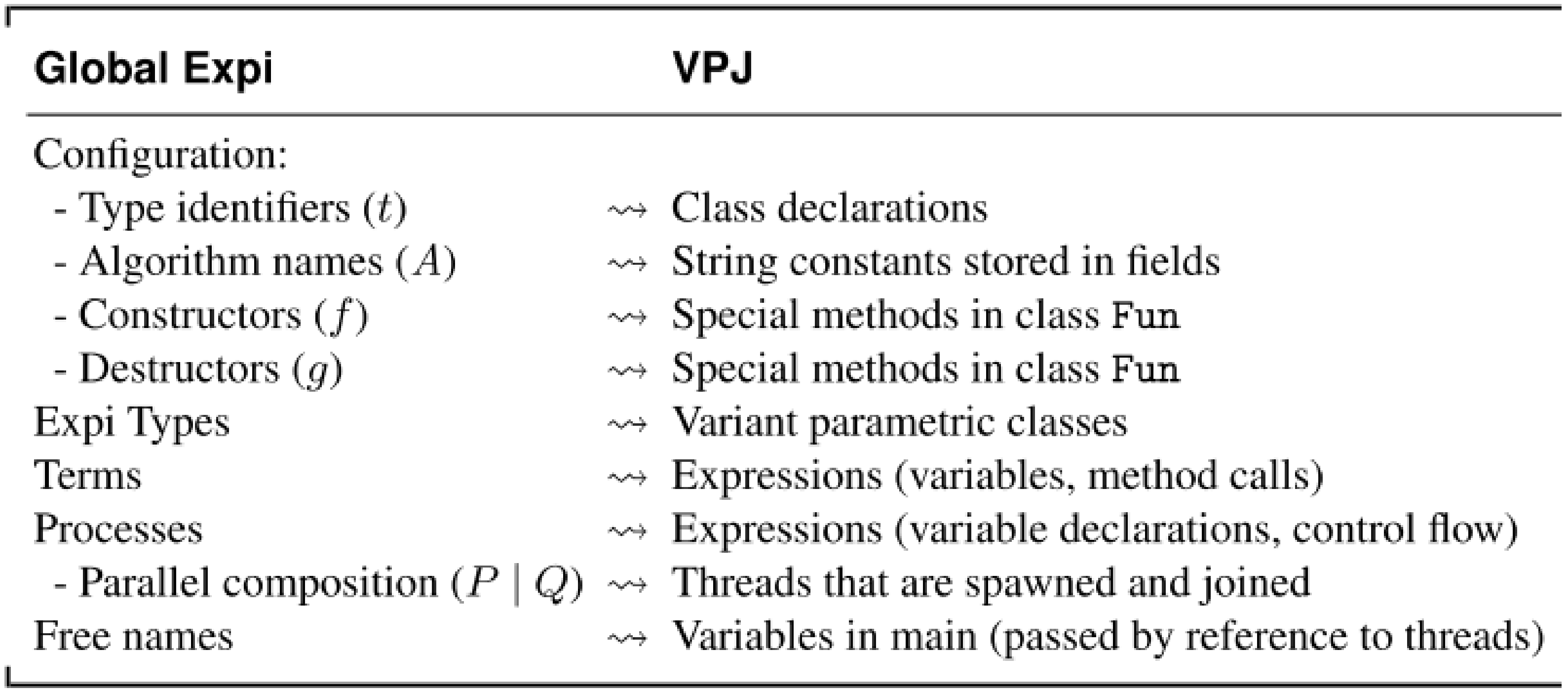
Proving correctness (speculative)

- translation preserves **trace properties** of model
  - use weak labeled simulation to relate VPJ programs to Expi processes
- translation preserves **security properties** of model (e.g. robust safety)
  - simulation is contextual
  - each VPJ attacker can be mapped back to Expi attacker
- current work
  - solid ground on which preservation of security properties can be formally investigated

Code generator

- can customize the code generator's templates
- also translates types
- details are in the paper

Table 7. Translation Overview



Global Expi		**VPJ**
Configuration:		
- Type identifiers (t)	$\rightsquigarrow$	Class declarations
- Algorithm names (A)	$\rightsquigarrow$	String constants stored in fields
- Constructors (f)	$\rightsquigarrow$	Special methods in class `Fun`
- Destructors (g)	$\rightsquigarrow$	Special methods in class `Fun`
Expi Types	$\rightsquigarrow$	Variant parametric classes
Terms	$\rightsquigarrow$	Expressions (variables, method calls)
Processes	$\rightsquigarrow$	Expressions (variable declarations, control flow)
- Parallel composition ($P \mid Q$)	$\rightsquigarrow$	Threads that are spawned and joined
Free names	$\rightsquigarrow$	Variables in main (passed by reference to threads)

The importance of good errors

```
  [javac] ./expi2java/trunk/testCode/expi2java/perrigsong/Thread_pB_44.java:239:
<U,V,W>decDtor(V,library.data.SymEnc<W>,library.data.SymKey<U>)
        in
library.data.SymEnc<library.data.Pair<library.data.Int,library.data.ExpiString>>
        cannot be applied to
(library.data.Pair<library.data.Int,library.data.ExpiString>,library.data.SymKey<library
.AbstractBase>,library.data.SymEnc<library.data.Pair<library.data.Int,library.data.Exp
iString>>)
  [javac]    Pair<Int, ExpiString> tmp_47 = msg3_218.decDtor(Pair.instance("Pair",
Int.instance("Nonce", "Nonce.PSNonceCfg") ...
  [javac]                                 ^
  [javac] 1 error
```

```
Type check failed:
In process "let (Nx, message) = dec(Kab, msg3) in LetProcess_46 ... ":
In term "dec(Kab, msg3)" at line 55, column 25. file "testData/expi/perrigsong.expi"
Can not gather constraints for the type " : SymEnc<T>"
since it is incompatible to the concrete type " : SymKey@AES<Top>"
```

Formalization size

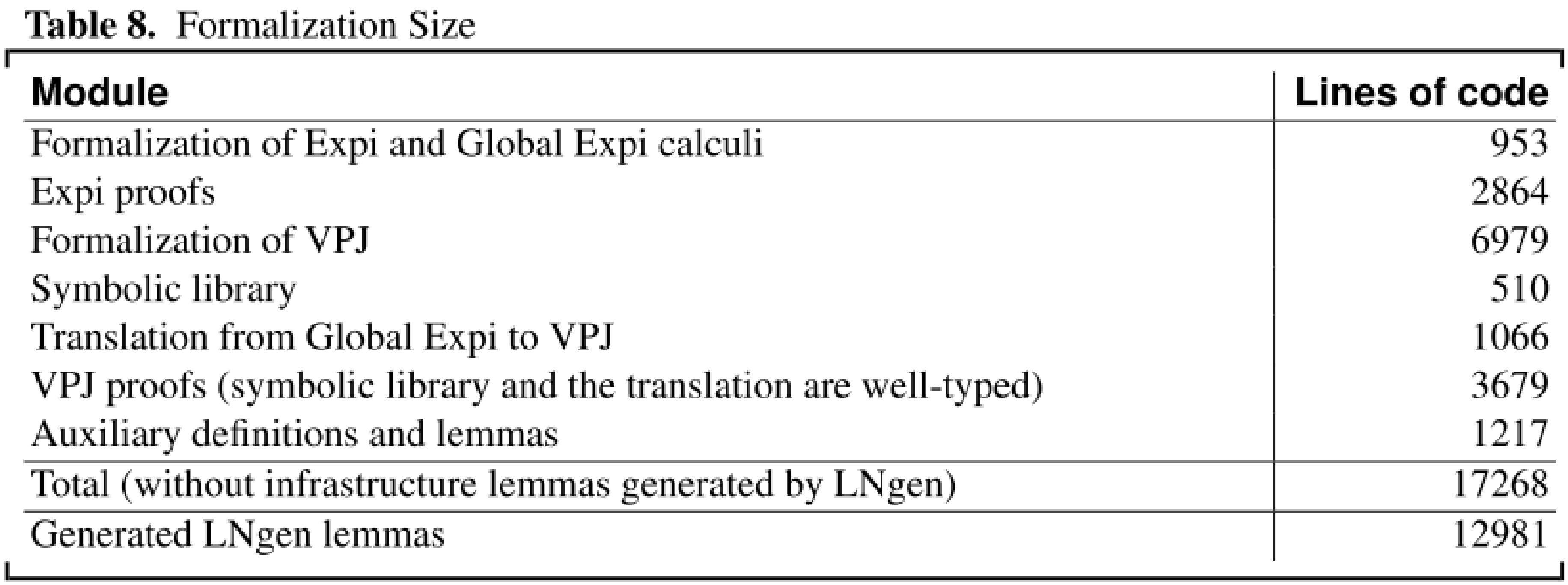


Table 8. Formalization Size

Module	Lines of code
Formalization of Expi and Global Expi calculi	953
Expi proofs	2864
Formalization of VPJ	6979
Symbolic library	510
Translation from Global Expi to VPJ	1066
VPJ proofs (symbolic library and the translation are well-typed)	3679
Auxiliary definitions and lemmas	1217
Total (without infrastructure lemmas generated by LNgen)	17268
Generated LNgen lemmas	12981

Spi2Java

- first code generator for protocols that attempts to be flexible and configurable
- SSH client case study proves interoperability
- quite limited:
  - customization mechanism is very hard to use (manually editing huge + redundant XML files)
  - cannot handle all cryptographic primitives in TLS
  - no extension mechanism to circumvent this
- no formalization and no credible proofs
- open source, so we used Spi2Java code to get started with Expi2Java; ended up rewriting it
- [Pozza et al, AINA '04] [Pironti & Sisto, ISCC '07]

OpenSSL Security Advisory [2 December 2010]

OpenSSL JPAKE validation error
=================================

Sebastian Martini found an error in OpenSSL's J-PAKE implementation which could lead to **successful validation by someone with no knowledge of the shared secret**. This error is fixed in 1.0.0c. Details of the problem can be found here:

http://seb.dbzteam.org/crypto/jpake-session-key-retrieval.pdf

Note that the OpenSSL Team still consider o

to be experimental and is not compile

This issue is tracked as CVE-2010-42

"they aren't adequately verifying the public parameters received from untrusted parties"