Extracting and Verifying Cryptographic Models from C Protocol Code by Symbolic Execution

Mihhail Aizatulin[1]
supervised by
Andrew Gordon[2,3], Jan Jürjens[4], Bashar Nuseibeh[1]

[1]The Open University

[2]Microsoft Research Cambridge

[3]University of Edinburgh

[4]Dortmund University

December 2011

The Goal

- Problem: we often verify formal models of cryptographic protocols, but what we rely on are their implementations.
- Bridge the gap by extracting high-level (pi calculus) models straight from C code.
- We check trace properties such as authentication and weak secrecy, aiming to be automated and sound.
- Assume correctness of cryptographic primitives.
- Main limitation so far: model extracted from a single program path.

Background

Types of properties and languages.

	Low-Level (C, Java)	High-Level (F#)	Formal (π, LySa)
low-level (NULL dereference, division by zero)	• VCC • Frama-C • ESC/Java • SLAM	N/A	N/A
high-level (secrecy, authentication)	• CSur • JavaSec • ASPIER • csec-modex	• F7/F* • fs2pv/fs2cv	• ProVerif • CryptoVerif • AVISPA • LySatool

Results

	C LOC	model LOC	outcome	result type	time
simple mac	~ 250	12	verified	symbolic	4s
RPC	~ 600	35	verified	symbolic	5s
NSL	~ 450	40	verified	computat.	5s
CSur	~ 600	20	flaws found	—	5s
Metering	~ 1000	51	flaws found	—	15s

- Three implementations (1300 LOC) verified in the symbolic model.
- One of them also verified in the computational model by application of a computational soundness result.
- Found 3 flaws in a Microsoft Research implementation of a smart metering protocol (1000 LOC) (all fixed now).

Example Flaw

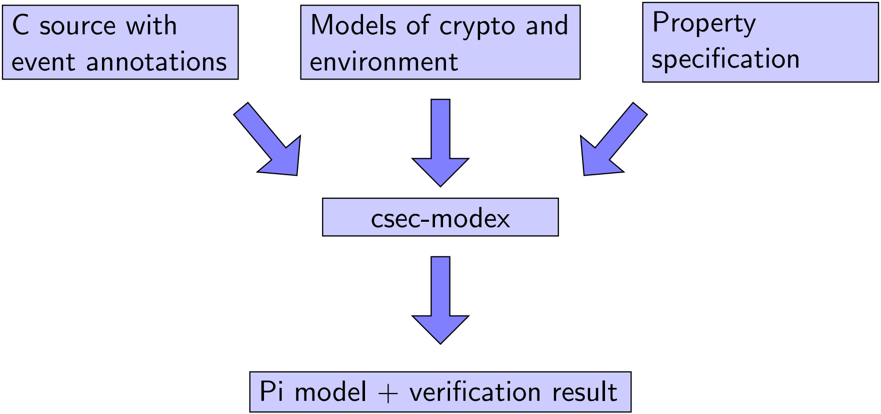
In the smart metering protocol:

```
unsigned char session_key[256 / 8];
...
encrypted_reading = ((unsigned int) *session_key) ^ *reading;
```

Extracted model:

let msg3 = (hash2{0, 1} castTo "unsigned_int") $\oplus$ reading1 **in** ...

Overview: What



Demo

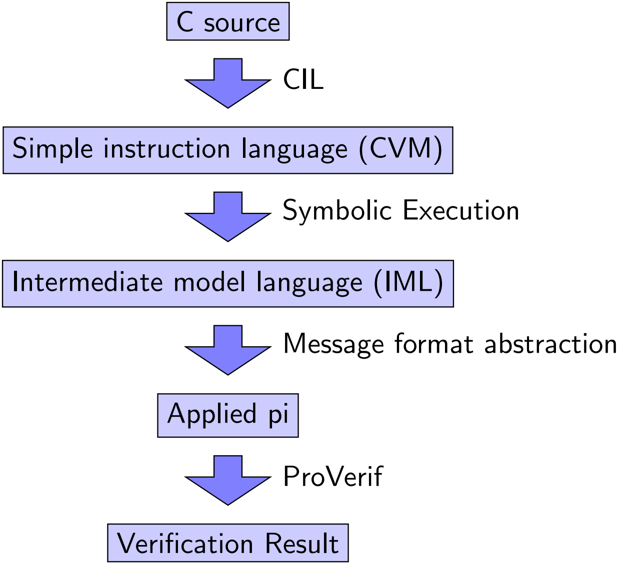
Abstract protocol:

$$A \xrightarrow{m, hmac(m, k_{AB})} B.$$

Concrete protocol:

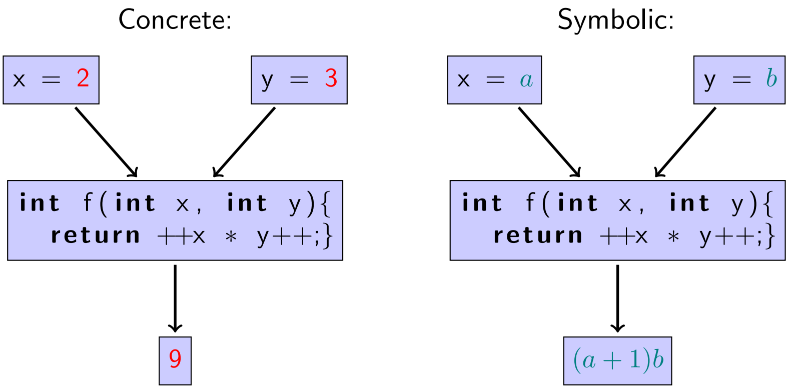
$$A \xrightarrow{\mathrm{len}(m)|1|m|hmac(\mathrm{len}(m)|2|m, k_{AB})} B.$$

Overview: How



Symbolic Execution: Basic Idea

Symbolic execution is a tool to simplify programs and extract their meaning.



Output:

$\text{stack}\, msg \rightsquigarrow \text{ptr}(\text{heap}\, 2, 0)$

$\text{heap}\, 2 \rightsquigarrow \text{len}(x)|x|y \oplus k$

$\text{stack}\, msg\_len \rightsquigarrow 4 + \text{len}(x) + \text{len}(y)$

```
write(msg, msg_len);
```

Generate IML "**out**$(\text{len}(x)|x|y \oplus k)$;".

An IML model:

let A =
  in(x);
  event($send(x)$);
  out($\mathrm{len}(x)|1|x|hmac(x, k_{AB})$).

let B =
  in(m);
  if $\mathrm{len}(m) < m\{0,4\} + 5$ **then**
  if $m\{4,1\} = 1$ **then**
  let $x = m\{5, m\{0,4\}\}$ **in**
  let $h = m\{5 + m\{0,4\}, \mathrm{len}(m) - 5 + m\{0,4\}\}$ **in**
  if $h = hmac(x, k_{AB})$ **then**
  event($accept(x)$).

P = !(νk_{AB}; (!A | !B)).

Message Format Abstraction (2)

Pi calculus translation of the IML model:

reduc $d_1(c_1(x, y)) = x$; $d_2(c_1(x, y)) = y$.

query **ev**:$accept(x) \Longrightarrow$ **ev**:$send(x)$.

let A =
  in(x);
  event$(send(x))$;
  out$(c_1(x, hmac(x, k_{AB})))$.
let B =
  in(m);
  let $x = d_1(m)$ **in**
  let $h = d_2(m)$ **in**
  if $h = hmac(x, k_{AB})$ **then**
  event$(accept(x))$.

process !(νk_{AB}; (!A | !B)).

Message Format Abstraction (3)

We prove that IML bitstring manipulation expressions implement pairing.

$$c_1/2 := \lambda xy.\, \mathrm{len}(x)|1|x|y,$$

$$d_1/1 := \lambda x.\textbf{if } \mathrm{len}(m) < x\{0,4\} + 5 \textbf{ then}$$

$$\textbf{if } x\{4,1\} = 1 \textbf{ then } x\{5, x\{0,4\}\} \textbf{ else } \bot,$$

$$d_2/1 := \lambda x.\textbf{if } \ldots \textbf{ then } x\{5 + x\{0,4\}, \mathrm{len}(x) - 5 + x\{0,4\}\} \textbf{ else } \bot.$$

Properties:

- all concatenation functions have disjoint ranges,
- for all x and y: $d_1(c_1(x,y)) = x$ and $d_2(c_1(x,y)) = y$,
- whenever $d_1(m) \neq \bot$ or $d_2(m) \neq \bot$, there exist x, y such that $m = c_1(x,y)$.

Current Status

Implementation available from
`https://github.com/tari3x/csec-modex`

Csec-challenge:
`http://research.microsoft.com/csec-challenge`

Working on:

- Using CryptoVerif for verification of models, removing need for computational soundness results.
- Adding support for arbitrary control flow.

Thank you!