e-Learning System for Cryptography on Moodle

†Osaka Electro-Communication University
‡Shinshu University
§Tokyo University of Technology

Tatsuki Miyamoto†, Shogo Shimura‡, Tatsuki Watanabe†,
Hiroyuki Okazaki‡, Yuichi Futa§, Yasuyuki Murakami†

Summary

In Japan, due to the lack of ICT engineers, fostering human resources for ICT is an urgent task. Especially, cryptographic technology is a fundamental element for realizing information security. Learning knowledge and training techniques on cryptography are indispensable not only to researchers and engineers specialized in cryptography but also to ICT engineers such as network engineers and operators. However, the educational environment and teaching materials of cryptographic technology for ICT engineers are not sufficient. There is also a lack of teachers who can educate the theory and the technology of the cryptography.

Recently, e-Learning using the Internet has been widely spread especially at the educational places such as universities. Moodle is popular and widely used as an e-Learning system. In Moodle, students answer questions on quizzes on the browser via Internet and whether the answer is correct or not can be judged depending on whether or not their answer matches the model answer set by the teacher. Moodle is excellent e-Learning system which also has an automatic scoring function. However, learning programming with Moodle is not always easy because the program is not necessarily one correct answer.

VPL, Virtual Programming Lab for Moodle, is a free system formed by two components: the Moodle plugin and the execution system(Jail-System). The Moodle plugin can be installed as a regular Moodle plugin. The execution system needs to be compiled for installation. The installation is not always easy for teachers.

Formal verification of cryptographic protocols has been studied extensively in recent years. ProVerif is one of the most successful automatic cryptographic protocol verifiers. In the previous work, we developed an e-Learning system for learning C programming based on Moodle with VPL by using VM, Virtual Machine, for easy installation[1]. In this research, we newly support ProVerif to the previously developed system and create the contents for learning ProVerif.

What is Moodle / VPL

Moodle is a free, online Learning Management system. Moodle is widely used as an e-Learning system in many educational institutions. It is easy to install and can be expanded with plugin.

VPL is a free system formed by two components: the Moodle plugin and the execution system. The Moodle plugin can be installed as a regular Moodle plugin. The execution system needs to be compiled for installation.

Students:
- Edit
- Run
- Evaluate

Teachers:
- Easy Evaluation

Developed e-Learning System

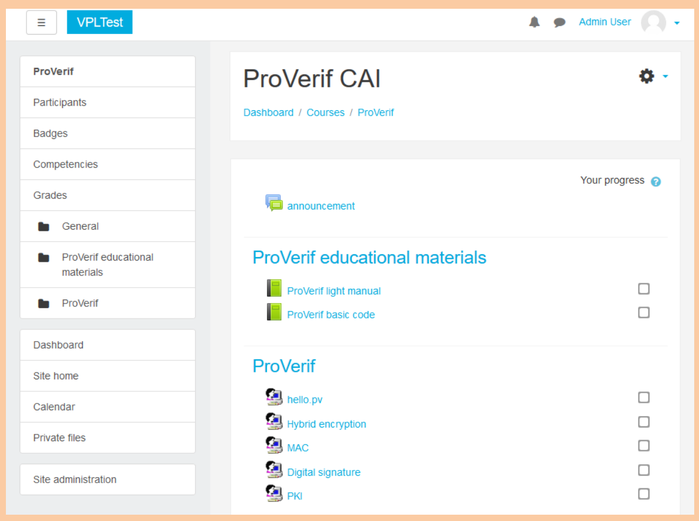

Fig.1 Moodle Screen on Browser

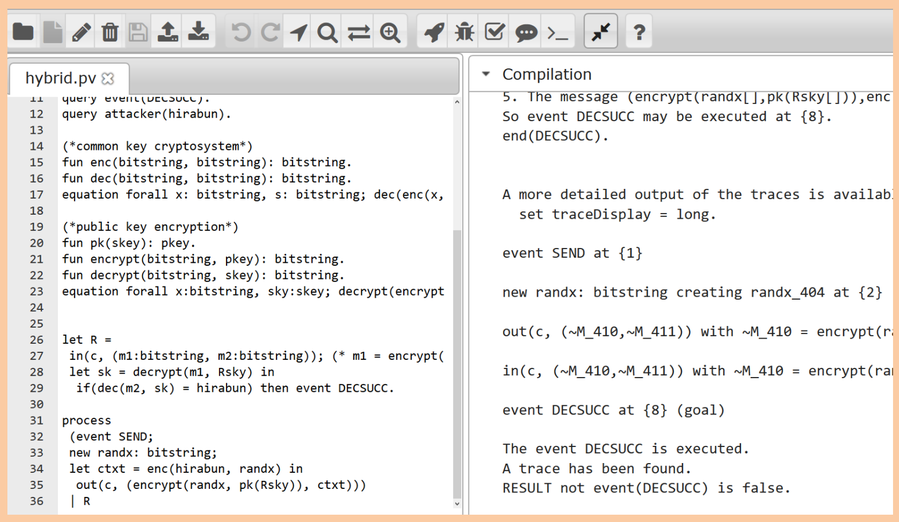

Fig.2 Learning ProVerif on Moodle+VPL

Benefits for Students:

- No compiler required, Browser only required.
  →Available on Tablet or Smartphone.
- No source code submission required.
- Discover mistakes themselves instantly.

Benefits for System Managers:

- Using VM, Any host OS available.
- Easy to Install, manage and update.
- No need to introduce compiler for each client.

Benefits for Teachers:

- Automatic evaluation.
  →No program download, nor compile, nor execution required.
- By the test pattern, it is easy to find mistakes.
- The correct answer is judged as correct even if it is not the model answer.

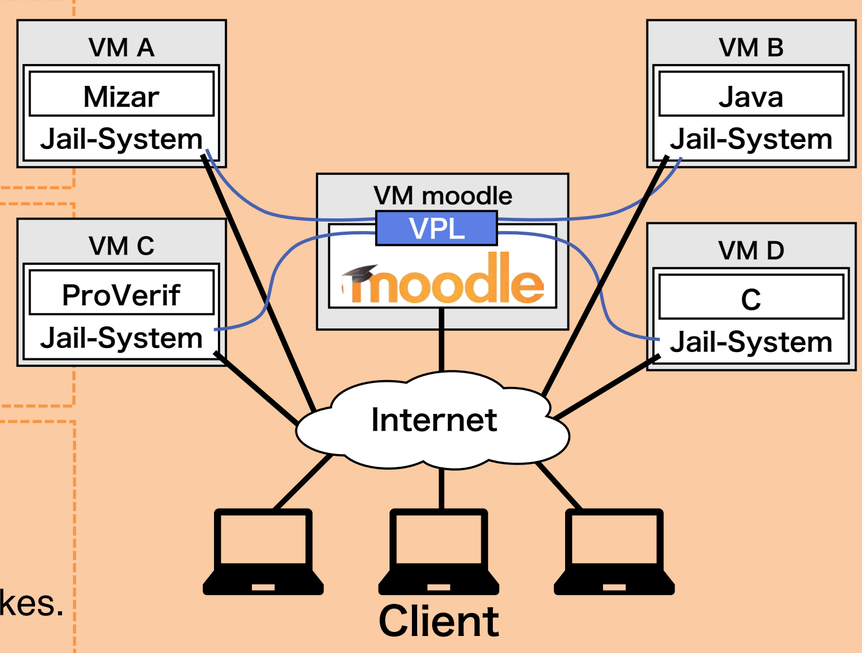

Fig.3 System Configuration

ProVerif Sample

```
free c: channel.
free hirabun: bitstring [private].

type pkey.
type skey.
free Rsky: skey [private].

event DECSUCC.
event SEND.

query event(DECSUCC).
query attacker(hirabun).

(*common key cryptosystem*)
fun enc(bitstring, bitstring): bitstring.
fun dec(bitstring, bitstring): bitstring.
equation forall x: bitstring, s: bitstring; dec(enc(x,s),s) = x.

(*public key encryption*)
fun pk(skey): pkey.
fun encrypt(bitstring, pkey): bitstring.
fun decrypt(bitstring, skey): bitstring.
equation forall x:bitstring, sky:skey; decrypt(encrypt(x, pk(sky)), sky) = x.


let R =
 in(c, (m1:bitstring, m2:bitstring)); (* m1 = encrypt() m2 = ctxt *)
 let sk = decrypt(m1, Rsky) in
  if(dec(m2, sk) = hirabun) then event DECSUCC.

process
 (event SEND;
 new randx: bitstring;
 let ctxt = enc(hirabun, randx) in
  out(c, (encrypt(randx, pk(Rsky)), ctxt)))
 | R
```

Fig.4 ProVerif Sample Source Code

```
Linear part:
Completing equations...
Completed equations:
Convergent part:
dec(enc(x,s),s) = x
decrypt(encrypt(x_7,pk(sky)),sky) = x_7
Completing equations...
Completed equations:
decrypt(encrypt(x_7,pk(sky)),sky) = x_7
dec(enc(x,s),s) = x
Process:
(
{1}event SEND;
{2}new randx: bitstring;
{3}let ctxt: bitstring = enc(hirabun,randx) in
{4}out(c, (encrypt(randx,pk(Rsky)),ctxt))
) | (
{5}in(c, (m1: bitstring,m2: bitstring));
{6}let sk: bitstring = decrypt(m1,Rsky) in
{7}if (dec(m2,sk) = hirabun) then
{8}event DECSUCC
)

- Query not attacker(hirabun[])
Completing...
Starting query not attacker(hirabun[])
RESULT not attacker(hirabun[]) is true.
- Query not event(DECSUCC)
Completing...
Starting query not event(DECSUCC)
goal reachable: end(DECSUCC)

1. The message (encrypt(randx[],pk(Rsky[])),enc(hirabun[],randx[])) may be sent to the attacker at output {4}.
attacker((encrypt(randx[],pk(Rsky[])),enc(hirabun[],randx[]))).

2. By 1, the attacker may know (encrypt(randx[],pk(Rsky[])),enc(hirabun[],randx[])).
Using the function 2-proj-2-tuple the attacker may obtain enc(hirabun[],randx[]).
attacker(enc(hirabun[],randx[])).

3. By 1, the attacker may know (encrypt(randx[],pk(Rsky[])),enc(hirabun[],randx[])).
Using the function 1-proj-2-tuple the attacker may obtain encrypt(randx[],pk(Rsky[])).
attacker(encrypt(randx[],pk(Rsky[]))).

4. By 3, the attacker may know encrypt(randx[],pk(Rsky[])).
By 2, the attacker may know enc(hirabun[],randx[]).
Using the function 2-tuple the attacker may obtain (encrypt(randx[],pk(Rsky[])),enc(hirabun[],randx[])).
attacker((encrypt(randx[],pk(Rsky[])),enc(hirabun[],randx[]))).

5. The message (encrypt(randx[],pk(Rsky[])),enc(hirabun[],randx[])) that the attacker may have by 4 may be received at input {5}.
So event DECSUCC may be executed at {8}.
end(DECSUCC).


A more detailed output of the traces is available with
set traceDisplay = long.

event SEND at {1}

new randx: bitstring creating randx_404 at {2}

out(c, (~M_410,~M_411)) with ~M_410 = encrypt(randx_404,pk(Rsky)), ~M_411 = enc(hirabun,randx_404) at {4}

in(c, (~M_410,~M_411)) with ~M_410 = encrypt(randx_404,pk(Rsky)), ~M_411 = enc(hirabun,randx_404) at {5}

event DECSUCC at {8} (goal)

The event DECSUCC is executed.
A trace has been found.
RESULT not event(DECSUCC) is false.
```

Fig.5 Verifying Process by ProVerif

Conclusion

In this research, we have constructed a programming education support system using Moodle and VPL using VM. We have newly supported ProVerif to previously developed e-Learning system using VM. In addition, we have newly created the contents for learning ProVerif and actually started the learning exercise of ProVerif by using the developed system. As a future plan, we would like to enrich the contents and evaluate the effect of learning the security programming by using the developed e-Learning system.

Acknowledgement: This study was supported in part by JSPS KAKENHI Grant Numbers JP18K02917 and JP17K00182.

[1] M. Nakamura, T, Watanabe, M. Kaneda, H. Okazaki and Y. Murakami, “Programing education support system using Moodle,” 40th Symposium on Information Theory and Its Application, Poster session, SITA2017, Nov. 2017 (in Japanese).