Cryptol을 이용한 국내 표준 블록 암호 ARIA 모듈의 자동 정형 검증

(Automated Formal Verification of Korean Standard Block Cipher ARIA using Cryptol)

Security Analysis aNd Evaluation Lab, CIST

최원빈 (Wonbin Choi)

Prof. Dr. Seungjoo (Gabriel) Kim

Contents

KOREA UNIVERSITY

1. Cryptol 개요

KOREA UNIVERSITY

- **What?**
  - **Galois, Inc.**
  - **High Assurance, Retargetable Crypto Development and Validation**

1. Cryptol 개요

KOREA UNIVERSITY

- **What?**
  - 암호에 대한 **formal-method** 기반 접근
  - **Galois**에서 **NSA**의 암호 전문가와 설계한 **high-level specification language**
  - 하드웨어 플랫폼과 독립적(**independent**)으로 **directly** 및 **formally**하게 표현함
  - **Haskell** 기반의 **Domain-specific language**

1. CryptoI 개요

KOREA UNIVERSITY

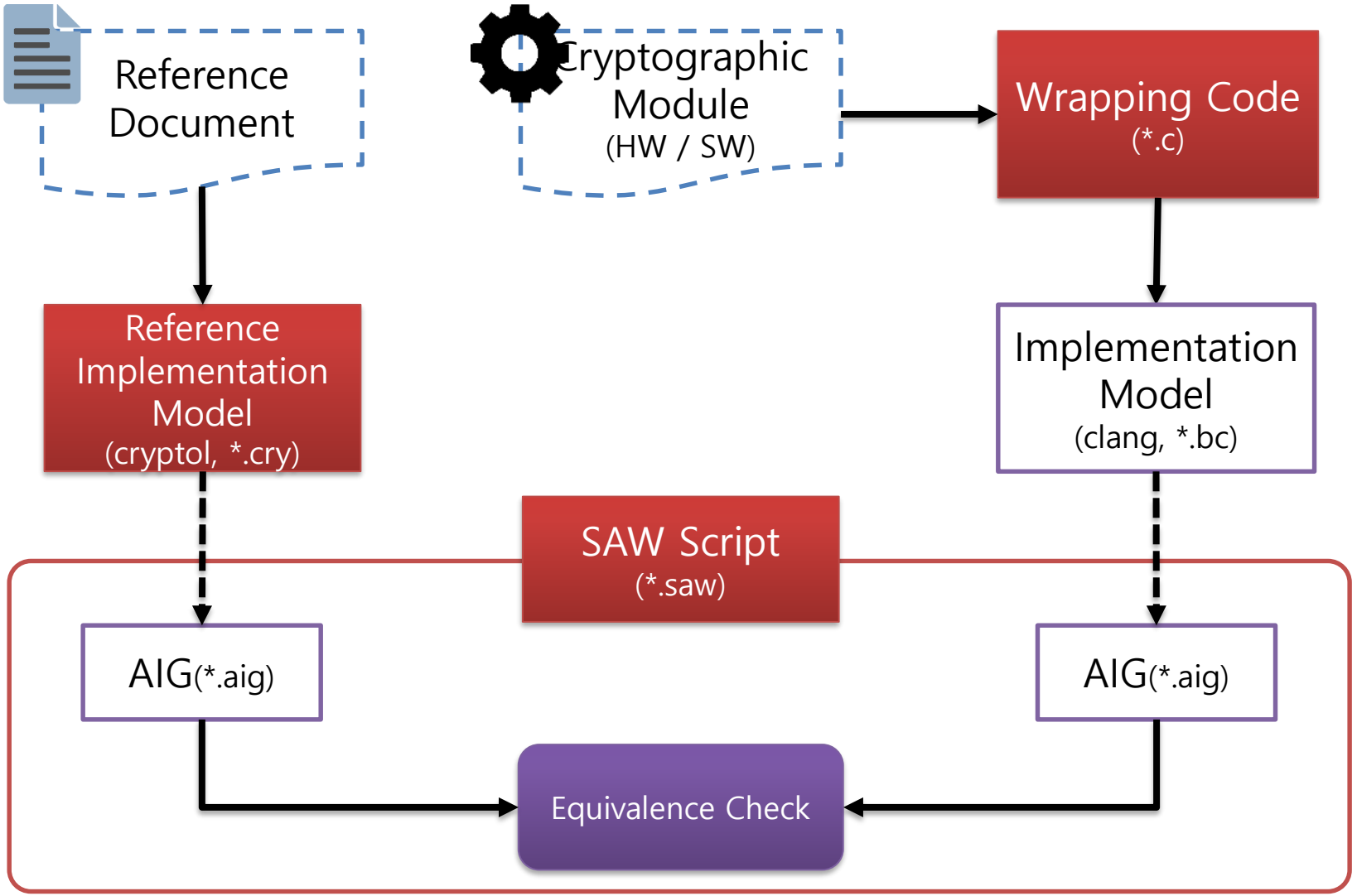
- **Why?**
  - 기존 암호 모듈 검증은 테스트 벡터 기반
    - **KAT(Known Answer Test), MCT(Monte-Carlo Test), MMT(Multi-block Message Test)**와 같은 방법이 존재
    - **ARIA** 알고리즘은 한국인터넷진흥원**(KISA)**에서 테스트 벡터가 제공되고 있음
  - 2^{128}만큼 전수테스트를 수행하는 것은 현실적으로 불가능
    - **AES** 알고리즘의 **MCT**는 **NIST**에서 암호화 **10**만회를 기준으로 함

      $100\ multi\_block\_messages \times 1{,}000\ block$

2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

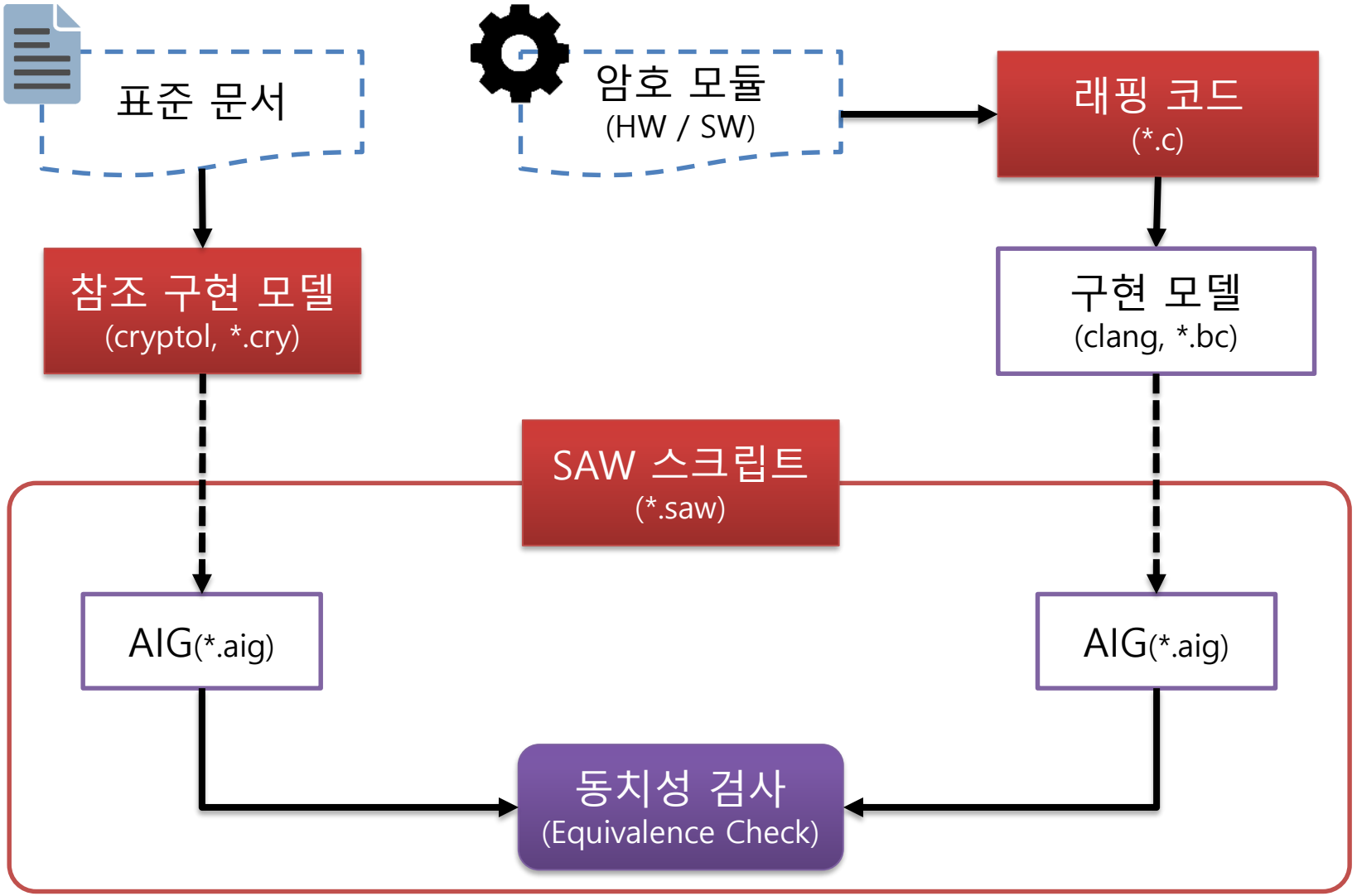
- **Implementation and verification architecture**



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

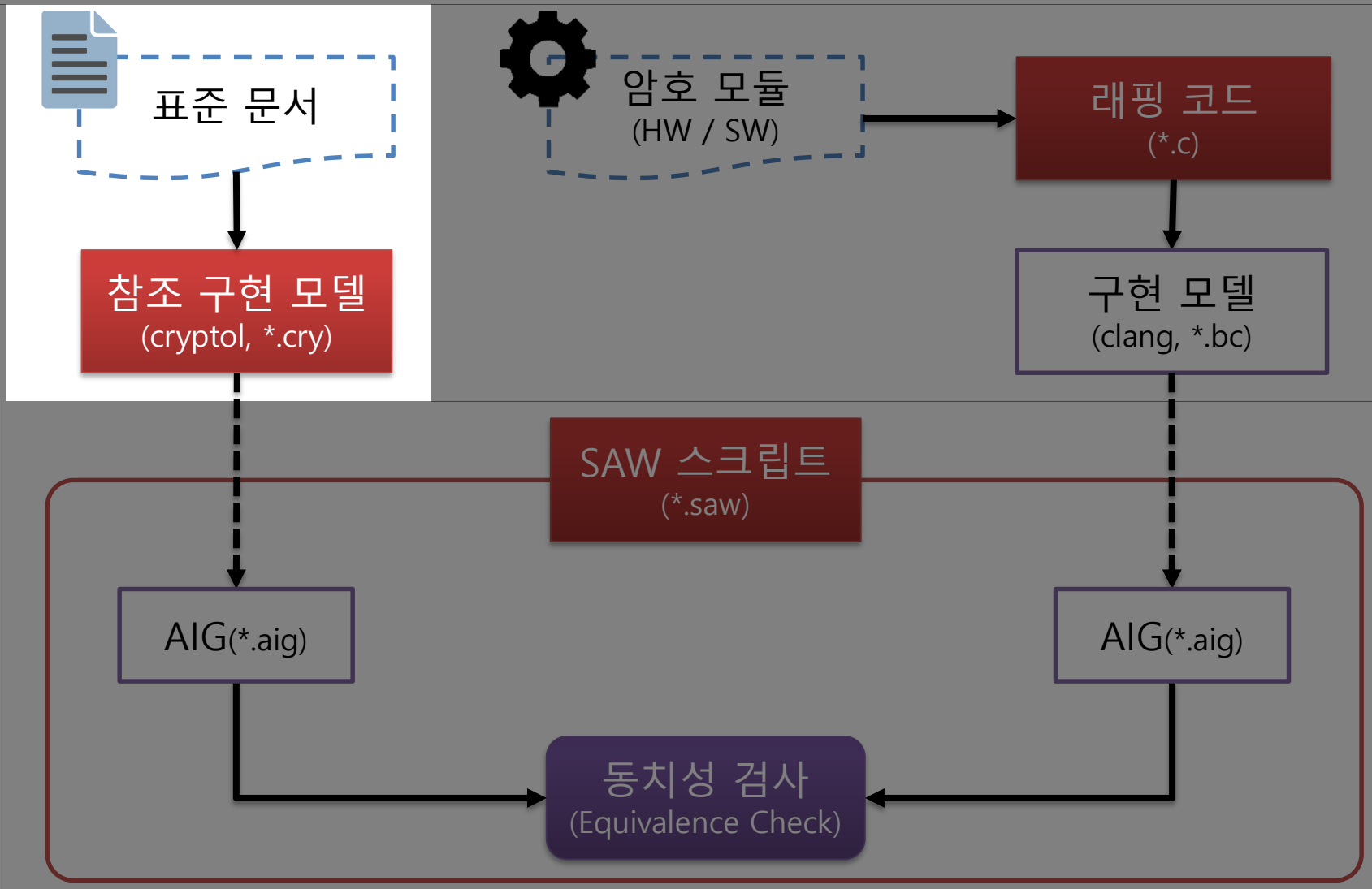
- 구현 및 검증 아키텍처



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

- 구현 및 검증 아키텍처



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

▪ 검증 프로세스(참조 구현 모델 - Cryptol)

5.3.2 라운드 키 생성 과정

$$ek_1 = (W_0) \oplus (W_1{}^{>>>19})$$

W_1, W_2, W_3를 조합하여 암호화 라운드
수는 암호키의 크기가 128, 192, 256
라운드 덧셈 계층이 두 번 있으
다. 암호화 라운드

$$ek_1 = (W_0) \oplus (W_1^{\gg 19}),$$
$$ek_3 = (W_2) \oplus (W_3^{\gg 19}),$$
$$ek_5 = (W_0) \oplus (W_1^{\gg 31}),$$
$$ek_7 = (W_2) \oplus (W_3^{\gg 31}),$$
$$ek_9 = (W_0) \oplus (W_1^{\ll 61}),$$
$$ek_{11} = (W_2) \oplus (W_3^{\ll 61}),$$
$$ek_{13} = (W_0) \oplus (W_1^{\ll 31}),$$
$$ek_{15} = (W_2) \oplus (W_3^{\ll 31}),$$
$$ek_{17} = (W_0) \oplus (W_1^{\ll 19})$$

위에서 설명한 키 확장 과정을 정리하면 (그림
여기서 ROT( )는 각 라운드별로 주어진 순환량에

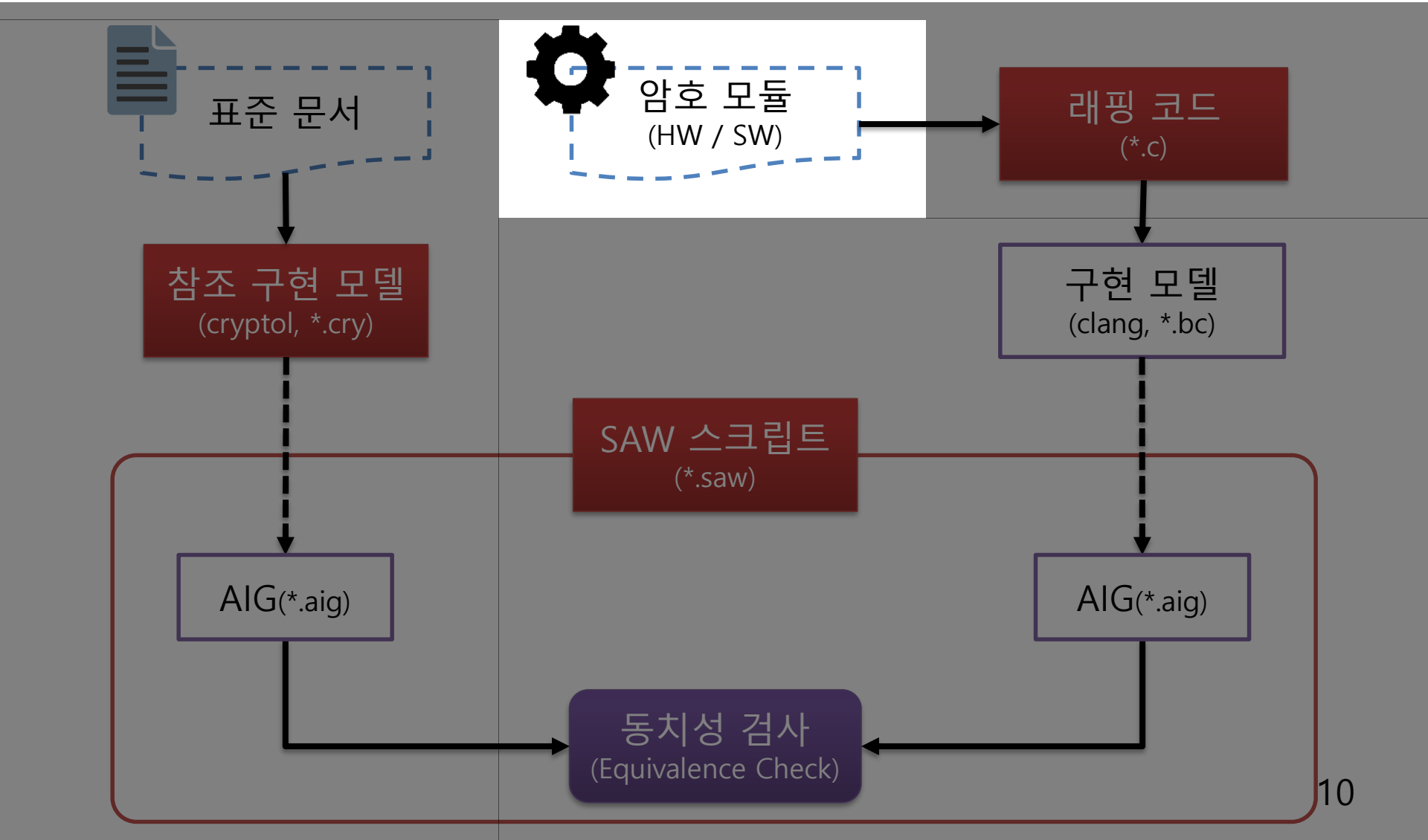
ek1 = (w0) ^ (w1 >>> 19)

```
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H)
generateEncKey : [128] -> [13][128]
generateEncKey masterKey = [ek1, ek2, ek3, ek4, ek5, ek6, ek7, ek8, ek9, ek10, e
k11, ek12, ek13]
        where
                w0    = masterKey
                w1    = oddFunction(w0, ck @ 0)  ^ zero : [128]
                w2    = evenFunction(w1, ck @ 1) ^ w0
                w3    = oddFunction(w2, ck @ 2)  ^ w1
                ek1   = (w0)          ^ (w1 >>> 19)
                ek2   = (w1)          ^ (w2 >>> 19)
                ek3   = (w2)          ^ (w3 >>> 19)
                ek4   = (w0 >>> 19) ^ (w3)
                ek5   = (w0)          ^ (w1 >>> 31)
                ek6   = (w1)          ^ (w2 >>> 31)
                ek7   = (w2)          ^ (w3 >>> 31)
                ek8   = (w0 >>> 31) ^ (w3)
                ek9   = (w0)          ^ (w1 <<< 61)
                ek10  = (w1)          ^ (w2 <<< 61)
                ek11  = (w2)          ^ (w3 <<< 61)
                ek12  = (w0 <<< 61) ^ (w3)
                ek13  = (w0)          ^ (w1 <<< 31)
                                                220,25-39     79%
```

2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

- 구현 및 검증 아키텍처



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

- 검증 프로세스(암호 모듈 - C)
  - Cryptography Module
    - KISA 32bit/8bit ARIA Module
    - hi-hee, minjeongJho ARIA Module in Github

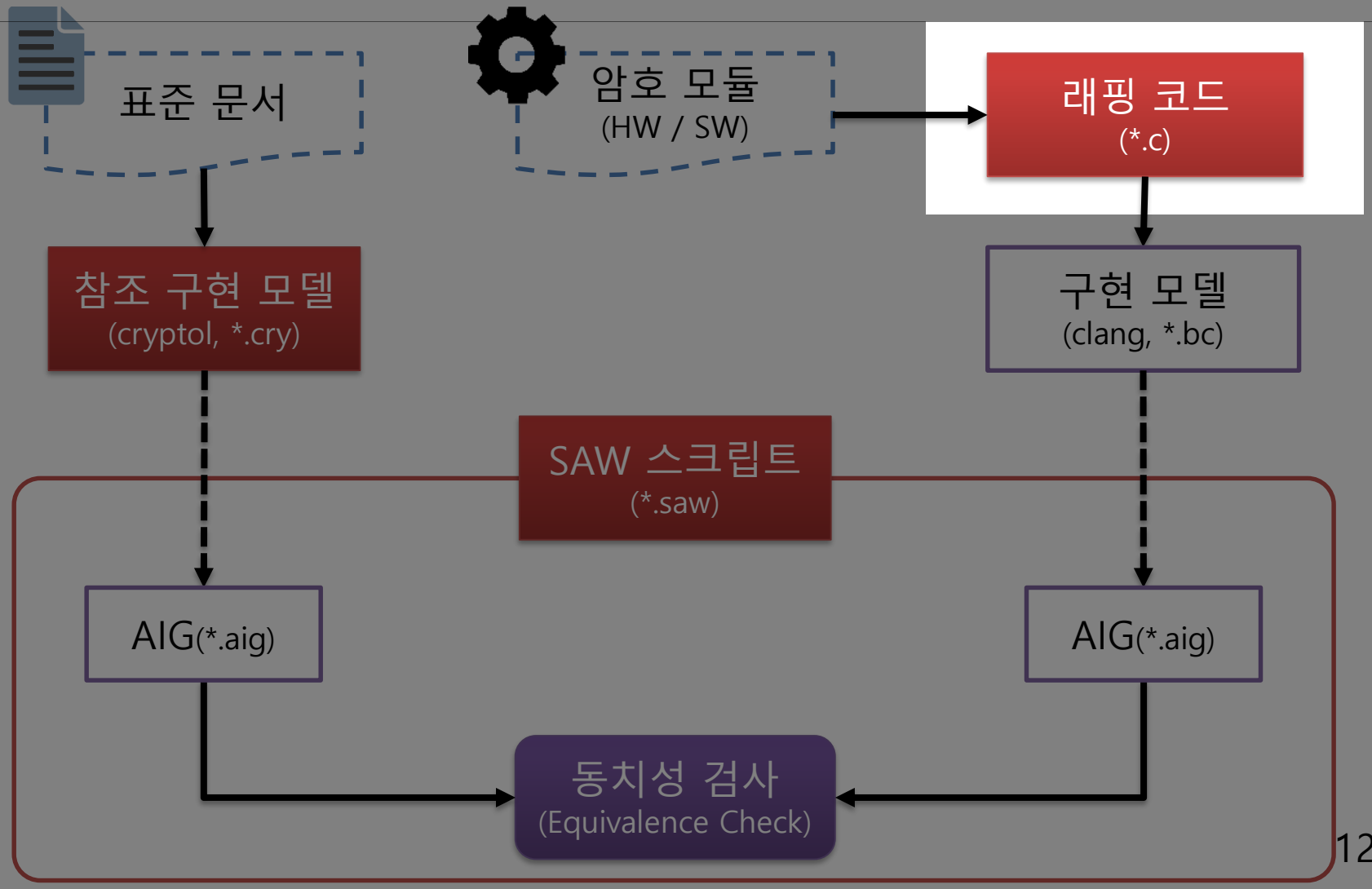
```c
#include <stdio.h>



void Crypt(const unsigned char *p, int R, const unsigned char *e, unsigned char *c);
int EncKeySetup(const unsigned char *w0, unsigned char *e, int keyBits);
int DecKeySetup(const unsigned char *w0, unsigned char *d, int keyBits);
```

```c
/* 암호화 함수.
 * const Byte *i: 입력
 * int Nr: 라운드 수
 * const Byte *rk: 라운드 키들
 * Byte *o: 출력
 */
void Crypt(const Byte *i, int Nr, const Byte *rk, Byte *o) {
  register Word t0, t1, t2, t3;

  WordLoad(WO(i,0), t0); WordLoad(WO(i,1), t1);
  WordLoad(WO(i,2), t2); WordLoad(WO(i,3), t3);

  if (Nr > 12) {KXL FO KXL FE}
  if (Nr > 14) {KXL FO KXL FE}
  KXL FO KXL FE KXL FO KXL FE KXL FO KXL FE
  KXL FO KXL FE KXL FO KXL FE KXL FO KXL
```

2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

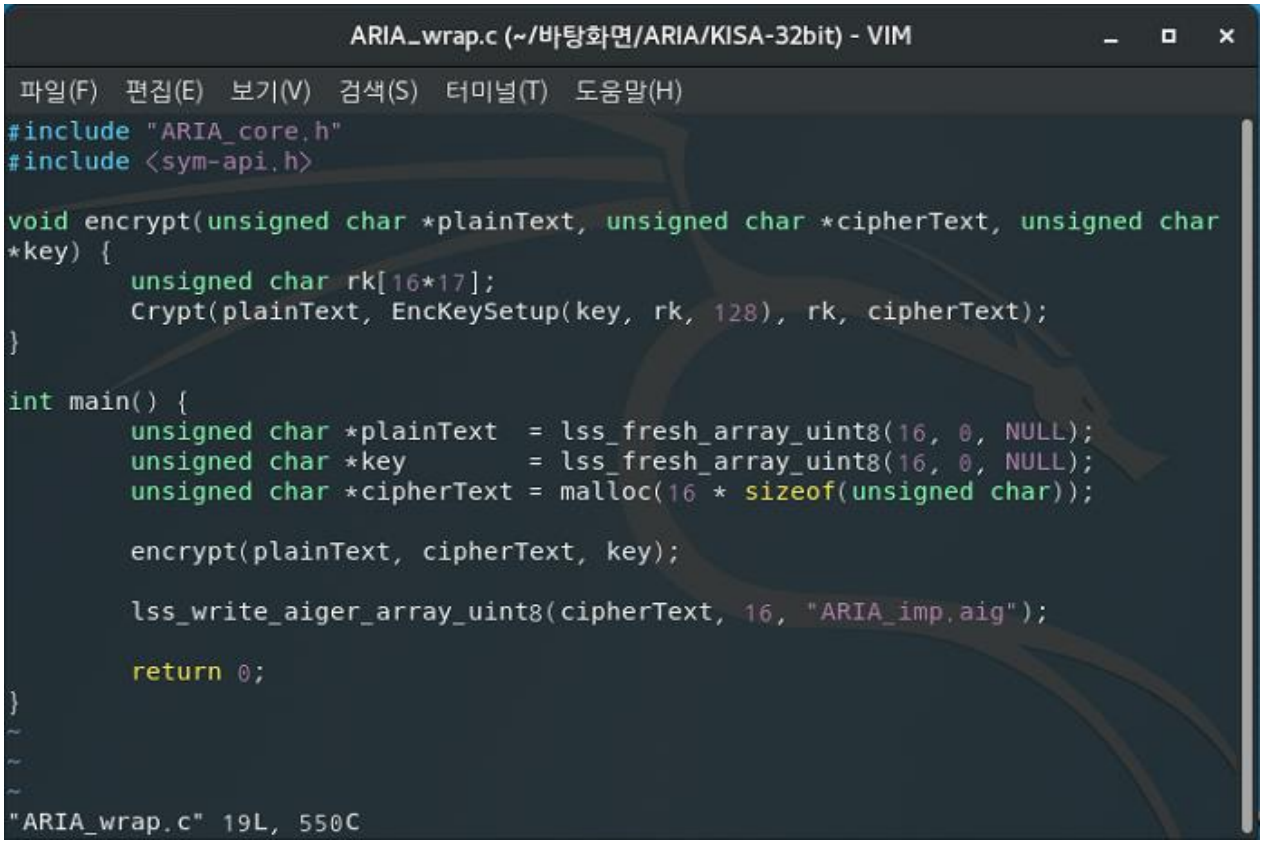
- 구현 및 검증 아키텍처



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

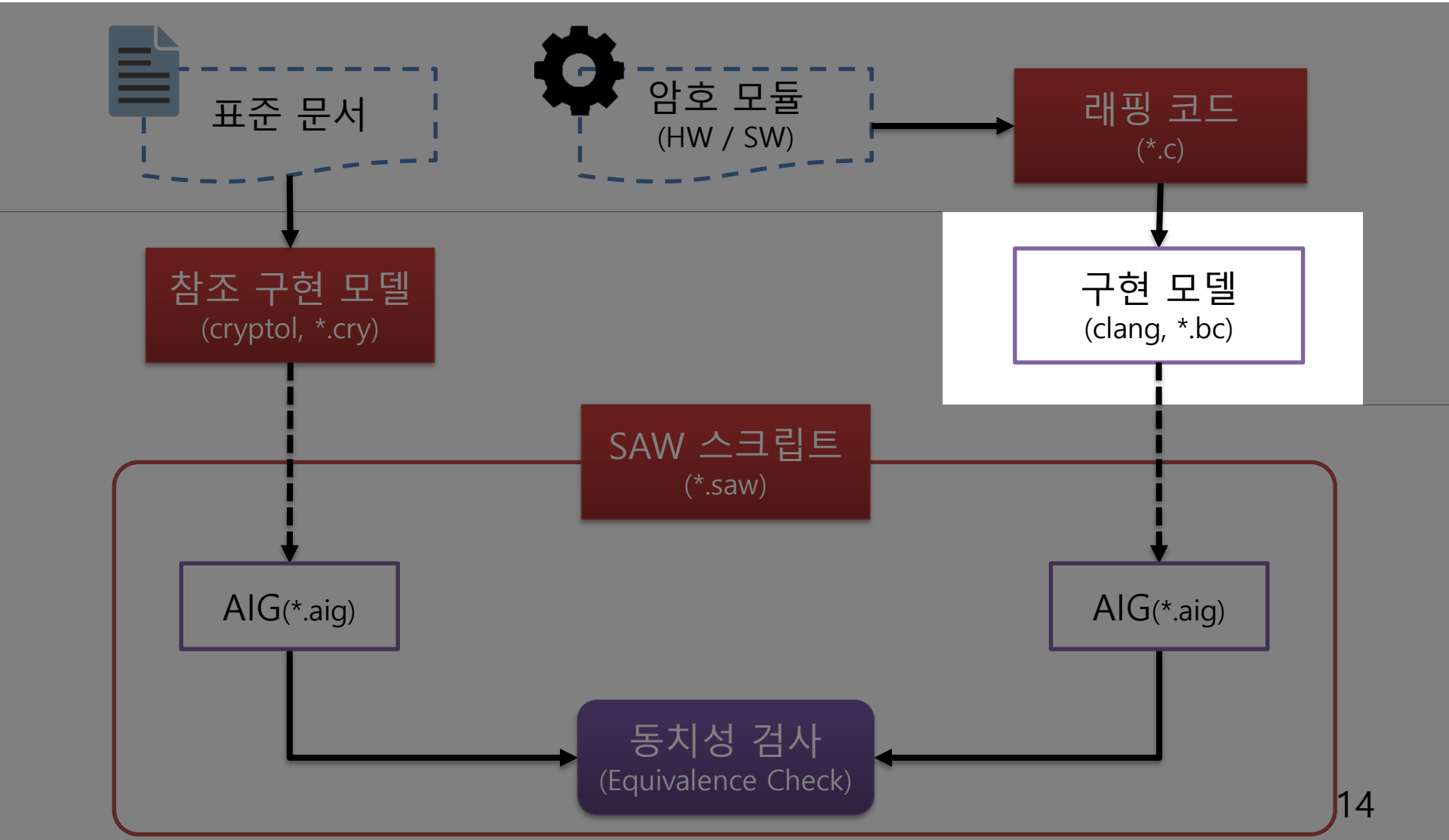
- 검증 프로세스(래핑 코드 - C)
  - Wrap Code
    - LSS(LLVM Symbolic Simulator) - "sym-api.h"



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

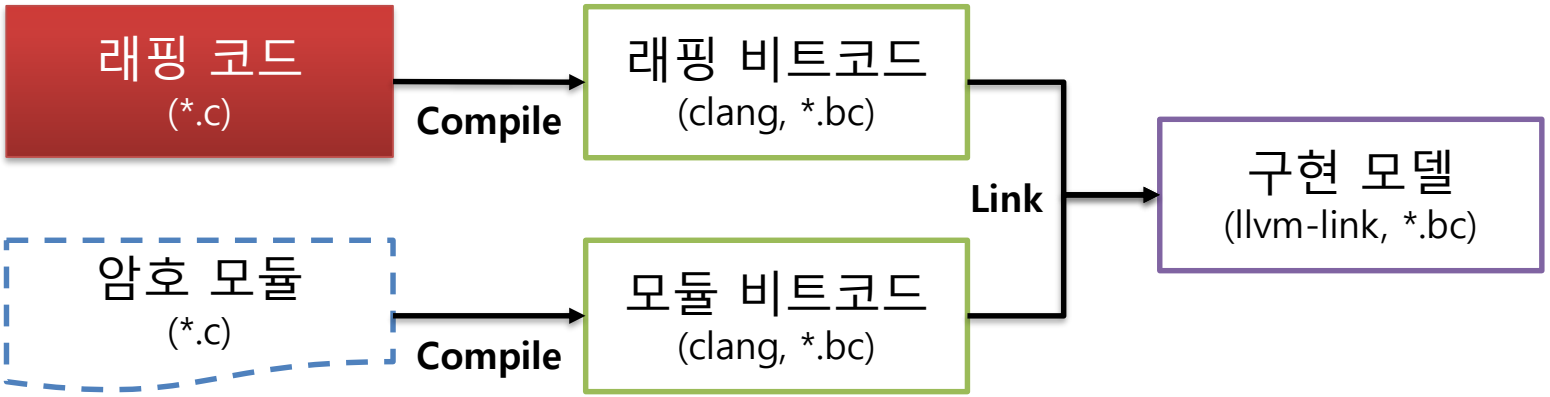
- 구현 및 검증 아키텍처



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

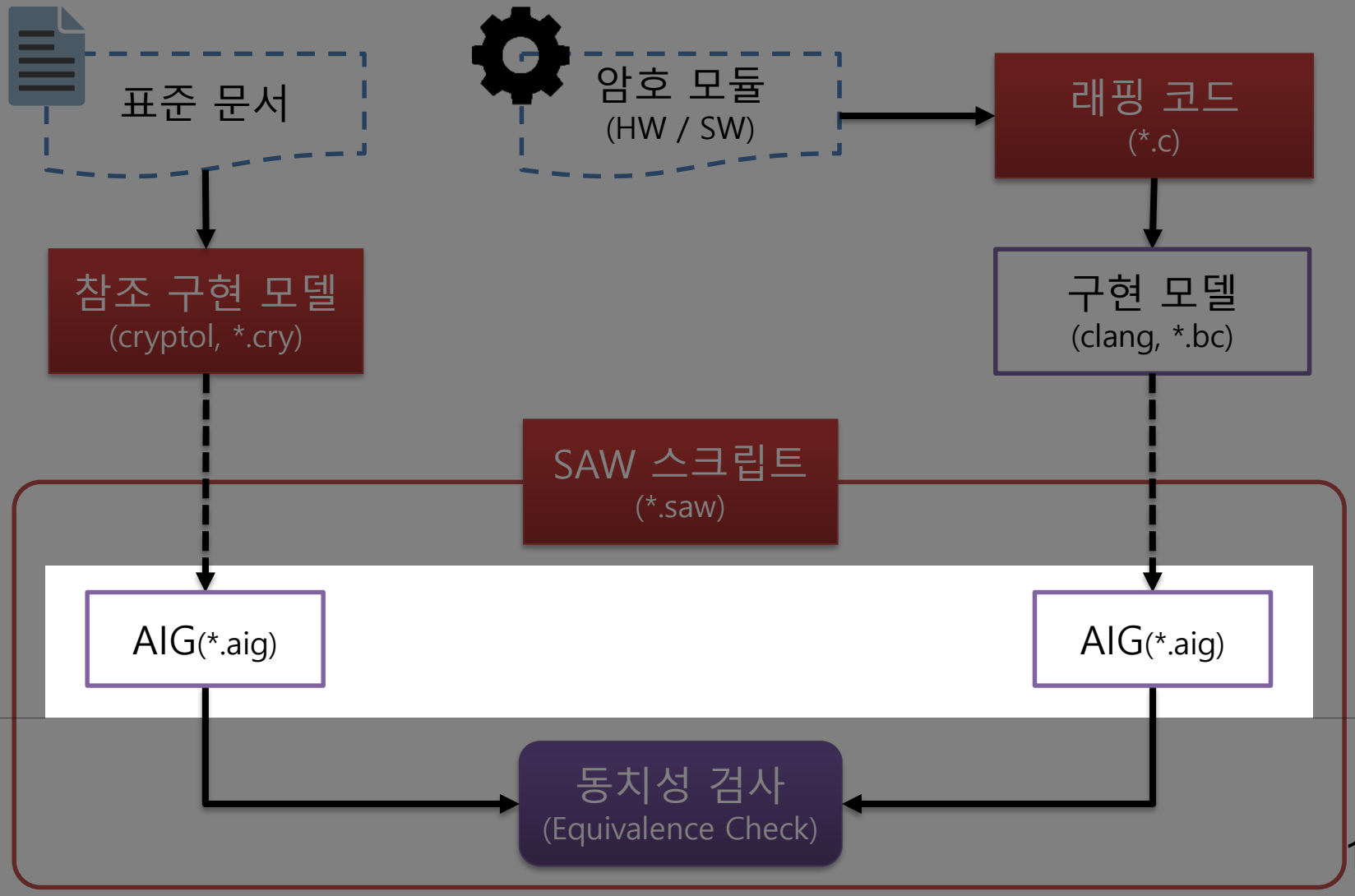
- 검증 프로세스(구현 모델 - Clang)
  - **Implementation Model**
    1. **compile ALL C PROGRAM using CLANG(LLVM)**
    2. **link ALL BIT CODE using LLVM-LINK**



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

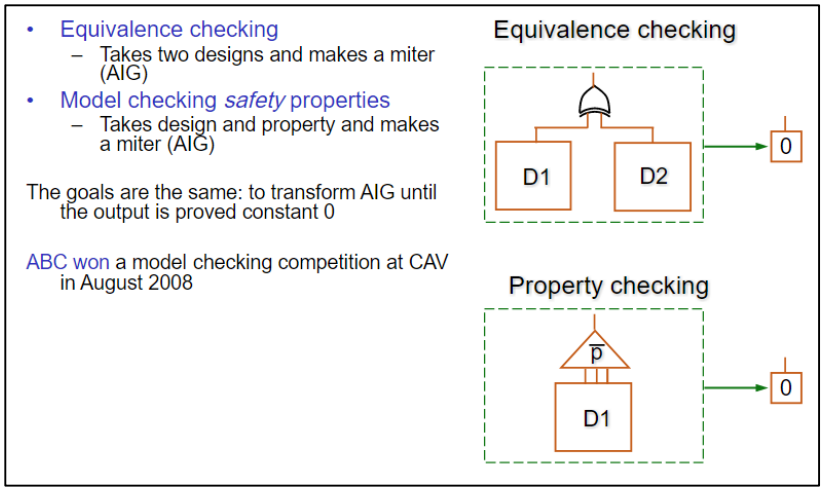
- 구현 및 검증 아키텍처

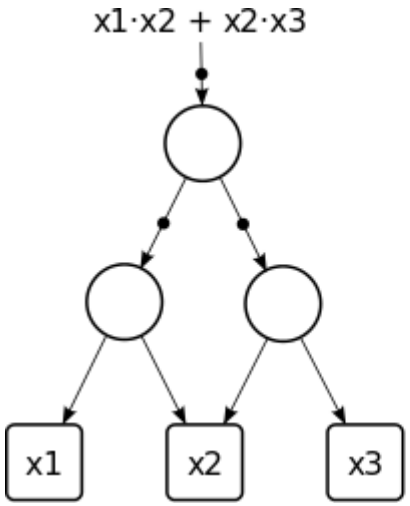


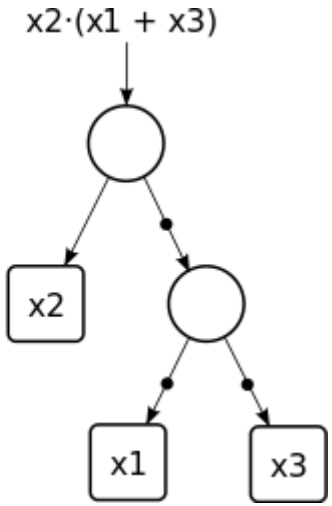
2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

- 검증 프로세스(AIG)
  - AIG(And-Inverter Graph)
    - Property Checking 및 Equivalence Checking을 위해 AIG 형태의 파일을 생성해야 함
      - Brayton, Robert, and Alan Mishchenko. "ABC: An academic industrial-strength verification tool." Computer Aided Verification. Springer Berlin/Heidelberg, 2010.
      - Biere, Armin. "The AIGER And-Inverter Graph (AIG) Format Version 20071012." (2007).



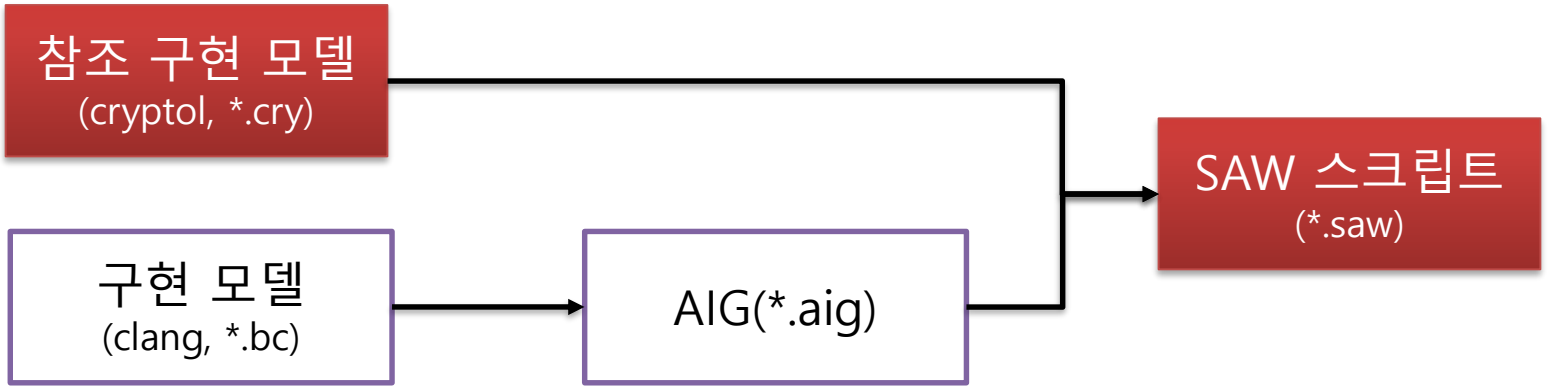




2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

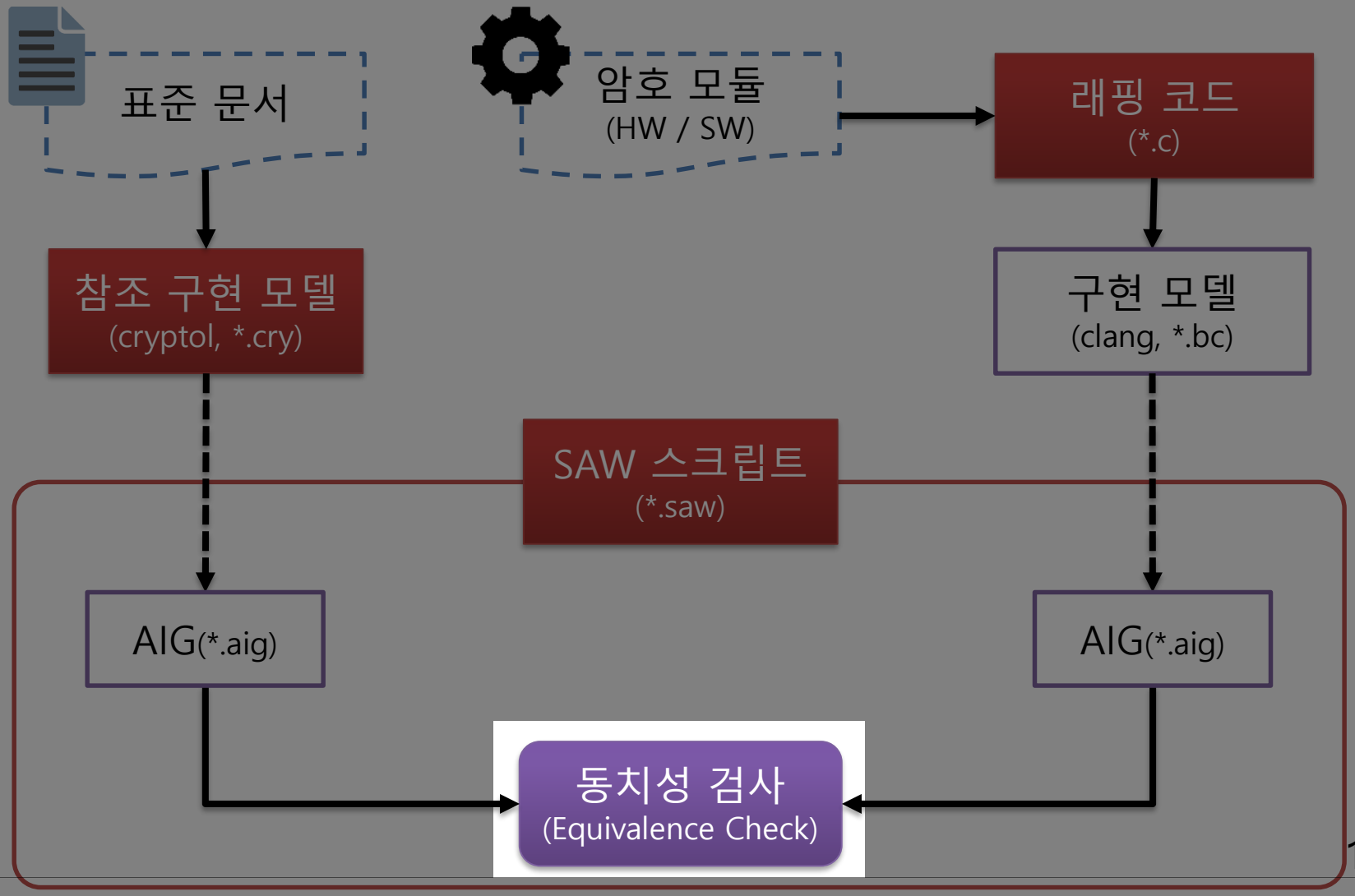
- 검증 프로세스(AIG)
  - 참조 구현 모델의 AIG 생성
    - 따로 생성하지 않아도 Cryptol을 사용하여 작성하였으면 추가적인 작업이 필요하지 않음
  - 구현 모델의 AIG 생성(AIG_{Impl} - LSS)
    - LSS(LLVM Symbolic Simulator)를 사용하여 구현 모델(*.bc)을 AIG파일로 변환
      - lss ARIA_imp.bc 입력



2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

- 구현 및 검증 아키텍처

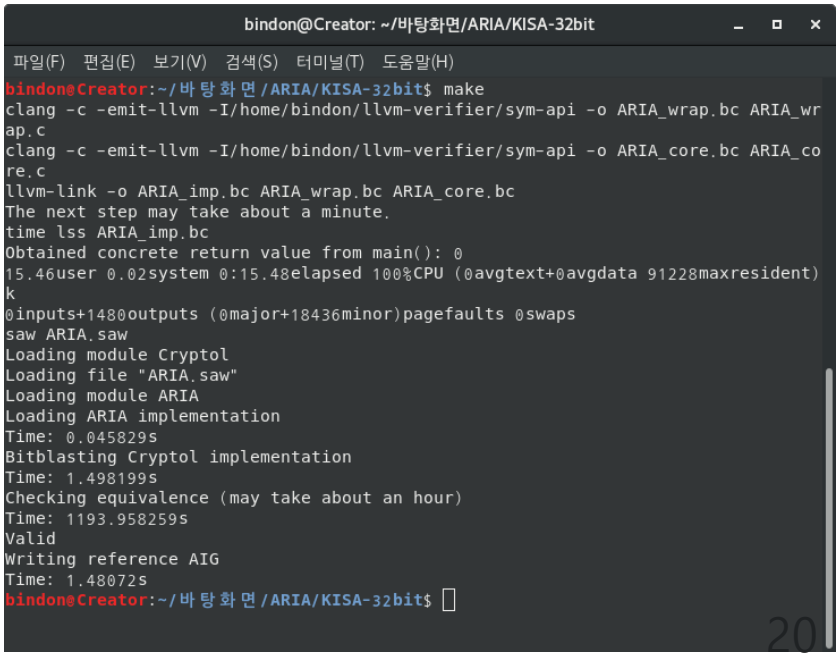


2. ARIA 구현 및 검증 프로세스

KOREA UNIVERSITY

- 검증 프로세스(동치성 검사 - SAW)
  - SAW Script
    1. 참조 구현 모델 import 및 래핑 코드 작성
    2. 구현 모델 AIG 로드
    3. CEC(Combinational Equivalence Checking) 수행

```
import "ARIA.cry";

let {{
  ariaExtract x = encrypt (plainText, key)
    where [plainText, key] = split x
}};

print "Loading ARIA implementation";
aes_imp <- time (load_aig "ARIA_imp.aig");

print "Bitblasting Cryptol implementation";
aes_ref <- time (bitblast {{ ariaExtract }});

print "Checking equivalence (may take about an hour)";
res <- time (cec aes_imp aes_ref);
print res;

print "Writing reference AIG";
time (write_aig "ARIA_ref.aig" {{ ariaExtract }});
```



3. 진행중인 연구

KOREA UNIVERSITY

- **Cryptol Ver.2 변경점**
  - **Cryptol Ver.1에서 변경된 사용법**

Cryptol version 2	Cryptol version 1	Summary
`[ False, True, True ] (==3)`	`[ False True True ] (== 6)`	Big-endian word representation
`[ 1, 1, 2, 3, 5 ]`	`[ 1 1 2 3 5 ]`	Commas separate sequence entries
`x = 1`	`x = 1;`	Uses *layout* instead of ;'s and {'s
`[ x | x <- [1 .. 10] ]`	`[| x || x <- [ 1 .. 10] |]`	Cleaner sequence constructor syntax
`f : {a,b} a -> b`	`f : {a b} a -> b`	Commas separate type variables
`take`{1} xs`	`take(1, xs)`	First-class type parameters
`x ^^ 2`	`x ** 2`	^^ for exponentiation
`<| x^^2 + 1 |>`	`<| x^2 + 1 |>`	Polynomial exponentiation now uniform
`[0 ..]:[_][8]`	`take(255, [0 ..]:[inf][8])`	Both produce `[0 .. 255]`
`[0 ...]:[inf][8]`	`[0 ..]:[inf][8]`	Both produce `[0 .. 255]` (repeated)
`[9, 8 .. 0]`	`[9 -- 0]`	Step defines decreasing sequences
`&&, ||, ^`	`&, |, ^`	Boolean operator syntax
`property foo xs=...`	`theorem foo: {xs}. xs==...`	Properties replace theorems (see below)

3. 진행중인 연구

KOREA UNIVERSITY

- **Property Checking**
  - **Haskell 언어로 작성된 CRY 파일의 검증에 사용**
  - ***property* 키워드를 이용하여 검증 방법 작성**
  - **Correctness 검증($D_k(E_k(P)) = P$)**

ARIA.cry

$$property\ ARIACorrect\ plainText\ key = decrypt\ (encrypt\ (plainText, key), key) ==\ plainText$$

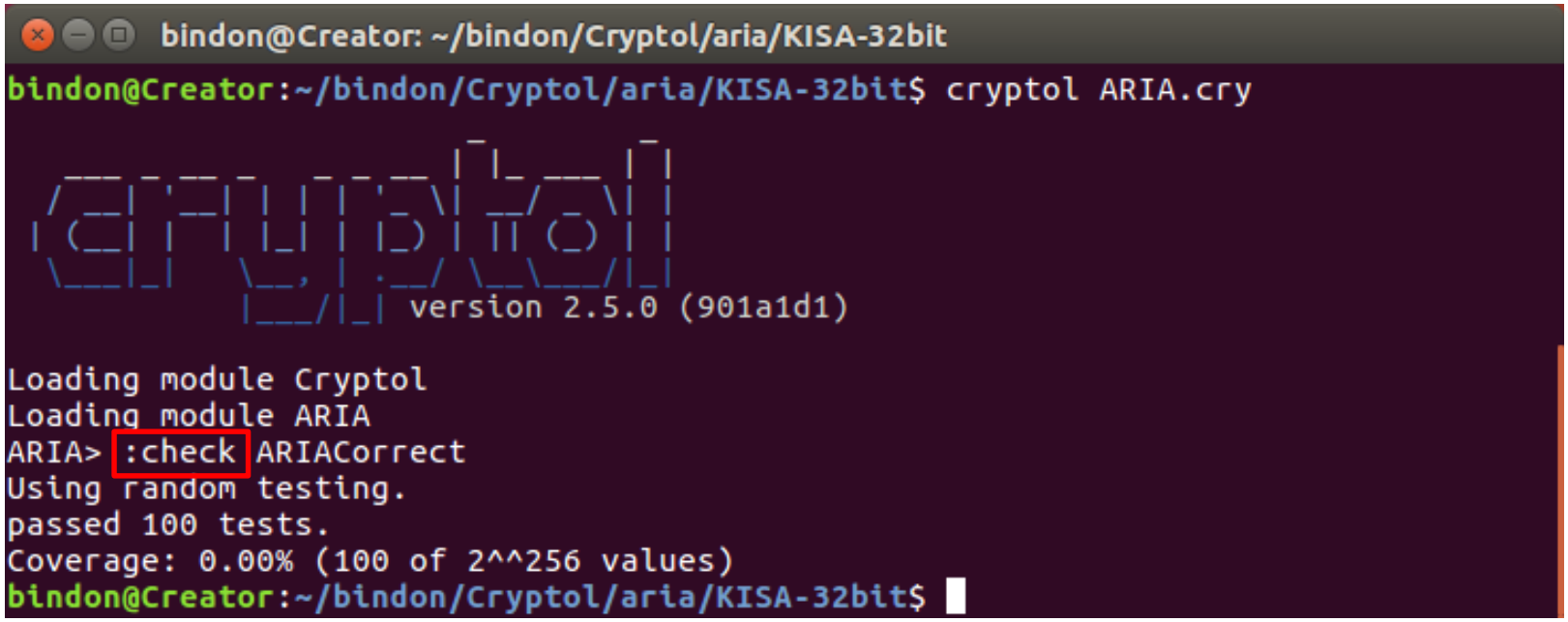
Cryptol

```
$ cryptol ARIA.cry
ARIA> :prove ARIACorrect
```

3. 진행중인 연구

KOREA UNIVERSITY

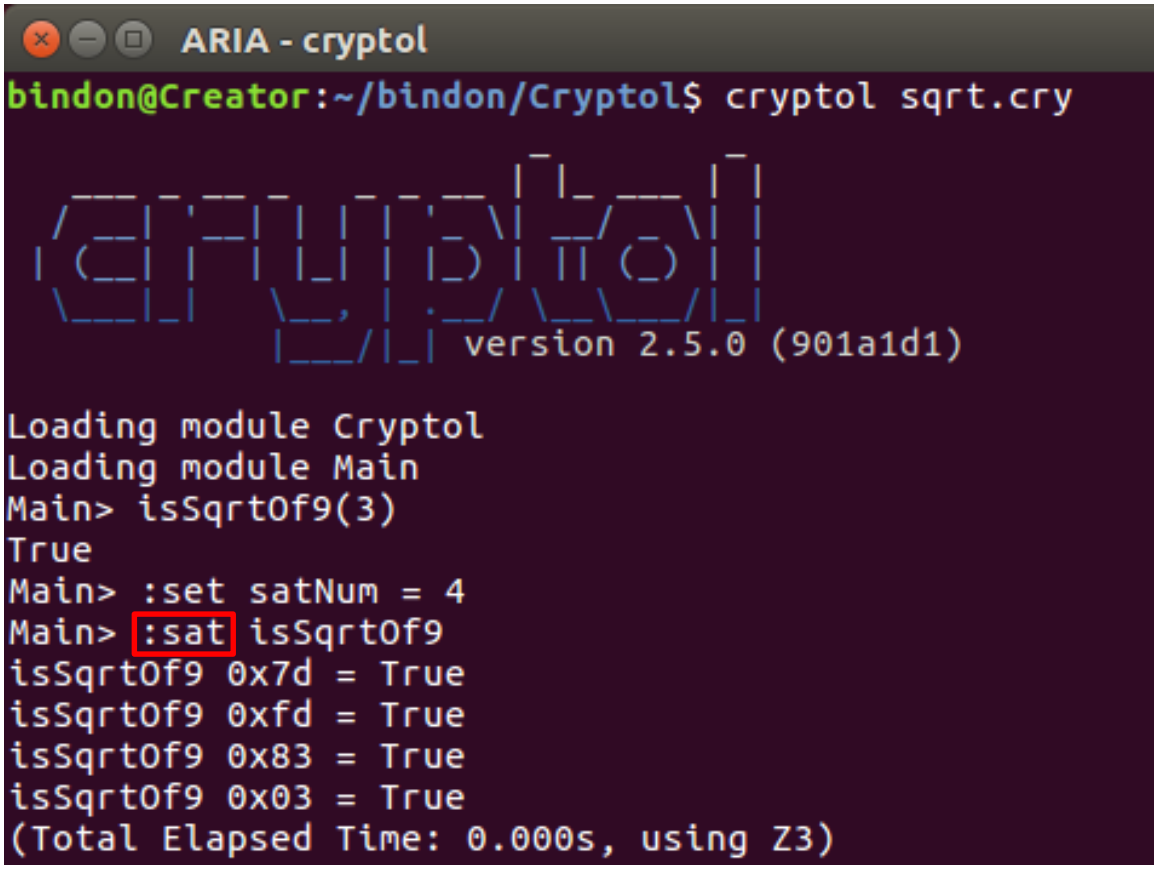
- **Property Checking(check)**
  - 기본 **100**개의 랜덤 한 값으로 테스트 수행
  - **:set tests=<개수>** 옵션을 통해 변경 가능
  - 코드를 작성하면서 신속한 테스트 시 사용
  - **:prove**와는 달리 증명은 불가능 함



3. 진행중인 연구

KOREA UNIVERSITY

- **Property Checking(sat)**
  - **SAT Solver(해당사항 없음)**
    - **True가 도출되기 위한 매개변수를 추정**



3. 진행중인 연구

KOREA UNIVERSITY

- **Property Checking(prove)**
  - **SAT/SMT Solver 사용**
  - **기본적으로 z3를 사용하지만 외부 SAT/SMT Solver를 사용할 수 있음**

```
bindon@Creator: ~/bindon/Cryptol/proveTest
bindon@Creator:~/bindon/Cryptol/proveTest$ cryptol proveTest.cry
                        _        _
   ___ _ __ _   _ _ __ | |_ ___ | |
  / __| '__| | | | '_ \| __/ _ \| |
 | (__| |  | |_| | |_) | || (_) | |
  \___|_|   \__, | .__/ \__\___/|_|
            |___/|_| version 2.5.0 (901a1d1)

Loading module Cryptol
Loading module Main
Main> :prove
:prove FG
        Q.E.D.
(Total Elapsed Time: 0.008s, using Z3)
:prove FH
        FH 0x00 = False
(Total Elapsed Time: 0.006s, using Z3)
Main>
```

```
bindon@Creator: ~/bindon/Cryptol/proveTest
f, g, h : [8] -> [8]
f x = (x-1)*(x+1)
g x = x*x - 1
h x = x*x + 1
property FG x = f x == g x
property FH x = f x == h x
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Previous Version)**
  - 코드에 구현 상의 취약점이 존재하는지 확인
  - 기존에는 **:safe** 키워드를 통한 검사가 가능 했었음
  - **lookup** 함수
    - 매개변수 **xs**는 **4**개의 배열, **i**는 **2비트** 숫자
      - **i**는 **2**비트이므로 **0~3**의 값을 가질 수 있음
      - **xs**는 **4**개의 배열이기 때문에 모든 **i**에 대해 안전**(safe)**

lookup.cry

```
lookup : ([4], [2]) -> Bit;
lookup(xs, i) = xs @ i;
```

Example

```
lookup> :set sbv
lookup> :safe lookup
"lookup" is safe; no safety violations exist.
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Previous Version)**
  - **lookup2 함수**
    - 매개변수 **xs**는 **4**개의 배열, **i**는 **3비트** 숫자
      - **i**는 **3**비트이므로 **0~7**의 값을 가질 수 있음
      - **xs**는 **4**개의 배열이기 때문에 **4** 이상의 **i**에 대해 안전하지 않음**(unsafe)**

lookup2.cry

```
lookup2 : ([4], [3]) -> Bit;
lookup2(xs, i) = xs @ i;
```

Example

```
lookup2> :safe lookup2
*** 1 safety condition to be checked.
*** Violation detected:
lookup (0, 4) = "lookup2.cry", line 2, col 20: index of 4 is out of
bounds (valid range is 0 thru 3).
*** 1 problem found.
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Previous Version)**
  - **lookup3 함수**
    - 매개변수 **xs**는 **4**개의 배열, **i**는 **3비트** 숫자
      - **i**는 **3**비트이므로 **0~7**의 값을 가질 수 있음
      - **xs**는 **4**개의 배열이기 때문에 **4** 이상의 **i**에 안전하지 않았으나 예외처리를 수행하여 안전**(safe)**

lookup3.cry

```
lookup3 : ([4], [3]) -> Bit;
lookup3 (xs, i) = if i >= 3 then False else xs @ i;
```

Example

```
lookup3> :safe lookup3
*** 1 safety condition to be checked.
*** Verified safe.
*** All safety checks pass, safe to execute.
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Current Version)**
  - 현재는 safe 키워드가 삭제됨
  - lookup 함수
    - 매개변수 xs는 4개짜리 배열, i는 2비트 숫자
      - i는 2비트이므로 0~3의 값을 가질 수 있음
      - xs는 4개의 배열이기 때문에 모든 i에 대해 안전(safe)
    - check를 이용해 모든 경우의 수 테스트
      - 배열 2비트 4개, index 2비트 : $(2^2)^4 \times 2^2 = 1,024$

lookup.cry

```
lookup : ([4], [2]) -> Bit;
lookup(xs, i) = xs @ i;
```

Example

```
lookup> :set sbv
lookup> :safe lookup
"lookup" is safe; no safety violations exist.
```

```
lkup : ([4][2], [2]) -> [2]
lkup (xs, i) = xs @ i

property safeTest(xs, i) = lkup (xs, i) == xs @ i
```

Main - cryptol

```
bindon@Creator:~/bindon/Cryptol/safeTest$ cryptol safeTest.cry
                                    _        _
   ___ _ __ _   _ _ __ | |_ ___ | |
  / __| '__| | | | '_ \| __/ _ \| |
 | (__| |  | |_| | |_) | || (_) | |
  \___|_|   \__, | .__/ \__\___/|_|
            |___/|_| version 2.5.0 (901a1d1)

Loading module Cryptol
Loading module Main
Main> :set tests=1024
Main> :check
property safeTest Using exhaustive testing.
passed 1024 tests.
Q.E.D.
Main>
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Current Version)**
  - **lookup2 함수**
    - 매개변수 **xs**는 **4**개짜리 배열, **i**는 **3비트** 숫자
      - **i**는 **3**비트이므로 **0~7**의 값을 가질 수 있음
      - **xs**는 **4**개의 배열이기 때문에 **4** 이상의 **i**에 대해 안전하지 않음**(unsafe)**

```
lkup2 : ([4][2], [3]) -> [2]
lkup2 (xs, i) = xs @ i

property safeTest(xs, i) = lkup2 (xs, i) == xs @ i
```

lookup2.cry

```
lookup2 : ([4], [3]) -> Bit;
lookup2(xs, i) = xs @ i;
```

Example

```
lookup2> :safe lookup2
*** 1 safety condition to be checked.
*** Violation detected:
lookup (0, 4) = "lookup2.cry", line 2, col 20: index of 4 is out of
bounds (valid range is 0 thru 3).
*** 1 problem found.
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Current Version)**
  - **lookup2 함수**
    - **check를 이용해 모든 경우의 수 테스트**
      - **배열 2비트 4개, index 3비트 :** $(2^2)^4 \times 2^3 = 2,048$
      - **array index out of bounds exception 발생**
      - **but, 테스트케이스가 너무 적으면 검증이 성공하는 일도 있음(i가 4 미만일 경우) -> 현재 암호 검증의 문제점**

```
bindon@Creator:~/bindon/Cryptol/safeTest$ cryptol safeTest.cry
                                        _
                                       | |
   ___ _ __ _   _ _ __ | |_ ___  | |
  / __| '__| | | | '_ \| __/ _ \ | |
 | (__| |  | |_| | |_) | || (_) | |
  \___|_|   \__, | .__/ \__\___/|_|
             |___/|_| version 2.5.0

Loading module Cryptol
Loading module Main
Main> :check
property safeTest Using random testing.
ERROR for the following inputs:
([0x0, 0x2, 0x3, 0x1], 0x7)
invalid sequence index: 7
Main> :set tests=1
Main> :check
property safeTest Using random testing.
passed 1 tests.
Coverage: 0.05% (1 of 2048 values)
```

인덱스가 0~3인 배열에 7을 참조하여 오류 발생

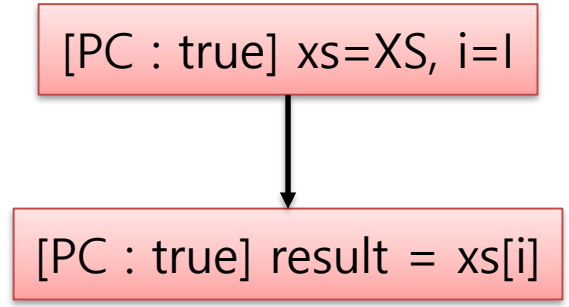
인덱스가 0~3인 배열에 0~3의 어떤 값이 임의로 선택되어 통과 (Coverage가 100%가 아니기 때문에 Q.E.D가 아님)

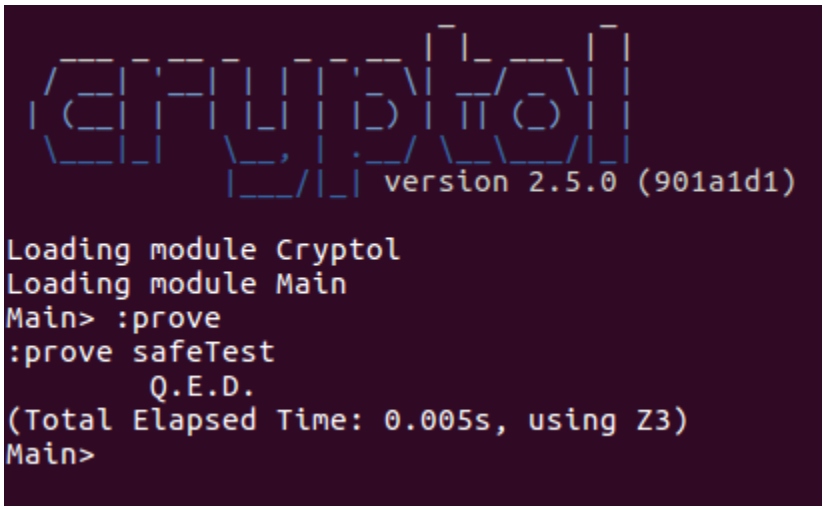
3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Current Version)**
  - **lookup2 함수**
    - **prove를 이용한 증명(SMT/SAT Solver)**
      - **Symbolic Execution을 수행하면 lkup2 (xs, i)가 xs @ i와 심볼이 동일하기 때문에 검증 완료됨(but, unsafe)**

```
lkup2 : ([4][2], [3]) -> [2]
lkup2 (xs, i) = xs @ i

property safeTest(xs, i) = lkup2 (xs, i) == xs @ i
```





3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Current Version)**
  - **lookup3 함수**
    - 매개변수 **xs**는 **4**개의 배열, **i**는 **3비트** 숫자
      - **i**는 **3**비트이므로 **0~7**의 값을 가질 수 있음
      - **xs**는 **4**개의 배열이기 때문에 **4** 이상의 **i**에 안전하지 않았으나 예외처리를 수행하여 안전**(safe)**

```
lkup3 : ([4][2], [3]) -> [2]
lkup3 (xs, i) = if i > 3 then (xs @ (i % 4)) else (xs @ i)

property safeTest(xs, i) = lkup3 (xs, i) == (if i > 3 then (xs @ (i % 4)) else (xs @ i))
```

lookup3.cry

```
lookup3 : ([4], [3]) -> Bit;
lookup3 (xs, i) = if i >= 3 then False else xs @ i;
```

Example

```
lookup3> :safe lookup3
*** 1 safety condition to be checked.
*** Verified safe.
*** All safety checks pass, safe to execute.
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Current Version)**
  - **lookup3 함수**
    - **check를 이용해 모든 경우의 수 테스트**
      - **배열 2비트 4개, index 3비트 :** $(2^2)^4 \times 2^3 = 2,048$
      - **예외처리를 수행하였기 때문에 검증이 정상적으로 수행**

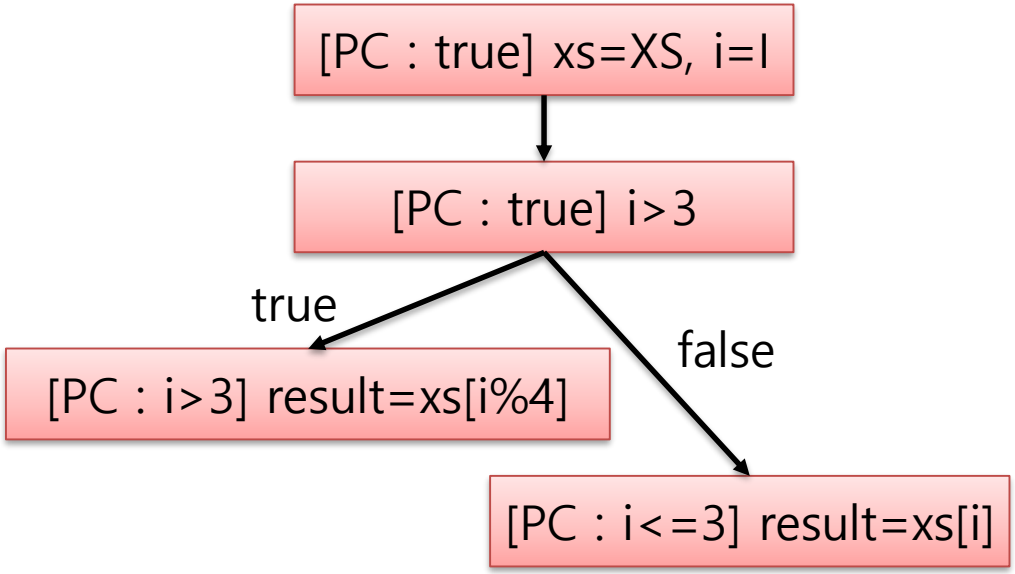
```
bindon@Creator:~/bindon/Cryptol/safeTest$ cryptol safeTest.cry
                                    _           _
   ___ _ __ _   _ _ __ | |_ ___ | |
  / __| '__| | | | '_ \| __/ _ \| |
 | (__| |  | |_| | |_) | || (_) | |
  \___|_|   \__, | .__/ \__\___/|_|
            |___/|_| version 2.5.0 (901a1d1)

Loading module Cryptol
Loading module Main
Main> :set tests=2048
Main> :check
property safeTest Using exhaustive testing.
passed 2048 tests.
Q.E.D.
Main>
```

3. 진행중인 연구

KOREA UNIVERSITY

- **Safety Checking(Current Version)**
  - **lookup3 함수**
    - **prove를 이용한 검증**
      - **Symbolic Execution을 수행하면 심볼이 동일하기 때문에 검증 완료됨**

```
lkup3 : ([4][2], [3]) -> [2]
lkup3 (xs, i) = if i > 3 then (xs @ (i % 4)) else (xs @ i)

property safeTest(xs, i) = lkup3 (xs, i) == (if i > 3 then (xs @ (i % 4)) else (xs @ i))
```

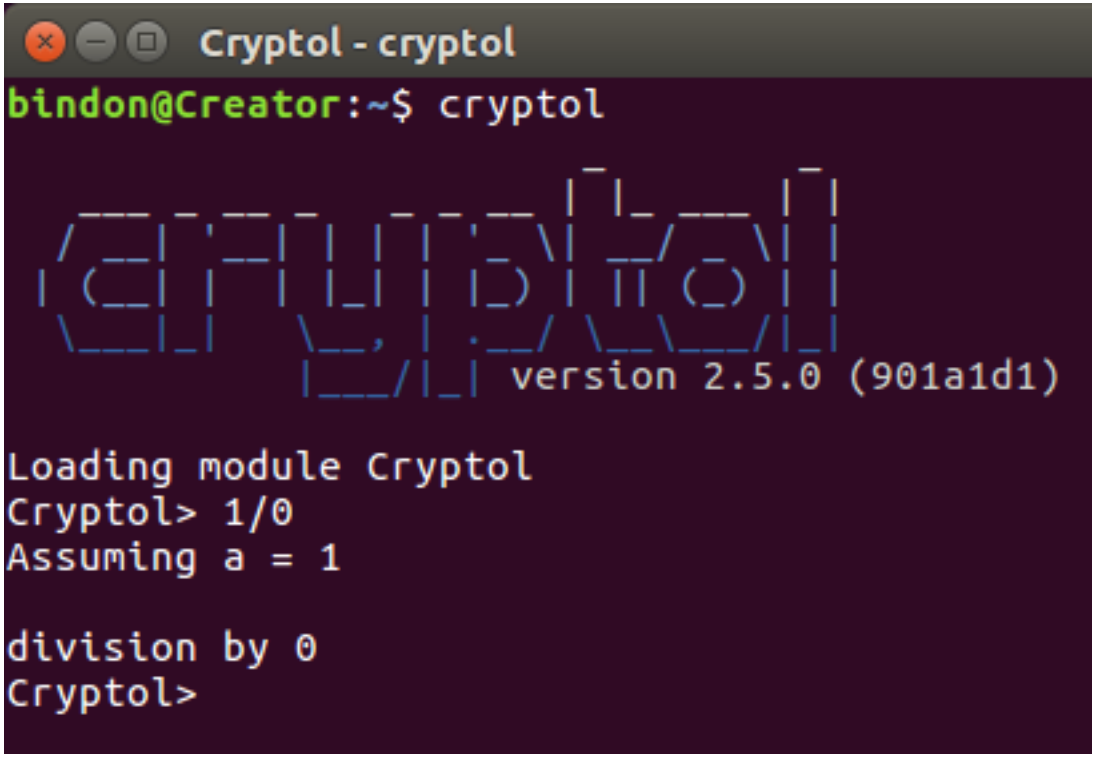



```
                                   _           _
 ___ _ __ _   _ _ __ | |_ ___ | |
/ __| '__| | | | '_ \| __/ _ \| |
| (__| |  | |_| | |_) | || (_) | |
 \___|_|   \__, | .__/ \__\___/|_|
           |___/|_| version 2.5.0 (901a1d1)

Loading module Cryptol
Loading module Main
Main> :prove
:prove safeTest
        Q.E.D.
(Total Elapsed Time: 0.004s, using Z3)
Main>
```

3. 진행중인 연구

KOREA UNIVERSITY

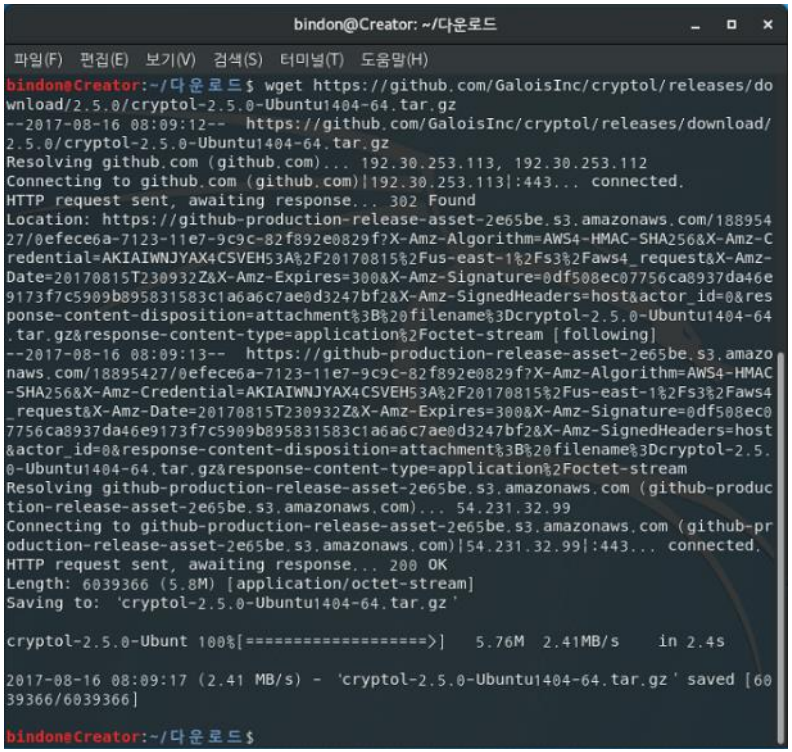
- **Safety Checking(Current Version)**
  - **division by zero**의 경우 아래와 같이 자동으로 검사됨
  - 변수에 의한 오류 발생도 동일하게 처리



4. Cryptol 환경 구축

KOREA UNIVERSITY

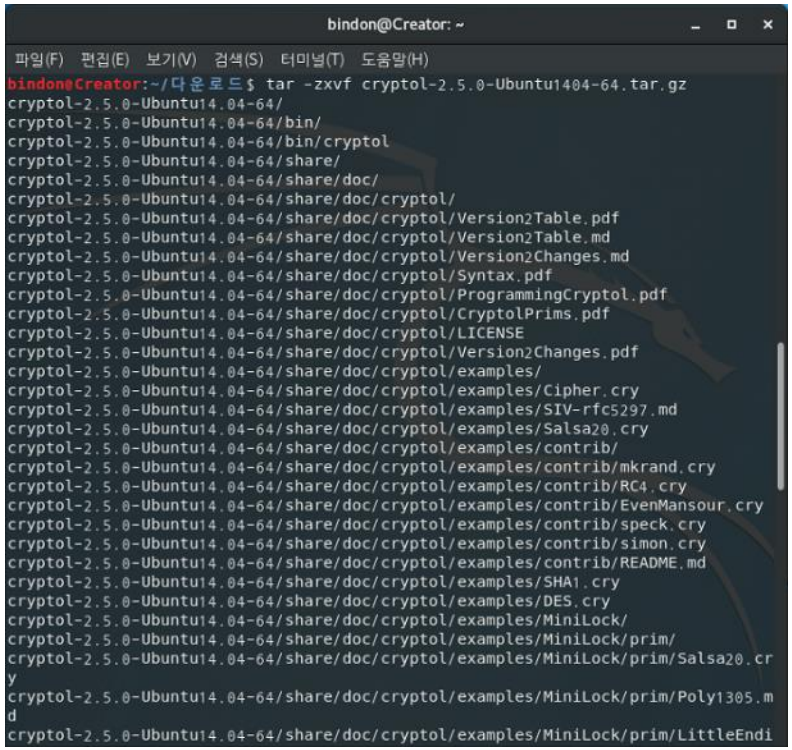
- **Cryptol 설치**
  - **Cryptol 다운로드**
    - **https://cryptol.net/downloads.html**
    - **작성 시점으로 cryptol 2.5.0 Ubuntu 14.04-64 사용**

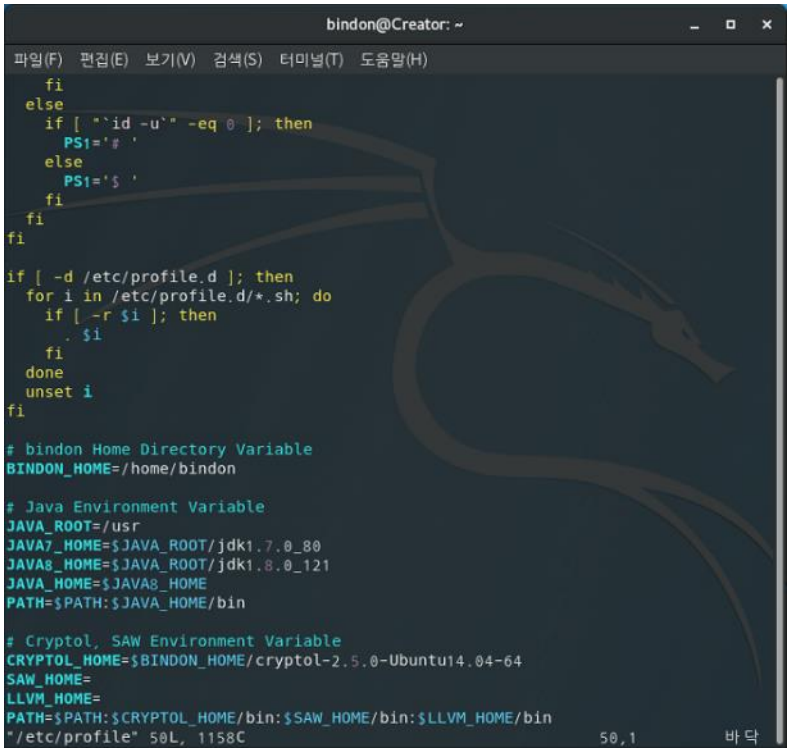


4. Cryptol 환경 구축

KOREA UNIVERSITY

- **Cryptol 설치**
  - **Cryptol 압축 해제 및 환경변수 등록**
    - **tar -zxvf cryptol-2.5.0-Ubuntu1404-64.tar.gz**





4. Cryptol 환경 구축

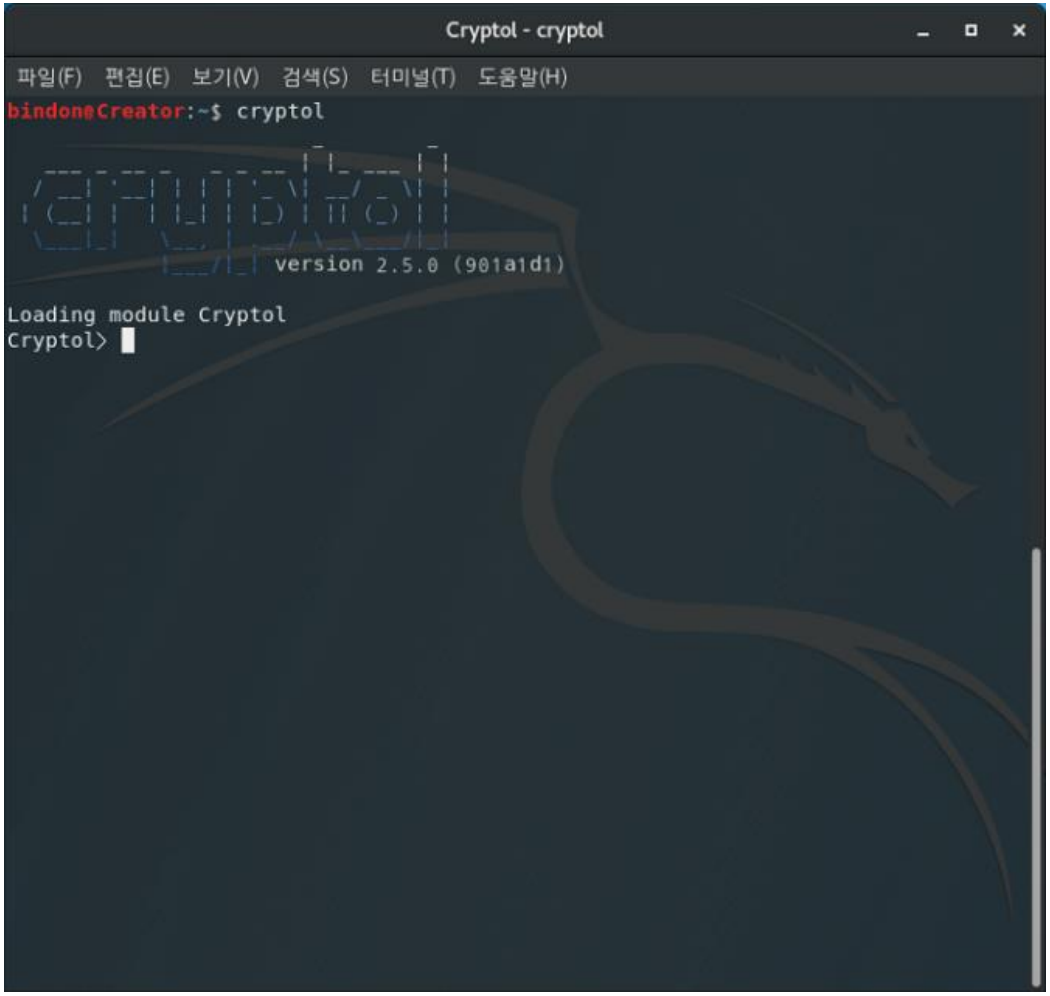
KOREA UNIVERSITY

- **Cryptol 설치**
  - **Cryptol을 사용하기 위해 Dependency Package인 Z3 SMT Solver 설치 필요**
  - **Package Manager를 이용한 z3 설치**
    - **/usr/bin에 자동으로 설치되는 편리한 방법**
    - **apt-get install z3**
    - **debian 계열이 아닐 경우 각 Package Manager를 통해 설치 수행**
  - **직접 다운로드를 통한 Z3 설치**
    - **환경변수 등록 필요**
    - **https://github.com/Z3Prover/z3/releases**

4. Cryptol 환경 구축

KOREA UNIVERSITY

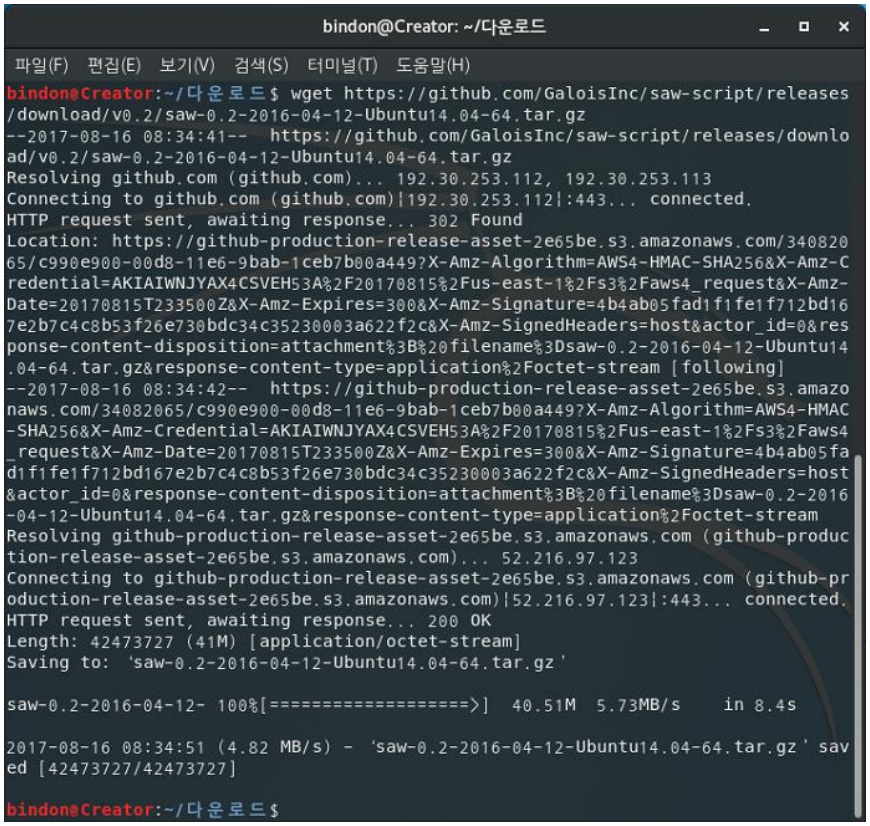
- **Cryptol 설치**
  - **Cryptol 실행**



4. Cryptol 환경 구축

KOREA UNIVERSITY

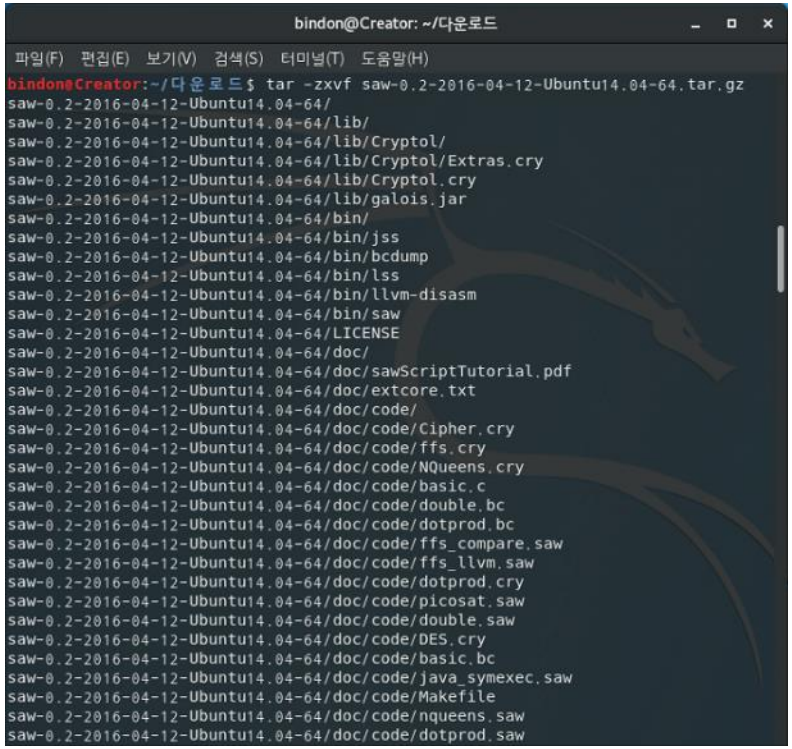
- **SAW 설치**
  - **SAW 다운로드**
    - **https://saw.galois.com/downloads.html**
    - **작성 시점으로 saw 0.2 Ubuntu 14.04-64 사용**

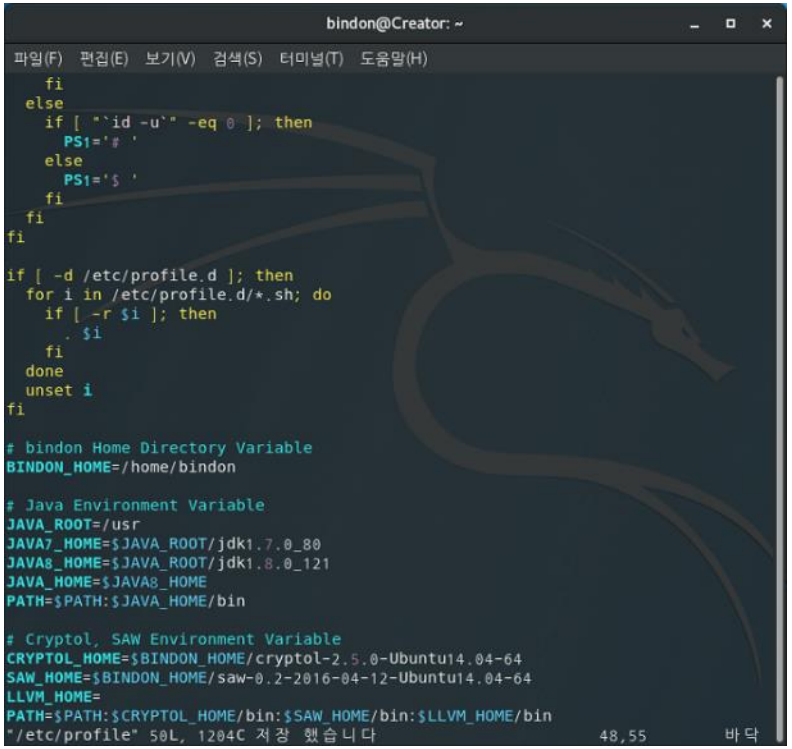


4. Cryptol 환경 구축

KOREA UNIVERSITY

- **SAW 설치**
  - **SAW 압축 해제 및 환경변수 등록**
  - **tar -zxvf saw-0.2-2016-04-12-Ubuntu14.04-64.tar.gz**

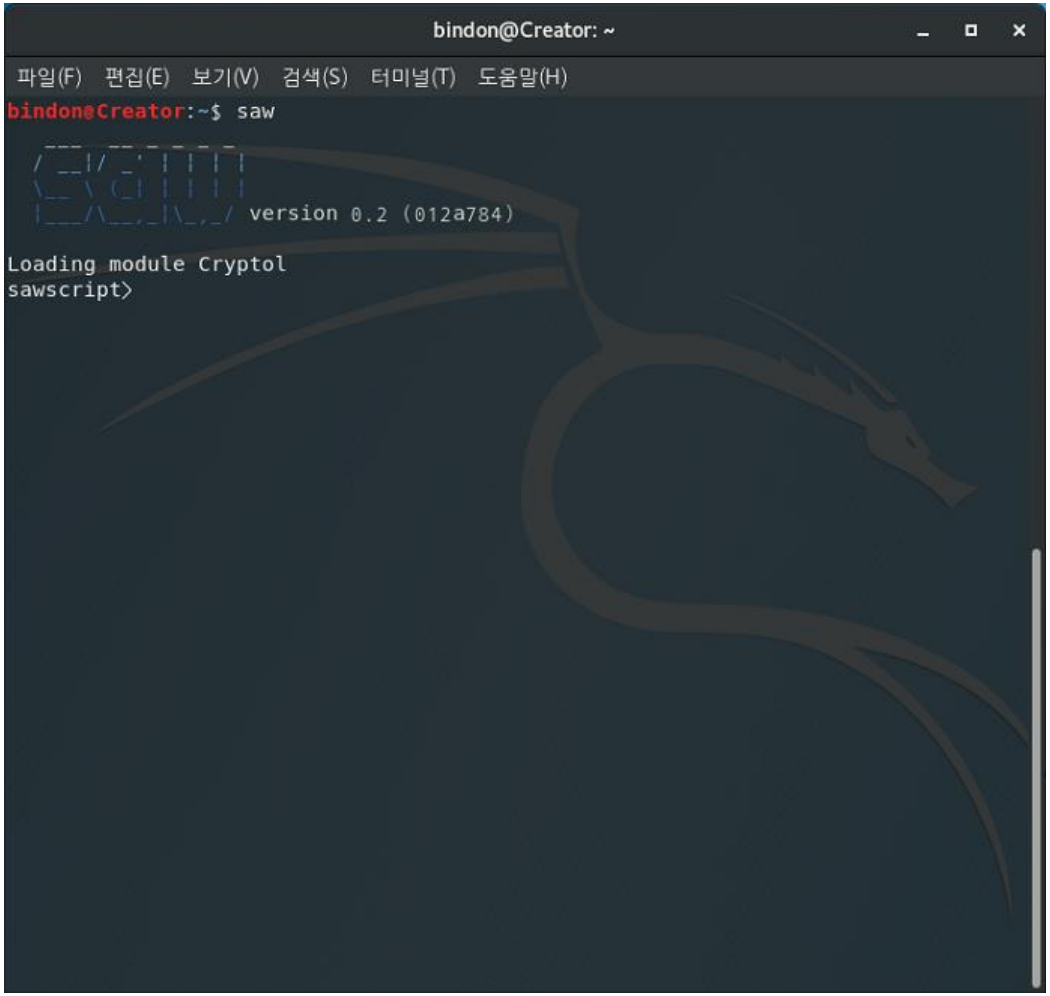




4. Cryptol 환경 구축

KOREA UNIVERSITY

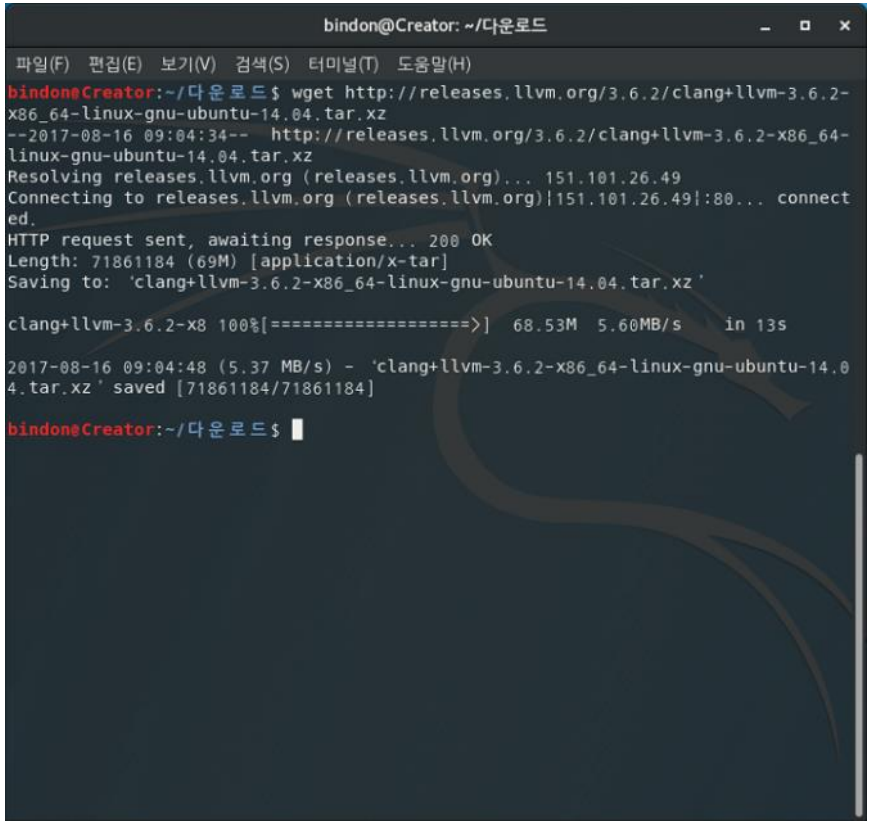
- **SAW 설치**
  - **SAW 실행**



4. CryptoI 환경 구축

KOREA UNIVERSITY

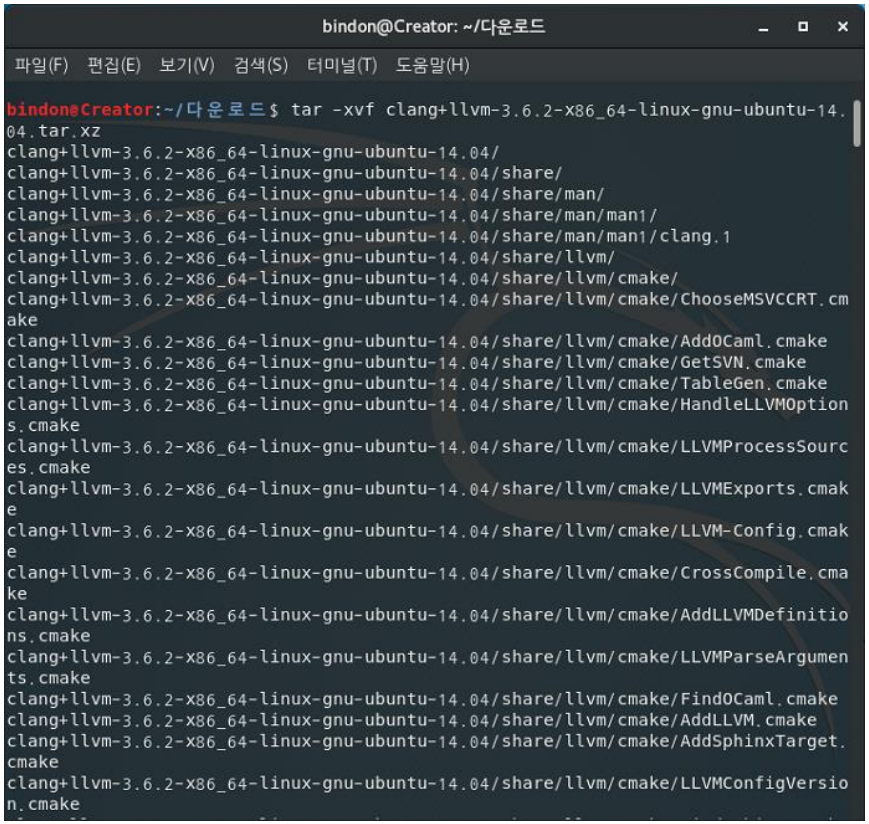
- **Clang 설치**
  - **Clang 다운로드**
    - 최신 버전이 아닌 **clang-llvm-3.6.2-x86_64** 다운로드
    - 최신 버전의 경우 **saw** 연동 시 오류가 발생할 수 있음

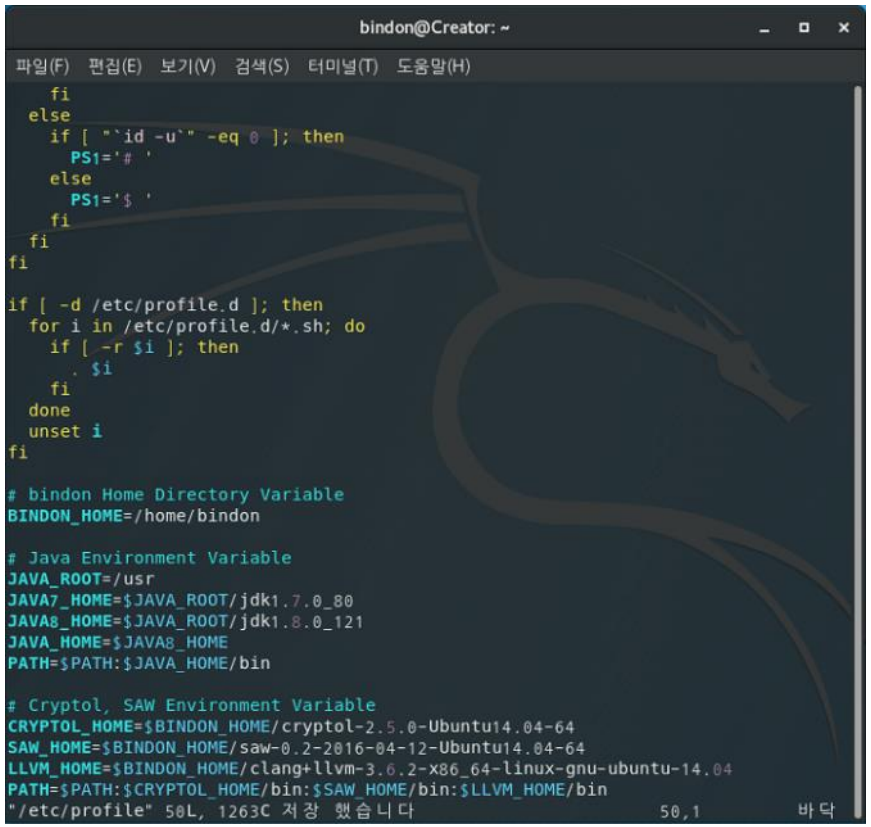


4. Cryptol 환경 구축

KOREA UNIVERSITY

- **Clang 설치**
  - **Clang 압축 해제 및 환경변수 등록**
    - **tar -xvf clang+llvm-3.6.2-x86_64-linux-gnu-ubuntu-14.04.tar.xz**

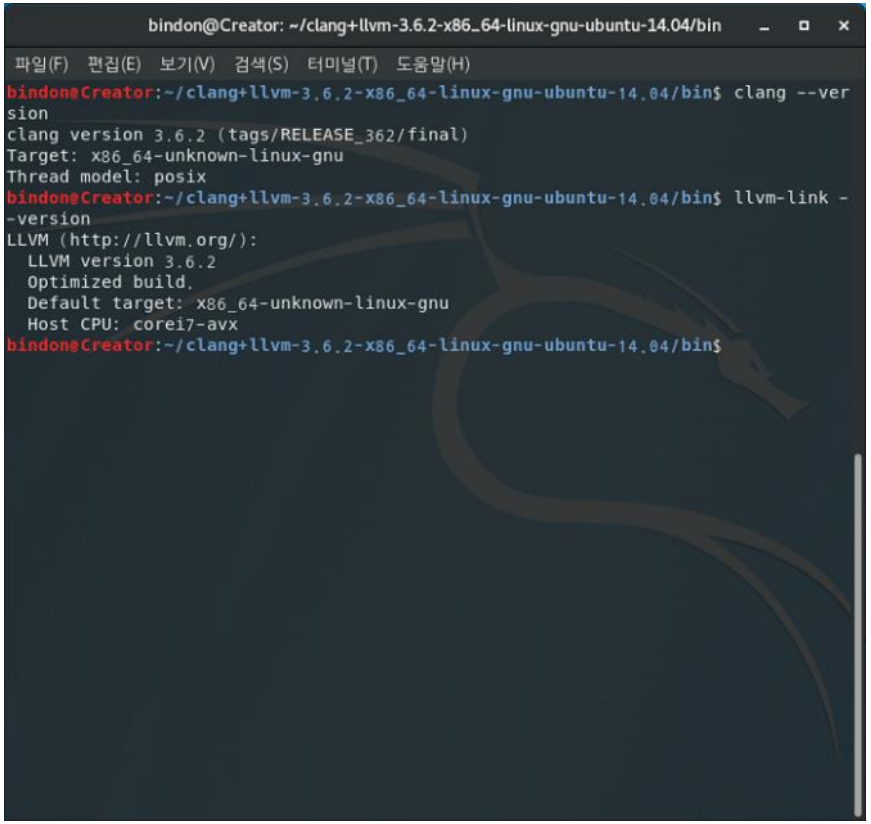




4. CrypTol 환경 구축

KOREA UNIVERSITY

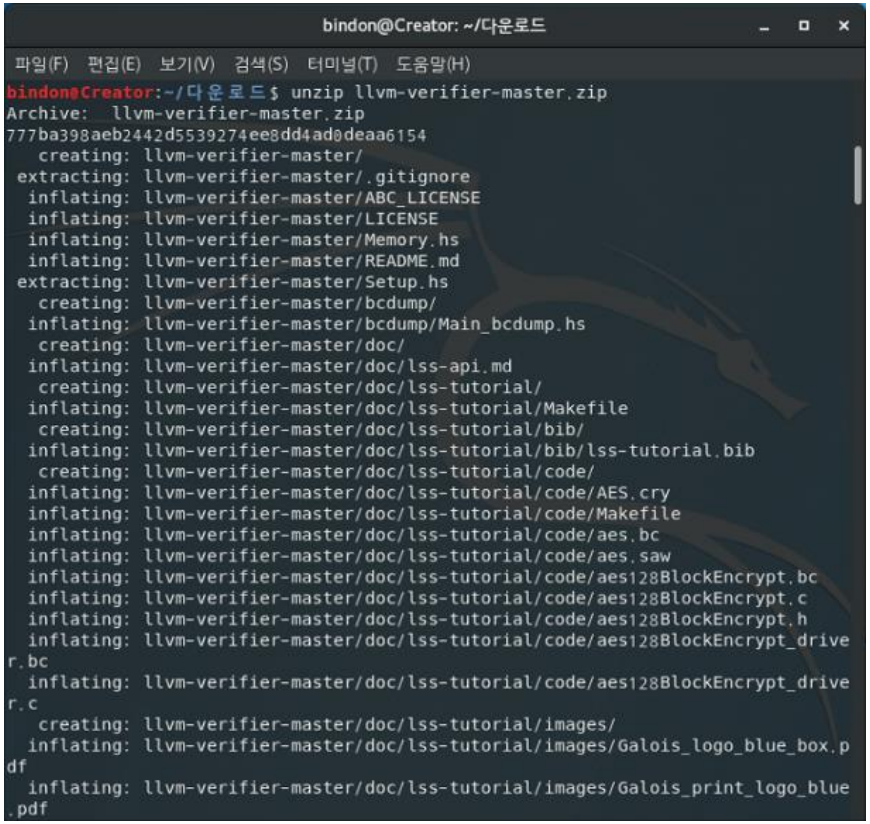
- **Clang 설치**
  - **Clang, LLVM 설치 확인**
    - **clang --version**
    - **llvm-link --version**



4. CrypTol 환경 구축

KOREA UNIVERSITY

- **llvm-verifier 설정**
  - **llvm-verifier 다운로드 및 압축 해제**
    - **c언어로 작성된 모듈을 wrapping하기 위한 library**
    - **https://github.com/GaloisInc/llvm-verifier**



5. ARIA 구현 및 검증

KOREA UNIVERSITY

- 참조 구현 모델 작성(ARIA.cry)
  - ARIA 표준 문서를 바탕으로 참조 구현 모델 작성

bindon@Creator: ~/bindon/Cryptol/aria/KISA-32bit

```
encrypt : ([128], [128]) -> [128]
encrypt (plainText, key) = cipherText
        where
                ek  = generateEncKey(key)
                r1  = oddFunction (plainText, ek @ 0 )
                r2  = evenFunction(r1,        ek @ 1 )
                r3  = oddFunction (r2,        ek @ 2 )
                r4  = evenFunction(r3,        ek @ 3 )
                r5  = oddFunction (r4,        ek @ 4 )
                r6  = evenFunction(r5,        ek @ 5 )
                r7  = oddFunction (r6,        ek @ 6 )
                r8  = evenFunction(r7,        ek @ 7 )
                r9  = oddFunction (r8,        ek @ 8 )
                r10 = evenFunction(r9,        ek @ 9 )
                r11 = oddFunction (r10,       ek @ 10)
                cipherText = finalFunction (r11, ek @ 11, ek @ 12)

decrypt : ([128], [128]) -> [128]
decrypt (plainText, key) = cipherText
        where
                dk  = generateDecKey(key)
                r1  = oddFunction (plainText, dk @ 0 )
                r2  = evenFunction(r1,        dk @ 1 )
                r3  = oddFunction (r2,        dk @ 2 )
                r4  = evenFunction(r3,        dk @ 3 )
                r5  = oddFunction (r4,        dk @ 4 )
                r6  = evenFunction(r5,        dk @ 5 )
                r7  = oddFunction (r6,        dk @ 6 )
                r8  = evenFunction(r7,        dk @ 7 )
                r9  = oddFunction (r8,        dk @ 8 )
                r10 = evenFunction(r9,        dk @ 9 )
                r11 = oddFunction (r10,       dk @ 10)
                cipherText = finalFunction (r11, dk @ 11, dk @ 12)

property ARIACorrect plainText key = decrypt (encrypt (plainText, key), key) == plainText
```

5. ARIA 구현 및 검증

KOREA UNIVERSITY

- 래핑 코드 작성(ARIA_wrap.c)
  - ARIA 암호 모듈을 호출하는 래핑 코드 작성
  - llvm-verifier 라이브러리 import를 통해 변수 할당

bindon@Creator: ~/bindon/Cryptol/aria/KISA-32bit

```
#include "ARIA_core.h"
#include <sym-api.h>

void encrypt(unsigned char *plainText, unsigned char *cipherText, unsigned char *key) {
        unsigned char rk[16*17];
        Crypt(plainText, EncKeySetup(key, rk, 128), rk, cipherText);
}

int main() {
        unsigned char *plainText  = lss_fresh_array_uint8(16, 0, NULL);
        unsigned char *key        = lss_fresh_array_uint8(16, 0, NULL);
        unsigned char *cipherText = malloc(16 * sizeof(unsigned char));

        encrypt(plainText, cipherText, key);

        lss_write_aiger_array_uint8(cipherText, 16, "ARIA_imp.aig");

        return 0;
}
```

5. ARIA 구현 및 검증

KOREA UNIVERSITY

- **SAW 스크립트 작성(ARIA.saw)**
  - **참조 구현 모델과 구현 모델 AIG를 이용하여 동치성 검사를 수행하는 SAW 스크립트 작성**

bindon@Creator: ~/bindon/Cryptol/aria/KISA-32bit

```
import "ARIA.cry";

let {{
  ariaExtract x = encrypt (plainText, key)
    where [plainText, key] = split x
}};

print "Loading ARIA implementation";
aes_imp <- time (load_aig "ARIA_imp.aig");

print "Bitblasting Cryptol implementation";
aes_ref <- time (bitblast {{ ariaExtract }});

print "Checking equivalence (may take about an hour)";
res <- time (cec aes_imp aes_ref);
print res;

print "Writing reference AIG";
time (write_aig "ARIA_ref.aig" {{ ariaExtract }});
```

5. ARIA 구현 및 검증

KOREA UNIVERSITY

- **Makefile 작성(Makefile)**
  - 작성된 코드들을 이용하여 배치 작업을 한 번에 수행할 수 있도록 **Makefile**을 작성
  - 현재 환경에 맞게 수정 필요

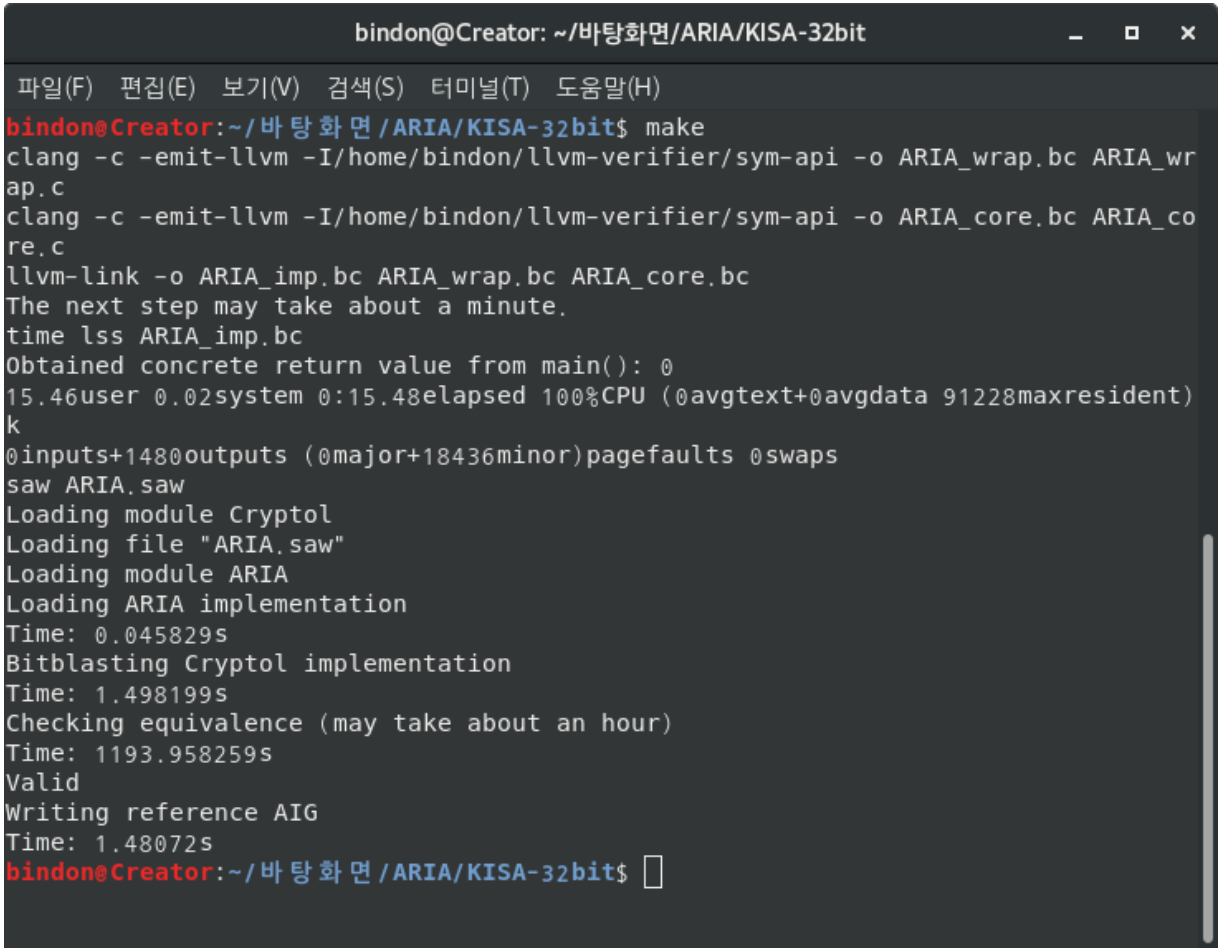
bindon@Creator: ~/bindon/Cryptol/aria/KISA-32bit

```
CLANG?=clang
LLVM_LINK?=llvm-link
SAW?=saw
LSS?=lss
LLVM_VERIFIER=/home/bindon/llvm-verifier/sym-api

all: ARIA_imp.aig

ARIA_all.bc:
	${CLANG} -c -emit-llvm -I${LLVM_VERIFIER} -o ARIA_wrap.bc ARIA_wrap.c
	${CLANG} -c -emit-llvm -I${LLVM_VERIFIER} -o ARIA_core.bc ARIA_core.c

ARIA_imp.bc: ARIA_all.bc
	${LLVM_LINK} -o ARIA_imp.bc ARIA_wrap.bc ARIA_core.bc

ARIA_imp.aig: ARIA_imp.bc
	@echo "The next step may take about a minute."
	time ${LSS} ARIA_imp.bc
	${SAW} ARIA.saw

clean:
	rm -f ARIA_wrap.bc ARIA_imp.aig
~
```

5. ARIA 구현 및 검증

KOREA UNIVERSITY

- 검증 수행(make)
  - Makefile이 존재하는 폴더에서 make 수행



Cryptol을 이용한 국내 표준 블록 암호 ARIA 모듈의 자동 정형 검증

(Automated Formal Verification of Korean Standard Block Cipher ARIA using Cryptol)

Security **A**nalysis a**N**d **E**valuation Lab, CIST

최원빈 (Wonbin Choi), bindon@hanmir.com

Prof. Dr. Seungjoo (Gabriel) Kim