# 形式的な暗号学的安全性証明による アセンブリプログラムの安全性検証:BBSの事例

# Certifying Assembly with Formal Cryptographic Proofs: the Case of BBS

Reynald Affeldt, David Nowak, and Kiyoshi Yamada
アフェルト レナルド, ノヴァック ダヴィッド, 山田 聖

Research Center for Information Security
Advanced Industrial Science and Technology (AIST), Japan

28/09/2009

# 動機

- 暗号ソフトウェアの形式的検証
  - Coq定理証明支援系を利用 [INRIA, 1985~]
- 難しさ: 暗号プリミティブ
  - **機能的正当性**(functional correctness)の証明は技術的な挑戦
    - 実装は低級言語、多くはアセンブリで記述される
  - 仕様は**安全性証明**(security proofs)から構成される
    - 暗号ソフトウェアの仕様の主要な部分

⇒ 目標:
機能的正当性の形式的検証と安全性証明の実用的なフレームワーク

# 成果

- 統合手法:
  - 機能的正当性の形式的検証
  - 安全性証明の形式的検証

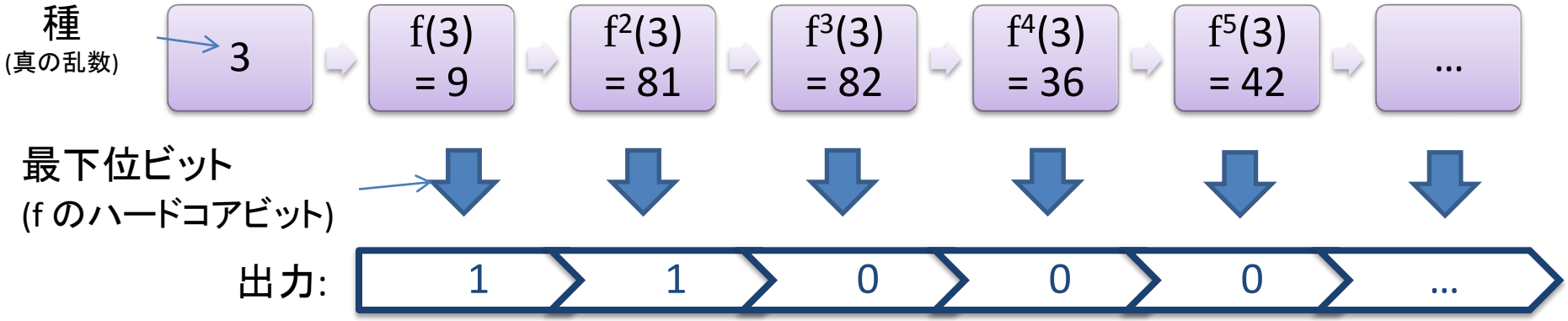
  “*game-playing*” [Shoup, 2004] に基づく
- アセンブリの形式的検証のためのフレームワーク
  - [Saabas and Uustalu, TCS 2007] に基づく
    - goto (labeled jump) を扱うため
  - [Affeldt and Marti, ASIAN 2006] を再利用
- 疑似乱数ビット生成器の形式的検証
  - BBS疑似乱数ビット生成器
    [Blum, Blum, and Shub, SIAM Journal on Computing 1986]
  - 機能的正当性と安全性証明
    - 安全性証明は [Nowak, ICISC 2008] による

# 概要

- BBS疑似乱数ビット生成器
  - 背景
  - アセンブリによる実装
- アセンブリの形式的検証
  - SGoto の形式化
- BBSの形式的検証
  - 機能的正当性
  - 安全性証明との統合

# Blum-Blum-Shub
# 疑似乱数ビット生成器

- $f(x) = x^2 \bmod n$, $n$は“*Blum integer*”
  - $n = p \times q$, $p, q$ は(mod 4)のもとで 3 と合同な素数, 例: $n = 11 \times 19$
  - $f$ はQR$_n$ 上の一方向性関数, QR$_n$ は “mod $n$ の平方剰余” の集合
    - QR$_n$ = { $x \in \boldsymbol{Z}^*_n$ | $\exists\, y \in \boldsymbol{Z}^*_n$ such that $x = y^2 \pmod n$}
- 構成:




- 安全性証明:
  - BBS は “左予測不可能”
    - “平方剰余仮定”に基づく安全性証明
      - Jacobi symbol が1となる$x \in \boldsymbol{Z}^*_n$が QR$_n$ に含まれるかどうかの判定は計算量的に困難

# アセンブリによるBBSの安全性
# アプローチ

- BBS の安全性証明は一般的に関数に対して行われる:
  1. BBS を関数へ形式化:

$$\begin{array}{l}\mathsf{bbs}(len \in \mathbb{N}, seed \in \mathbb{Z}_m^*) =_{def} \mathsf{bbs\_rec}(len, seed^2)\\ \mathsf{bbs\_rec}(len \in \mathbb{N}, x \in QR_m) =_{def} \textbf{match } len \textbf{ with}\\ \quad \mid 0 \Rightarrow []\\ \quad \mid len' + 1 \Rightarrow \mathsf{parity}(x) :: \mathsf{bbs\_rec}(len', x^2)\\ \quad \textbf{end}\end{array}$$

  2. 左予測不可能性を述語へ形式化、これを unpredictable とする
  3. unpredictable(bbs) の成立を証明
- 問題: アセンブリで実装された BBS のふるまいは上記の関数と完全に一致しない
  - 特に、実装上の選択による
- BBS のアセンブリによる実装の安全性の証明方法
  1. アセンブリプログラム bbs_asm から数学的関数を抽出
  2. unpredictable(⟦bbs_asm⟧) の成立を証明
     - unpredictable(bbs) を中間の補助定理として利用

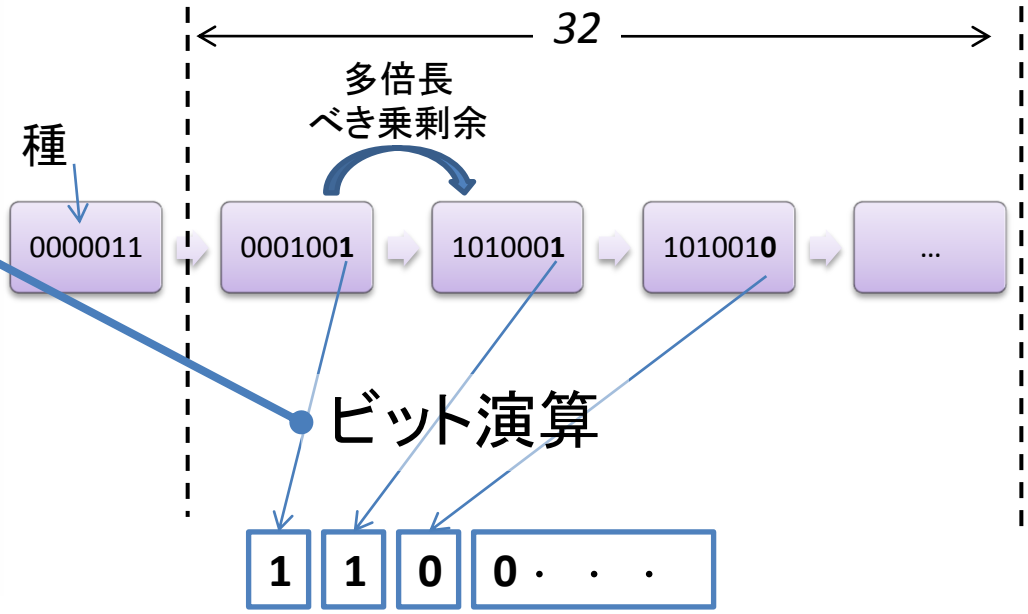
# BBSのアセンブリでの実装

```
bbs_asm =def
0:      addiu i gpr_zero 0_16
1:      addiu L l 0_16
2:      beq i n 240
3:        addiu j gpr_zero 0_16
4:        addiu w gpr_zero 0_16
5:        beq j thirtytwo 236
6:          mul_mod k x x m ...
230:        lw w_ 0_16 x
231:        andi w_ w_ 1_16
232:        sllv w_ w_ j
233:        cmd_or w w w_
234:        addiu j j 1_16
235:        jmp 5
236:      sw w 0_16 L
237:      addiu L L 4_16
238:      addiu i i 1_16
239:      jmp 2
240:
```

疑似乱数ビットの必要なワード数分繰り返す

繰り返しごとに1ワード分の疑似乱数を生成

– 最初の繰り返しの例:



# べき乗剰余のアセンブリでの実装

- 2つの“モンゴメリ乗算”による実装
  - [Montgomery, *Modular multiplication without trial division*, 1985]

```
mont_mul_strict_init =def
6:      multi_zero ext k Z z
13:     mflhxu gpr_zero
14:     mthi gpr_zero
15:     mtlo gpr_zero
16:     montgomery k alpha x y z m one ext int X Y M Z quot C t s
54:     beq C gpr_zero 81
55:       addiu t t 4_16
56:       sw C 0_16 t
57:       addiu ext k 1_16
58:       multisub ext one z m z M int quot C Z X Y X
80:       jmp 118
81:       multi_lt_prg k z m X Y int ext Z M
93:       beq int gpr_zero 96
94:         skip
95:         jmp 118
96:         multisub k one z m z ext int quot C Z X Y X
118:
```

- モンゴメリ乗算:
  - 入力: $k$ ワード整数 $\mathcal{M}$, $\mathcal{X}$, $\mathcal{Y}$
  - 出力: $k$+1ワード整数 $\mathcal{Z}$

    such that

    $\beta^k \mathcal{Z} = \mathcal{X} \cdot \mathcal{Y} \pmod{\mathcal{M}}$  NB: $\beta = 2^{32}$

    and

    $\mathcal{Z} < 2\mathcal{M}$ (“raw” version)

    $\mathcal{Z} < \mathcal{M}$ (“strict” version)

- べき乗剰余:
  - 入力: $k$ワード整数 $\mathcal{A} = \beta^{2k}$, $\mathcal{M}$, $\mathcal{X}$, $\mathcal{Y}$
  - 出力: $\mathcal{Z} = \mathcal{X} \cdot \mathcal{Y} \pmod{\mathcal{M}}$

    1. $\beta^k \mathcal{Z}' = \mathcal{X} \cdot \mathcal{Y} \pmod{\mathcal{M}}$
    2. $\beta^k \mathcal{Z} = \mathcal{Z}' \cdot \mathcal{A} = \beta^k \mathcal{X} \cdot \mathcal{Y} \pmod{\mathcal{M}}$

    注意: $\mathcal{M}$ は奇数

# 概要

- 疑似乱数ビット生成器
  - 背景
  - アセンブリによる実装
- アセンブリの形式的検証
  - SGOTO の形式化
- BBSの形式的検証
  - 機能的正当性
  - 安全性証明との統合

# アセンブリの形式的証明

- 分離論理を利用[Reynolds, LICS 2002]
  - ポインタを扱うホーア論理
  - 機械化 [Affeldt and Marti, ASIAN 2006]:
    - Coq定理証明支援系を利用
    - スマートカードアセンブリを対象
    - 基礎的な算術関数の形式的検証へ適用
  - WHILE プログラムに限定
    - 構造化制御フロー(sequence, if-then-else, while)
- 新規性:
  - SGOTO のCoq上での形式化
    - Gotoを持つ低級言語のための結合可能なホーア論理 [Saabas and Uustalu, TCS 2007]

# 階乗プログラム [Saabas and Uustalu, TCS 2007]

WHILEでは: **while** $x < n$ **do** ($x := x+1$; $s := s*x$)

- WHILEのホーア論理での機能的正当性:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\begin{smallmatrix} x+1 \le n \\ \wedge\ s*(x+1) = (x+1)! \end{smallmatrix}\}\, x := x+1 \,\{\begin{smallmatrix} x \le n \\ \wedge\ s*x = x! \end{smallmatrix}\}} \quad \dfrac{}{\{\begin{smallmatrix} x < n \\ \wedge\ s = x! \end{smallmatrix}\}\, x := x+1 \,\{\begin{smallmatrix} x \le n \\ \wedge\ s*x = x! \end{smallmatrix}\}} \quad \dfrac{}{\{\begin{smallmatrix} x \le n \\ \wedge\ s*x = x! \end{smallmatrix}\}\, s := s*x \,\{\begin{smallmatrix} s = x! \\ \wedge\ x \le n \end{smallmatrix}\}}}{\{x < n \wedge s = x!\}\, x := x+1; s := s*x \,\{x \le n \wedge s = x!\}}}{\{x < n \wedge x \le n \wedge s = x!\}\, x := x+1; s := s*x \,\{x \le n \wedge s = x!\}}}{\{x \le n \wedge s = x!\}\, S \,\{x \not< n \wedge x \le n \wedge s = x!\}}}{\{n \ge 0 \wedge x = 0 \wedge s = 1\}\, S \,\{x = n \wedge s = n!\}}$$

# 階乗プログラム [Saabas and Uustalu, TCS 2007]

SGOTOでは : (*1*, **ifnot** $x < n$ **goto** *5*) ⊕(((*2*, $x := x + 1$)⊕(*3*, $s := s*x$))⊕(*4*, **goto** *1*))

- SGOTO のホーア論理での機能的正当性:

$$I_1 =_{\text{df}} pc = 1 \wedge n \leq 0 \wedge x = 0 \wedge s = 1$$
$$I_{1'} =_{\text{df}} pc = 1 \wedge x \leq n \wedge s = x!$$
$$I_2 =_{\text{df}} pc = 2 \wedge x < n \wedge x \leq n \wedge s = x!$$
$$I_{2'} =_{\text{df}} pc = 2 \wedge x <$$
$$I_{2''} =_{\text{df}} pc = 2 \wedge x +$$ $$+ 1)!$$
$$I_3 =_{\text{df}} pc = 3 \wedge x \leq$$

$n \leq 0$ ?!

$$I_4 =_{\text{df}} pc = 4 \wedge x \leq n \wedge s = x!$$
$$I_5 =_{\text{df}} pc = 5 \wedge x \not\leq n \wedge x \leq n \wedge s = x!$$
$$I_{5'} =_{\text{df}} pc = 5 \wedge x = n \wedge s = x!$$

$$J_{1'} =_{\text{df}} (pc = 1 \wedge ((x < n \wedge I_{25}[pc \mapsto 2]) \vee (x \not< n \wedge (I_{25}[pc \mapsto 5] \vee 5 = 1))) \vee (pc \neq 1 \wedge I_{25})$$
$$J_{2''} =_{\text{df}} (pc = 2 \wedge I_3[(pc, x) \mapsto (2, x + 1)]) \vee (pc \neq 2 \wedge I_3)$$
$$J_3 =_{\text{df}} (pc = 3 \wedge I_4[(pc, s) \mapsto (3, s * x)]) \vee (pc \neq 3 \wedge I_4)$$
$$J_4 =_{\text{df}} (pc = 4 \wedge (I_{1'}[pc \mapsto 4] \vee 1 = 4)) \vee (pc \neq 4 \wedge I_{1'})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{J_{2''}\}\,2\,\{I_3\}}{\{I_{2''}\}\,2\,\{I_3\}}}{\{I_{2'}\}\,2\,\{I_3\}}}{\{pc = 2 \wedge I_{2'34}\}\,2\,\{I_{2'34}\}} \quad \dfrac{\dfrac{\{J_3\}\,3\,\{I_4\}}{\{I_3\}\,3\,\{I_4\}}}{\{pc = 3 \wedge I_{2'34}\}\,3\,\{I_{2'34}\}}}{\{I_{2'34}\}\,2 \oplus 3\,\{pc \notin [2,4) \wedge I_{2'34}\}}}{\{I_2'\}\,2 \oplus 3\,\{I_4\}}}{\{I_2\}\,2 \oplus 3\,\{I_4\}}}{\{pc \in [2,4) \wedge I_{1'24}\}\,2 \oplus 3\,\{I_{1'24}\}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\{J_{1'}\}\,1\,\{I_{25}\}}{\{pc = 1 \wedge I_{1'25}\}\,1\,\{I_{1'25}\}}}{\{I_{1'25}\}\,C\,\{pc \notin [1,5) \wedge I_{1'25}\}}}{\{I_{1'}\}\,C\,\{I_5\}}}{\{I_1\}\,C\,\{I_{5'}\}}$$

$$\dfrac{\dfrac{\{J_4\}\,4\,\{I_{1'}\}}{\{pc = 4 \wedge I_{1'24}\}\,4\,\{I_{1'24}\}}}{\{I_{1'24}\}\,(2 \oplus 3) \oplus 4\,\{pc \notin [2,5) \wedge I_{1'24}\}}}{\{pc \in [2,5) \wedge I_{1'25}\}\,(2 \oplus 3) \oplus 4\,\{I_{1'25}\}}$$

⇒ SGOTO の導出木は対話的証明に適していない

# WHILE から SGOTO への変換

## コンパイル

**c** =
**while** $b$ ( **body**)

compile $l_0$

**c′** =

| | |
|---|---|
| $l_0$: | **cjmp** $(\neg b)$ $l_f$+1 |
| $l_0$+1: | **body′** |
| $l_0$+1+card(**body′**): | **jmp** $l_0$ |
| $l_0$+card(**prg′**): | |

## 性質 [Saabas and Uustalu, TCS 2007]

### 操作的意味を保存

Some $s$ —— **c** → Some $s'$ ⇒ Some $(l_0, s) \succ\!\!-$ **c′** → Some ($l_0$ +card(dom(**c′**)), $s'$)

### ホーア論理の導出木を保存

$\{\mathcal{P}\}$ **c** $\{Q\}$ ⇒ [ $pc = l_0 \wedge \mathcal{P}$ ] **c′** [$pc = l_0$ +card(dom(**c′**)) $\wedge Q$]

# 概要

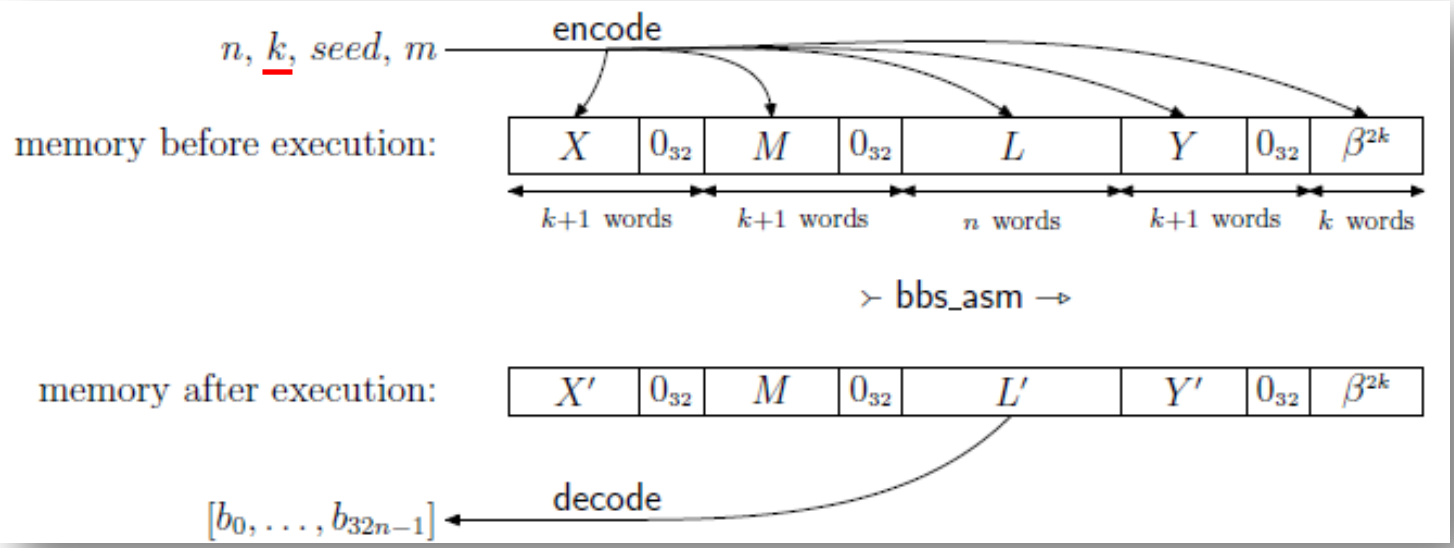
- 疑似乱数ビット生成器
  - 背景
  - アセンブリによる実装
- アセンブリの形式的検証
  - SGOTO の形式化
- BBSの形式的検証
  - 機能的正当性
  - 安全性証明との統合

# BBS の仕様

- ポイント: ギャップを埋める
  - BBS の暗号学的証明 [Nowak, ICISC 2008]
    - 整数のサイズに制限なし
  - BBS のアセンブリによる実装
    - 実装上の選択による制約

⇒ encode/decode関数を利用

  - encode: 任意長整数から多倍長整数へ
  - decode: メモリーからビット列へ



# BBS の機能的正当性

- 証明対象, SGOTOによるホーアの三つ組:

$$\left[\ \lambda pc.\ \lambda s.\ pc = 0 \wedge \text{encode}\ (n, k, seed, m) = s\ \right]$$

bbs_asm

$$\left[\ \lambda pc.\ \lambda s.\ pc = 240 \wedge \text{decode}(s) = \text{bbs}(32 \times n, seed, m)\ \right]$$

BBS のアセンブリによる実装

BBS関数 [Nowak, ICISC 2008]

- 実際の証明:

$4(4k + n + 2) < 2^{32} \rightarrow$ ← 実装上の選択による制約

$$\left\{\ \lambda s.\ \text{encode}\ (n, k, seed, m) = s\ \right\}$$

bbs_asm_WHILE

$$\left\{\ \lambda s.\ \text{decode}(s) = \text{bbs}(32 \times n, seed, m)\ \right\}$$

  - コンパイル操作により証明対象のSGOTOによるホーアの三つ組へ変換

# 機能的正当性と安全性証明
## 統合

- 機能的正当性から⟦bbs_asm⟧へ:
  1. 停止性と決定性の証明:

     $4(4k + n + 2) < 2^{32} \rightarrow \exists!\, s'$ . Some (0, encode ($n$, $k$, $seed$, $m$)) $\succ\!\!\!-$ bbs_asm $\rightarrow s'$

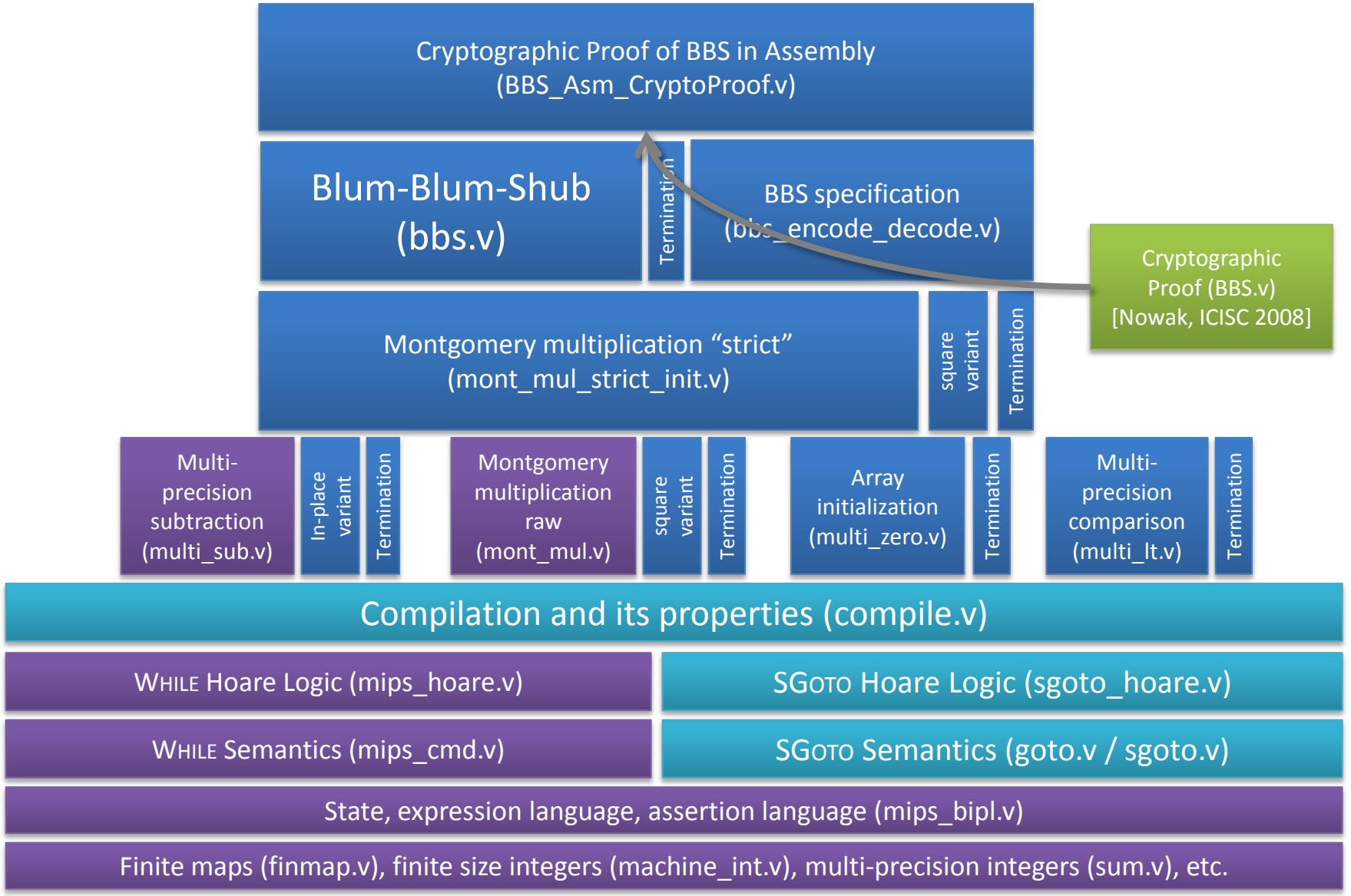
  2. 関数 $\mathrm{exec}_{\mathrm{bbs\_asm}}$ を抽出 :

     ⟦bbs_asm⟧ = $\mathrm{prefix}_{len+1}$ (decode ($\mathrm{exec}_{\mathrm{bbs\_asm}}$ ($\lceil (len+1)/32 \rceil, \lceil \log_{2^{32}}(m) \rceil, seed, m$)))

- ⟦bbs_asm⟧ は関数bbsとしてふるまう
  - ホーアの三つ組による
  - 実装上の選択から引き継がれた制約に従う

  ⇒ unpredictable(bbs) の証明を再利用し、
  unpredictable(⟦bbs_asm ⟧) を証明できる. (Q.E.D.)

# 形式化の概要


Cryptographic Proof of BBS in Assembly (BBS_Asm_CryptoProof.v)
Blum-Blum-Shub (bbs.v)
Termination
BBS specification (bbs_encode_decode.v)
Cryptographic Proof (BBS.v) [Nowak, ICISC 2008]
Montgomery multiplication "strict" (mont_mul_strict_init.v)
square variant
Termination
Multi-precision subtraction (multi_sub.v)
In-place variant
Termination
Montgomery multiplication raw (mont_mul.v)
square variant
Termination
Array initialization (multi_zero.v)
Termination
Multi-precision comparison (multi_lt.v)
Termination
Compilation and its properties (compile.v)
WHILE Hoare Logic (mips_hoare.v)
SGOTO Hoare Logic (sgoto_hoare.v)
WHILE Semantics (mips_cmd.v)
SGOTO Semantics (goto.v / sgoto.v)
State, expression language, assertion language (mips_bipl.v)
Finite maps (finmap.v), finite size integers (machine_int.v), multi-precision integers (sum.v), etc.

# 結論

- まとめ:
  - 暗号プリミティブの形式的検証の実用的アプローチ
    - アセンブリコードの検証フレームワーク
    - 安全性証明の検証フレームワークとの統合
  - 具体的な事例研究
    - BBS疑似乱数ビット生成器のSmartMIPSでの実装に対する形式的検証
  - 形式的検証結果:
    `http://staff.aist.go.jp/reynald.affeldt/bbs`
- 今後の予定:
  - ElGamalの実装の形式的検証
    - 広く使われている公開鍵暗号スキーム
    - いくつかの部分は準備済み
      - 疑似乱数生成器, べき剰余演算等
      - 安全性証明 [Nowak, ICICS 2007]