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

アフェルト レナルド
Reynald Affeldt *

ノヴァック ダヴィッド
David Nowak

Kiyoshi Yamada
山田 聖

産業技術総合研究所 情報セキュリティ研究センター
National Institute of Advanced Industrial Science and Technology

キーワード：形式的検証, 疑似乱数生成器, ゲーム手法, アセンブリプログラム
Keywords: Formal verification, Pseudorandom number generator, Game-playing, Assembly code

はじめに 機密情報を扱う組込みシステムの普及に伴い，低レベルプログラムの安全性の保証に対する重要性が高まっている．しかし暗号研究者らによる安全性証明の対象はアルゴリズムであり，実際に動作するプログラムではない．そこで我々は，アセンブリ言語で実装された暗号プリミティブの安全性を保証するための，暗号学的安全性証明の拡張方法を提案する．我々のアプローチは，証明可能安全性のゲーム列による証明に，アセンブリプログラムの正しさの証明を統合した，定理証明支援系 Coq 上に構成されたフレームワークに基づく [1]．このアプローチの有効性を示すために，Blum-Blum-Shub 疑似乱数生成器 (以後，BBS) の，スマートカード用の MIPS アーキテクチャ(以後，SmartMIPS) による実装の，暗号学的安全性の証明を行った．

1 BBS 疑似乱数生成器

BBS は，平方剰余の性質を利用した疑似乱数生成器である．この性質は，2 つの異なる奇素数の積 m を法とする乗法群 $(\mathbb{Z}_m^*)$ の要素が与えられた場合に，それが m の平方剰余であるかどうかの判定は困難であるというものであり，この困難性は暗号研究者らにより一般に信じられている．BBS が暗号学的安全であるとは，BBS の出力する疑似乱数列と真の乱数列を区別できないことを意味する．BBS は，平方剰余問題の困難性の仮定に基づき，(左) 予測不可能性を満たす．BBS は，平方剰余演算を繰り返し，パリティ検査の結果のリストを出力する:

$$\mathsf{bbs\_rec}(l \in \mathbb{N}, x \in QR_m) \stackrel{def}{=} \textbf{if } l > 0 \textbf{ then } \mathsf{parity}(x) :: \mathsf{bbs\_rec}(l-1, x^2) \textbf{ else } []$$

$$\mathsf{bbs}(l \in \mathbb{N}, seed \in \mathbb{Z}_m^*) \stackrel{def}{=} \mathsf{bbs\_rec}(l, seed^2)$$

l は出力させる疑似乱数のビット数，QR_m は法 m の下での平方剰余の集合である．BBS の SmartMIPS アセンブリでの実装を以下に示す:

```
bbs_asm ≝
  0:   addiu i gpr_zero 0x10
  1:   addiu L l 0x10
  2:   beq i n 240
  3:     addiu j gpr_zero 0x10
  4:     addiu w gpr_zero 0x10
  5:     beq j thirtytwo 236
  6:       mul_mod k x x m ...
230:       lw w_ 0x10 x
231:       andi w_ w_ 0x1
232:       sllv w_ w_ j
233:       cmd_or w w w_
234:       addiu j j 0x1
235:       jmp 5
236:     sw w 0x10 L
237:     addiu L L 0x4
238:     addiu i i 0x1
239:     jmp 2
```

mul_mod はモンゴメリ乗算法により平方剰余計算を行う部分である (詳細は [1] を参照).

問題点 このアセンブリでの実装が BBS 関数と同じ動作をすることは自明ではなく，更に，BBS 関数が暗号学的に安全であっても，アセンブリによる実装が同じ性質を持つことは明らかではない．

2 アセンブリの安全性証明

アセンブリプログラムの実装に対する暗号学的安全性の証明を行えるようにするために，ゲーム列による暗号証明法を拡張し，アセンブリプログラムの実装を直接的に扱えるようにした．

ゲーム列による証明手法では安全性の性質を攻撃者によって解かれるゲームとしてモデル化し，いかなる攻撃者もランダムなプレーヤに比べ優位性がほとんど無いことを示す．暗号証明は，モデル化されたゲームを安全性の根拠となる計算論的仮定へ帰着させるという形をとる．

関数 f の予測不可能性に対するゲーム $\mathsf{unpredictability}(f)$ を次のように定義する: $seed$

*email: reynald.affeldt at aist.go.jp

を乱数の種の集合から適当に選ぶ; f はビット列 $[b_0, \ldots, b_l]$ を計算; 最初のビット b_0 を除いた残りのビット列は攻撃者 A へ渡される; A は最初のビットの予測値 $\widehat{b_0}$ を返す．A による予想の正しさをゲームの結果とする．

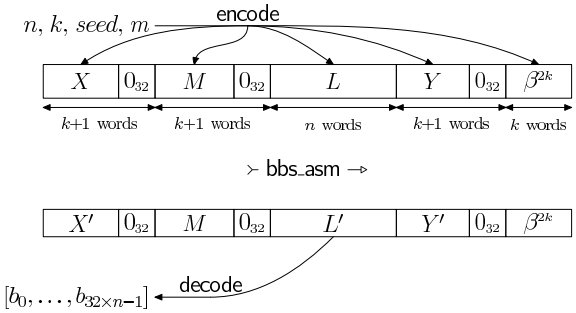
アセンブリコードの予測不可能性は，数学的関数の形で取り出したアセンブリコードの意味を，予測不可能性の定義に組み込む形となる:

$$\mathsf{unpredictability\_asm}(c) \stackrel{def}{=} \mathsf{unpredictability}([\![c]\!])$$

アセンブリコード c の意味を表す関数 $[\![c]\!]$ が well-defined であるために，コード c の停止性，決定性に加え (4 節)，アセンブリコードが期待通りに動作するための条件 (3 節) を示す必要がある．

3 アセンブリの正しさの証明

アルゴリズム上の値とプログラムの扱うデータとの対応をとるために，2 つの関数 encode, decode を利用する．encode は，出力させる疑似乱数のワード数 (n)，乱数の種やモジュラスをエンコードするための領域のワード数 (k)，疑似乱数の種，モジュラスから，この実装に固有の領域 (Y, β^{2k}) を含むメモリ上の領域を用意する．decode は，実行後のメモリ状態から疑似乱数ビット列を取り出す．これらの関数は，以下のメモリ配置に従う:



ここで，$4(4k+n+2) < 2^{32}$ である限り，n, k は非常に大きい値となっても良く，k は平方剰余問題が実際に手に負えないと信じられている長さを事実上カバーしている．

上記の関数を利用すると，bbs_asm の正しさはホーア論理に従い以下のように表すことができる:

$$\frac{4(4k+n+2) < 2^{32} \rightarrow}{\left[\begin{array}{c} pc = 0 \wedge \mathsf{encode}\,(n,k,seed,m) = s \end{array}\right]}$$
$$\mathsf{bbs\_asm} \qquad (\P)$$
$$\left[\begin{array}{c} pc = 240 \wedge \\ \mathsf{decode}\,(s) = \mathsf{bbs\_fun}(32\times n, seed, m) \end{array}\right]$$

これは，分岐を含むコードからループを用いたコードへの逆コンパイル手法により，一般的なホーア論理で証明できる (詳細は [1] を参照).

4 アセンブリから数学的関数へ

ここでは，ホーアトリプル (¶) を用いて，ゲームによる証明で必要となる bbs_asm の意味を数学的関数として定義する (詳細は [1] を参照).

まず，bbs_asm の停止性と決定性を証明する: 全ての n, k, $seed$, m に対し，唯一の状態 σ が存在し，$\mathsf{Some}(0, \mathsf{encode}\,(n,k,seed,m)) \succ \mathsf{bbs\_asm} \twoheadrightarrow \sigma$，すなわち，入力が適切にエンコードされたメモリ状態で，プログラム bbs_asm を 0 行目から実行すると，状態 s' で停止する．

ホーア論理の健全性とトリプル (¶) から以下を得る．

Lemma (*correctness*): $4(4k+n+2) < 2^{32}$ かつ $\mathsf{Some}(0, \mathsf{encode}\,(n,k,seed,m)) \succ \mathsf{bbs\_asm} \twoheadrightarrow \mathsf{Some}(l', s')$ ならば $l' = 240$ かつ $\mathsf{decode}\,(s') = \mathsf{bbs\_fun}(32 \times n, seed, m)$.

停止性の証明は 2 つのステップから成る．まず，唯一の最終状態の存在を，エラー状態を考慮せず示す:

Lemma (*execution*): $4(4k+n+2) < 2^{32}$ の場合，σ が存在し，$\mathsf{Some}(0, \mathsf{encode}\,(n,k,seed,m)) \succ \mathsf{bbs\_asm} \twoheadrightarrow \sigma$.

これは，bbs_asm の外，内それぞれのループに対する帰納法で証明できる．更に，ホーアトリプル (¶) から，最終状態はエラー状態にならない事を得る．

次に，上記の 2 つのレンマから，全ての n, k, $seed$, m, σ に対し，$\mathsf{Some}(0, \mathsf{encode}\,(n,k,seed,m)) \succ \mathsf{bbs\_asm} \twoheadrightarrow \sigma$ となるような関数 $\mathsf{exec}_{\mathsf{bbs\_asm}}$ を得る．この関数を利用し，bbs_asm の意味は数学的関数として表記できる:

$$[\![\mathsf{bbs\_asm}]\!] \stackrel{def}{=} \mathsf{prefix}_{l+1}\big(\mathsf{decode}\big(\mathsf{exec}_{\mathsf{bbs\_asm}}\left(\left\lceil\tfrac{l+1}{32}\right\rceil, \lceil\log_{2^{32}}(m)\rceil, seed, m\right)\big)\big)$$

bbs_asm は常に疑似乱数ビット列の 32 倍の値を返す実装となっているため，出力される値のプレフィクスを取り出す必要がある．$l+1$ が予測不可能ゲームに必要なビット列長である．

参考文献

[1] Affeldt, R., Nowak, D., Yamada, K. *Certifying Assembly with Formal Cryptographic Proofs: the Case of BBS.* 9th Int. Work. on Automated Verification of Critical Systems (AVoCS 2009).