対話型安全性証明つきプログラム配信方式における証明の秘匿とその応用

塚田恭章†

悪性プログラムから計算機を守るためプログラム本体とそれが正しく安全に動作することの数学的証明を同時に流通させる方式（Proof-Carrying Code）が近年提案され，電子署名に基づく現行方式より高い安全性を保証する新技術として期待が集まっている．この方式の課題であった証明サイズの爆発の問題も，対話的・確率的手法に基づく効率的な証明検証技法を応用した対話型安全性証明つきプログラム配信方式の適用により対処できる．しかし，証明をそのまま流通させることは証明の盗用や悪用（たとえば証明を利用したリバースエンジニアリングなど）を招く恐れがある．この問題を解決するため，本論文において，安全性証明の存在をゼロ知識対話型プロトコルを利用して保証する方式を提案する．本方式により，証明の盗用・悪用を防ぐことができる．特に，証明の所有者が正規のプログラム開発者に限られるため，証明の所有をプログラム著作権の保持と見なすことも可能となる．すなわち，提案方式は，プログラムの安全性を効率的に検証したいという利用者側の要求とプログラムの著作権を保護したいという開発者側の要求を同時に満たすことができる．

Proof Hiding in Interactive Proof-carrying Code and Its Applications

Yasuyuki Tsukada†

Proof-carrying code (PCC) is a promising new mechanism that can protect computers from unreliable and possibly malicious foreign programs transmitted by untrusted hosts. One problem with PCC is that safety proofs carried by codes are inherently complex and often larger than the codes themselves and hence proof checking would be an intractable task for each code consumer. This problem was solved by extending the simple certification mechanism of PCC to make it interactive and probabilistic. Another problem is that safety proofs are open and can be used by stealth. To solve this problem, the present paper proposes the use of zero-knowledge protocols, with which proofs in the interactive and probabilistic extension of PCC can be hidden. The proposed mechanism is shown to have a potential application to the proofs-as-copyrights interpretation. In other words, the proposed mechanism has an advantage in that it not only efficiently ensures "safety" (which is requested from the code consumer's side) but also guarantees "copyright" (which is important for the code producer's side) at the same time.

1. はじめに

今日，プログラム化された高機能コンテンツがWWW ブラウザや携帯端末に日常的にダウンロードされ実行されている．また，パケットやノードなどのネットワーク構成要素をプログラマブルとすることで拡張性や耐故障性を増したアクティブネットワーク[1]の研究も進んでいる．プログラムが流通する時代を迎え，悪性プログラムから計算機を守る新しい技術への需要が高まってきている[2]．

現在，公開鍵暗号を利用した電子署名つきプログラム配信方式が広く実用化されている．この方式では，開発者が電子的な「印鑑」をプログラムに捺印し，それを受け取った利用者がその「印鑑」を検証することで，プログラムが信頼できる開発者から送られてきたことを確認できるようになっている．ところが，開発者が信頼できたからといってプログラムも信頼できるとは限らない．信頼できる開発者が作ったとしてもプログラムには誤りが混入しやすいからである．プログラムそのものの安全性を利用者が直接検証することはできないのであろうか？

ウィルスと呼ばれる悪性プログラムの一種に対して

† 日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories, NTT Corporation

は，パターンマッチによる検知・駆除システムが広く実用化されている．しかし，最新のウィルス定義ファイルを用いたとしても，次々に現れる新種のウィルスの感染を未然に防ぐことはできない．未知のウィルスから計算機を守るためには，プログラムが安全に実行可能であるための十分条件をあらかじめ厳密に定義し，その条件を満たしたプログラムのみを実行するようにしなければならない．このような条件を厳密に定義することはできるのであろうか？

悪性プログラムから計算機を守る技術として，仮想機械[3]のバイトコード検証系が普及している．この技術はプログラム（バイトコード）そのものの安全性を検証するものの，特定の検証体系に基づく限定的な安全性（各命令のオペランドが適切な型を持つことなど）しか対象にできない．にもかかわらず，バイトコード検証系を含む仮想機械全体の規模は小さくなく，それ自体の安全性を前もって確認することは容易でない．さらに，仮想機械においては一般にプログラム実行時の検証も行われるため，非力な計算環境では実行速度の低下が問題となる．プログラムの多様な安全性を比較的小さなシステムで検証し，安全性を保証しながら高速に実行することはできないのであろうか？

これらを可能にする新技術として，安全性証明つきプログラム配信方式[4),5)]が近年提案され期待を集めている．この方式では，プログラムが正しく安全に動作すること（たとえば，与えられた入力条件のもとでプログラムを実行した場合，結果が出力条件を満たし，実行時には禁止されたメモリ領域へのアクセスがまったくないこと）の数学的証明をプログラムに添付し，プログラムと証明を同時に流通させる．利用者は受け取った証明を証明チェッカと呼ばれるソフトウェアを用いて機械的に検査することで，プログラムの安全性を自動的に検証することができる．

安全性証明つきプログラム配信方式は，従来技術と比べて次の長所を持っている．

(1) 誤りを含むプログラムや安全でないプログラムに対しては正しい証明が得られないので，証明を検証するこの方式はプログラムそのものを検証する方式である．いい換えると正しい証明の存在がプログラムの安全性の十分条件になっている．

(2) 証明の記述には一階述語論理[5)]，高階論理[6)]，あるいは時相論理[7)]といった汎用の論理体系を用いるので，多様な安全性（型安全性，メモリ安全性，リソース範囲に関するその他の安全性，停止性など）を議論できる．

(3) （証明の作成が困難だとしても）証明の正しさをチェックすること自体は（証明の各ステップで適切な推論規則が適用されていることを逐一確認するだけなので）単純な作業であり，利用者が用いる証明チェッカはコンパクトである．

(4) 安全性の検証はプログラムの実行前に行うため，実行時検証をともなう他方式に比べプログラムを高速に実行できる．

むろん，数学における証明と同様に，安全性証明の作成は一般には容易ではなく，また必ずしも自動化できるとは限らないという問題がある．しかし，証明の作成は証明の検証と異なりオフラインで時間をかけて作業することが許される．さらに，簡単な安全性に対しては，証明を自動生成するコンパイラがいくつか設計されている[8)~11)]．このような特徴をあわせ持つ安全性証明つきプログラム配信方式は，中規模サイズ以下のプログラムの比較的簡単な安全性に対してはきわめて有効であることが実験により示されている[12),13)]．

しかし，安全性証明つきプログラム配信方式には，証明を同時に流通させることに起因する，以下のような問題がある．

第 1 に，安全性証明はプログラム本体の指数オーダのサイズになりうるという問題がある．すなわち，プログラムと証明をそのまま組にして流通させる上記方式を大規模なプログラムに適用した場合，巨大な証明のチェックを利用者に強いることになってしまう．いい換えると，利用者が証明確認に費やせる時間を現実的な範囲に制限した場合，短い証明を有する限られた種類のプログラムしかこの方式では安全に転送・実行することができない．証明サイズの爆発の問題は，利用者がより複雑で高度な安全性を要求した場合，さらに深刻となる．

この証明サイズの爆発の問題は，対話的・確率的手法に基づく効率的な証明検証技法[14)~17)]を応用した対話型安全性証明つきプログラム配信方式の適用により対処できる．文献 18) では共有リソースに対する相互排除プロトコルの検証，文献 19), 20) では機械語プログラムのメモリ安全性の検証にこの方式が適用された．

第 2 に，悪性プログラムから計算機を守るために安全性の証拠として添付する証明が，逆に悪意を持った利用者に盗用されかねないという問題がある．証明をそのまま流通させた場合，たとえば証明を利用したリバースエンジニアリングなどを招く恐れを否定できない．

本論文はこの第 2 の問題を取り上げる．安全性証明の内容を秘匿しつつプログラムの安全性を利用者に

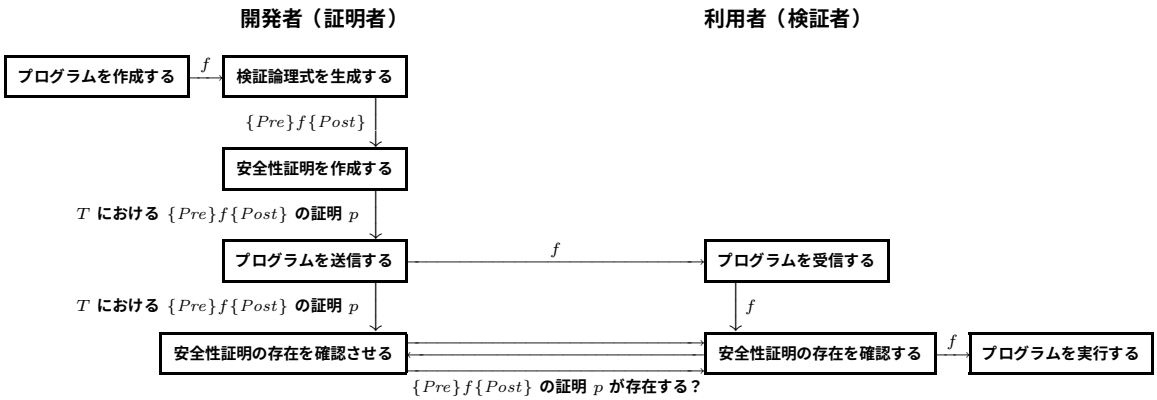

図 1 対話型安全性証明つきプログラム配信方式の概要
Fig. 1 Abstract framework for interactive proof-carrying code.

納得させるため，本論文において，対話型安全性証明つきプログラム配信方式の証明・検証プロトコルをゼロ知識化する[21)~23)]．本提案方式により，安全性証明の盗用・悪用を防ぐことができる．特に，安全性証明の所有者が正規のプログラム開発者に限られる（より正確には，正規のプログラム開発者以外の者が証明を所有することは計算論的に困難になる）ため，証明の所有をプログラム著作権の保持と見なすことも可能となる．

本論文の構成は次のとおりである．2 章では対話型安全性証明つきプログラム配信方式の概要を述べる．3 章では，安全性証明つきプログラム配信方式の提案者による事例研究[5)] の枠組みを用いて，機械語プログラムのメモリ安全性の検証が co-NP 完全問題であることを説明する．4 章では，この安全性が対話的・確率的拡張方式により効率的に検証できることを，文献 19)，20) の結果に基づきプロトコルの具体例とともに示す．5 章は本論文の主要結果であり，対話型安全性証明つきプログラム配信方式のゼロ知識化を示す．特に，4 章で例示したプロトコルが通信複雑度の増加を抑えてゼロ知識化できることを示す．6 章ではゼロ知識対話型安全性証明つきプログラム配信方式のプログラム著作権保護への応用について述べる．7 章でまとめと今後の課題を述べる．

2. 対話型安全性証明つきプログラム配信方式

図 1 に対話型安全性証明つきプログラム配信方式[19),20)] の概要を示す．開発者はプログラム f が安全であることを示すために，検証論理式 $\{Pre\}f\{Post\}$ を生成する．ここで，前条件 Pre および後条件 $Post$ はそれぞれプログラムを実行する直前および直後に成立すべき条件であって，プログラムの利用者が開発者に事前に公開しているものである（たとえば，自然数の列の総和を求めるプログラムを利用したい場合，「入力レジスタは自然数のリスト型を持つ」という前条件と，「出力レジスタは自然数の型を持つ」という後条件を公開しておく）．さらに，プログラムを実行するにあたりいかなる安全性を要求するかについても利用者は事前に公開しているものとする．さて，f と Pre および $Post$ から $\{Pre\}f\{Post\}$ を生成する手法は，利用者が要求する安全性に依存して決まるが，いずれにせよ $\{Pre\}f\{Post\}$ が妥当な論理式である場合，「Pre が成立する状況で f を実行し計算が終了すると必ず $Post$ が成立し，しかも f の実行時には，利用者の要求する安全性が満たされる」ことが数学的に保証される．

この検証論理式は，それが妥当な論理式であることを示すために，ある論理体系 T 内で証明される．得られた証明 p は，検証論理式の妥当性の証拠として使われる．T は一般に一階述語論理，高階論理，あるいは時相論理といった汎用の論理体系であり，その公理は利用者の要求する安全性に依存して定まる．

得られた安全性証明 p をもとに開発者は利用者と対話的通信を行い，送信した f が

$$\mathrm{Safe} = \{f \mid T \text{ における } \{Pre\}f\{Post\} \text{ の 証明 } p \text{ が存在する }\}$$

によって定義される安全なプログラムの集合 Safe に属することを利用者に確認させる．一方，利用者は開発者と対話的通信を行い，受信した f が前記集合 Safe に属すること，すなわち f が安全であることを確認した後に実行する．

開発者と利用者との対話的通信において，f が Safe に属することを示す最も簡単な方法は，作成された安全性証明 p を開発者がそのまま利用者に送ることで

前条件 Pre: $\mathbf{r}_1$:int+int*int ∧ $\mathbf{r}_2$:int+int*int ∧ $\mathbf{r}_3$:int+int*int

プログラム f:

```
        LD r4, 4(r1)   % メモリ番地 r1+4 の内容をレジスタ r4 にロード
        LD r1, 0(r1)   % メモリ番地 r1 の内容をレジスタ r1 にロード
        LD r5, 4(r2)   % メモリ番地 r2+4 の内容をレジスタ r5 にロード
        LD r2, 0(r2)   % メモリ番地 r2 の内容をレジスタ r2 にロード
        LD r6, 4(r3)   % メモリ番地 r3+4 の内容をレジスタ r6 にロード
        LD r3, 0(r3)   % メモリ番地 r3 の内容をレジスタ r3 にロード
        BEQ r1, La     % レジスタ r1 が 0 ならばラベル La の命令を，0 以外ならば次の命令を実行
La      BEQ r2, Lb     % レジスタ r2 が 0 ならばラベル Lb の命令を，0 以外ならば次の命令を実行
Lb      BEQ r3, Lc     % レジスタ r3 が 0 ならばラベル Lc の命令を，0 以外ならば次の命令を実行
Lc      RET            % リターン
```

後条件 $Post$: **sel**($\mathbf{r}_4$)+**sel**($\mathbf{r}_5$)+**sel**($\mathbf{r}_6$):int ∨ **sel**($\mathbf{r}_4$)+$\mathbf{r}_5$:int ∨ **sel**($\mathbf{r}_5$)+$\mathbf{r}_6$:int ∨ $\mathbf{r}_4$+**sel**($\mathbf{r}_6$):int ∨ $\mathbf{r}_4$+$\mathbf{r}_5$+$\mathbf{r}_6$:int

図 2 プログラム例
Fig. 2 Example of a program.

ある．これがすなわち従来の安全性証明つきプログラム配信方式である．この方式の欠点は，利用者の計算能力を f のサイズ $|f|$ についての多項式時間に制限した場合，安全性証明 p のサイズ $|p|$ も $|f|$ の多項式長に制限されてしまうという点である．すなわち，従来方式では，利用者の計算能力に現実的な制限を施した場合，プログラム f が

$$\mathrm{Safe}_h = \{f \mid T \text{ における } \{Pre\}f\{Post\} \text{ の長さ } h(|f|) \text{ 以下の証明 } p \text{ が存在する }\}$$

（ただし h はある多項式）によって定義される集合 Safe_h に属することしか示すことができない．これらの集合 Safe_h の全体は NP にほかならない．

これに対し対話型安全性証明つきプログラム配信方式は，f が Safe に属することを示すために，対話型証明プロトコル[14] を利用する．その結果，利用者の計算能力に現実的な制限を施した場合，PSPACE に含まれる任意の Safe に対し，プログラム f が Safe に属するかどうかを示すことができる[15),16)]．すなわち，本方式は，適用可能なプログラムの集合（あるいは適用可能な安全性といってもよい）の全体を従来の NP から PSPACE へと拡大することで，複雑な安全性が要求される多様なプログラムを安全に転送・実行できるようにした．

プログラムの実行前に利用者が必ず開発者と通信しなければならないという制約は，プログラムの自律的な移動可能性の概念と衝突するため，本方式の限界と考えられる．しかし，サーバからダウンロード後にインストールして実行するといった典型的なプログラム移動形態に対しては本手法は有効であり，WWW ブラウザやオペレーティングシステム・カーネルの安全な拡張に利用できると期待される．

3. 機械語プログラムのメモリ安全性

本章では，機械語プログラムのメモリ安全性を例にとり，その計算の複雑さについて述べる．メモリ安全性とは，プログラムの実行中に，禁止された領域へのメモリアクセスがないという性質であり，最も基本的な安全性の 1 つとされる．

プログラムの一例として，Alpha アセンブリ言語[24)] で記述された簡単なプログラムを**図 2** に示す．レジスタ $\mathbf{r}_1$，$\mathbf{r}_2$，$\mathbf{r}_3$ の値に応じて分岐するプログラムである．

いま，利用者は代表的な関数型言語である Standard ML 言語[25)] の型つき中間言語コンパイラ実行時システム[26)] を搭載していると仮定する．この利用者システムは，その安全性要求やプログラムの前条件・後条件といった仕様を型の概念を用いて規定する．たとえば先のプログラム例の前条件・後条件も型を用いて規定される（図 2）．ここで，型 int の値は 32 ビットで表される自然数である．型 $\tau_1*\tau_2$ の値は，型 τ_1 と型 τ_2 の値が格納された隣接するメモリ番地へのポインタである．型 $\tau_1+\tau_2$ の値は隣接するメモリ番地へのポインタであり，最初の番地にはタグ（0 または 1）が格納され，タグが 0 ならば型 τ_1 の値，タグが 1 ならば型 τ_2 の値が続きの番地に格納される．また，**sel**($\mathbf{r}_i$) はメモリ番地 $\mathbf{r}_i$ の内容を指す．実はこの例は，プログラムがメモリ安全であることと，量化命題論理式

$$\exists X_1 \exists X_2 \exists X_3.\ (X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_2) \wedge (X_2 \vee \neg X_3) \wedge (\neg X_1 \vee X_3) \wedge (\neg X_1 \vee \neg X_2 \vee \neg X_3)$$

が偽であることが同値となるように作られている．命題変数 X_i の真偽は，4($\mathbf{r}_i$)（タグが格納された番地 0($\mathbf{r}_i$) の続きの番地）が int であるかあるいは int の

組へのポインタであるかということにエンコードされる．さらに，$\mathbf{r}_{3+i}$ にその内容を読み込むことにより，真偽は $\mathbf{r}_{3+i}$: int あるいは $\mathbf{sel}(\mathbf{r}_{3+i})$: int の成立へと変換される．

プログラム f がこの利用者システムでメモリ安全に実行可能であることを意味する検証論理式 $\{Pre\}f\{Post\}$ は，機械語プログラムの操作的意味論に基づく Floyd 流検証論理式生成器[27] によって自動的に生成される．検証論理式は，その妥当性の証拠を得るために，一階述語論理体系 T において証明される．いずれにせよ検証論理式 $\{Pre\}f\{Post\}$ が T で証明された場合，Standard ML 言語の型つき中間言語コンパイラ実行時システムにおいて，Pre が成立する状況で f を実行し計算が終了すると必ず $Post$ が成立し，しかも（f の実行中に用いられるすべての値に適切な型がつくため）f の実行時のメモリアクセスが安全であることが保証される．たとえば図 2 のプログラム例も検証論理式が証明可能であり，メモリ安全であることが保証される．

検証論理式生成器と論理体系の詳細については文献 19)，20) に譲り，ここでは，安全性の検証に要する計算量について考えてみたい．検証論理式 $\{Pre\}f\{Post\}$ が T で証明可能なプログラム f の全体を Safe としたとき，その計算の複雑さは従来の安全性証明つきプログラム配信方式が適用可能な計算量クラス NP を超えているように思われる．実際，安全性証明のサイズだけでなく検証論理式そのもののサイズがプログラムのサイズに比して指数オーダになりうることが分かる．その理由は，プログラム中に条件分岐命令が n 回続いた場合，たとえループを含まないプログラムであっても 2^n 通りの実行パスが存在することになるが，そのどれをとってもメモリ安全に実行されることが保証されなければならないため，一般に検証論理式は 2^n 個の論理式の論理積で表されるからである．個々の論理式の証明は短く実際に多項式時間で検証可能であるが，検証論理式全体となると証明は指数オーダのサイズとなる．この事実を計算の複雑さの理論の言葉を用いて表現すれば，集合 Safe は co-NP 完全である（証明は文献 19)，20) を参照されたい）．

4. メモリ安全性の対話型証明

co-NP 完全な集合は NP に属さないと考えられているため，機械語プログラムのメモリ安全性は従来方式では必ずしも効率的に検証することができない．この問題が対話型証明の理論[14)~16)] により解決されることを示す．

対話型証明は 3 ステップからなる．最初のステップは安全性の量化命題論理式への翻訳である．メモリ安全に実行可能な Alpha アセンブリ言語プログラムの全体 Safe は co-NP 完全であった．一方，偽の量化命題論理式の全体はより計算量の大きいとされる PSPACE 完全である．したがって，プログラムが Safe に属するか否かといった安全性検証の問題は，量化命題論理式の真偽判定問題に還元可能である．たとえば図 2 のプログラム例が Safe に属することは，3 章に既出の量化命題論理式が偽であることと同値である．

次のステップは量化命題論理式の算術化である．論理式を算術化することで，論理式の真偽判定を数式の値の零判定に置き換えることができる．実際，命題変数 X を自然数変数 X に，$\vee$ を $+$ に，$\wedge$ を $\times$ に，$\neg X$ を $1-X$ に，そして量化子 $\exists$ を和 Σ に変換することで算術化は完了する．たとえば，先の量化命題論理式が偽であることは，次の数式の値が零であることと同値である．

$$\sum_{X_1=0}^{1}\sum_{X_2=0}^{1}\sum_{X_3=0}^{1}(X_1+X_2+X_3)(X_1+(1-X_2))(X_2+(1-X_3))((1-X_1)+X_3)((1-X_1)+(1-X_2)+(1-X_3)).$$

このような数式の値を通常の展開による計算手法で求めようとした場合，その計算時間は Σ 記号が入れ子になるにつれ指数関数的に増大する．一方，Σ 記号の入れ子の深さは，元のプログラム中の条件分岐命令の個数を反映する．したがって，プログラムの安全性の検証をこのように数式の評価に還元したとしても，プログラムが大規模になれば計算時間の爆発の問題は依然として避けがたい．しかし，算術化によって利用可能となった代数的な手法を適用し，この問題が解決されることを次に示す．

上記数式が零であることを証明する対話型プロトコルの実行例を**図 3** に示す．対話は実質的に 3 ラウンドからなり，各ラウンドが 1 つの Σ 記号に対応する．各ラウンドごとに 1 つの数式が提示され，利用者はその値の正当性を開発者との対話を通じて検証する．なお，実際のプロトコルにおいては，元の数式の長さ l に応じて決まる長さ $\mathcal{O}(l)$ の適当な素数 q を法とした計算が行われるが，簡単のためここでは省略する．まず開発者は，各ラウンドで提示された数式から Σ 記号を 1 つ除去し，それを展開して得られた 1 変数多項式（の係数）をヒント情報として利用者に送る．続いて利用者は，各ラウンドごとに自動的に決まる検証タスクを実行する．具体的には，受け取った多項式の 0

開発者（証明者） **利用者（検証者）**

第 1 ラウンドは以下の証明・検証を目的とする:

$$A \equiv \sum_{X_1=0}^{1}\sum_{X_2=0}^{1}\sum_{X_3=0}^{1}(X_1+X_2+X_3)(X_1+(1-X_2))(X_2+(1-X_3))((1-X_1)+X_3)((1-X_1)+(1-X_2)+(1-X_3)) = 0?$$

$$A[X_1] \equiv \sum_{X_2=0}^{1}\sum_{X_3=0}^{1}(X_1+X_2+X_3)(X_1+(1-X_2))(X_2+(1-X_3))((1-X_1)+X_3)((1-X_1)+(1-X_2)+(1-X_3)) = 3X_1^4-6X_1^3-6X_1^2+9X_1 \equiv \alpha[X_1]$$

開発者は 情報 1 を送付 →

← **利用者は受け取った情報に基づき タスク 1 を実行後，乱数 2 を送付**

第 2 ラウンドは以下の証明・検証を目的とする:

$$A[2] = \sum_{X_2=0}^{1}\sum_{X_3=0}^{1}(2+X_2+X_3)(3-X_2)(X_2-X_3+1)(X_3-1)(1-X_2-X_3) = \alpha[2]?$$

$$A[2][X_2] \equiv \sum_{X_3=0}^{1}(2+X_2+X_3)(3-X_2)(X_2-X_3+1)(X_3-1)(1-X_2-X_3) = -X_2^4+X_2^3+7X_2^2-X_2-6 \equiv \alpha[2][X_2]$$

開発者は 情報 2 を送付 →

← **利用者は受け取った情報に基づき タスク 2 を実行後，乱数 4 を送付**

第 3 ラウンドは以下の証明・検証を目的とする:

$$A[2][4] = \sum_{X_3=0}^{1}(6+X_3)(-1)(5-X_3)(X_3-1)(-3-X_3) = \alpha[2][4]?$$

$$A[2][4][X_3] \equiv (6+X_3)(-1)(5-X_3)(X_3-1)(-3-X_3) = -X_3^4-3X_3^3+31X_3^2+63X_3-90 \equiv \alpha[2][4][X_3]$$

開発者は 情報 3 を送付 →

← **利用者は受け取った情報に基づき タスク 3 を実行後，乱数 7 を送付**

最終ラウンドは以下の証明・検証を目的とする:

$$A[2][4][7] = (13)(-1)(-2)(6)(-10) = \alpha[2][4][7]?$$

← **利用者は開発者からの情報なしに単独で タスク 4 を実行**

開発者は 情報 5 を送付 →

← **利用者は受け取った情報に基づき タスク 5 を実行**
最終的に安全性を確認!

情報 1: $A[X_1]$ **を展開して得られた多項式** $\alpha[X_1]$ **の係数** $(3,-6,-6,9,0)$
タスク 1: $A = A[0]+A[1] = \alpha[0]+\alpha[1] = 0+0$ **が** 0 **に一致するかどうかの検証**
情報 2: $A[2][X_2]$ **を展開して得られた多項式** $\alpha[2][X_2]$ **の係数** $(-1,1,7,-1,-6)$
タスク 2: $A[2] = A[2][0]+A[2][1] = \alpha[2][0]+\alpha[2][1] = -6+0$ **が** $\alpha[2] = -6$ **に一致するかどうかの検証**
情報 3: $A[2][4][X_3]$ **を展開して得られた多項式** $\alpha[2][4][X_3]$ **の係数** $(-1,-3,31,63,-90)$
タスク 3: $A[2][4] = A[2][4][0]+A[2][4][1] = \alpha[2][4][0]+\alpha[2][4][1] = -90+0$ **が** $\alpha[2][4] = -90$ **に一致するかどうかの検証**
タスク 4: $A[2][4][7] = (13)(-1)(-2)(6)(-10) = -1560$ **が** $\alpha[2][4][7] = -1560$ **に一致するかどうかの検証**
情報 5: 該当なし
タスク 5: 該当なし

図 3 対話型証明プロトコル実行例
Fig. 3 Example of a protocol execution for interactive proof-carrying code.

と 1 におけるそれぞれの値の和を計算し，それが期待される値と等しいかどうかを検証する．利用者の計算能力には制限があるが，このようにヒント情報を利用することにより，複雑な数式の値を効率的に計算することができる．この検証をパスすると，続いてヒント情報の妥当性を検証する新しいラウンドに移行し，Σ 記号の 1 つ少ない新しい数式が提示され，開発者によるヒントの送付と利用者による値の検証が繰り返され

る．不正な開発者が妥当でないヒントを送付した場合が問題であるが，このような攻撃に対処するため利用者は，各ラウンドで Σ 記号の 1 つ少ない数式を提示する際に変数に代入する定数を，乱数で指定する．さらに最後のラウンドでは，送付された多項式 $\alpha[2][4][X_3]$ が個々の X_3 の値について $A[2][4][X_3]$ と一致するかどうかをヒントなしで調べることができるので，最終的にヒントの妥当性を確認できる．このようにして，利用者は最初に提示された数式の値が零であること，すなわち我々のプログラム例が安全に実行可能であることを最終的に結論づける．

本プロトコル実行中における利用者のタスクは多項式の値の評価・その和の計算・値の比較といった簡単な演算のみから構成される．さらに，対話のラウンド数は Σ 記号の数にほかならない．その結果，利用者が計算に要する時間および対話のラウンド数はいずれもプログラムのサイズの多項式で抑えられる（もちろん，プロトコル実行の前段階としてプログラムの安全性を量化命題論理式の真偽判定に還元する処理が利用者に課せられるが，それに要する時間も多項式で抑えられる）．すなわち利用者は効率的にプログラムの安全性を検証できる．一方，開発者が計算に要する時間は多項式では抑えられない．しかし，プログラムの送り手には相応の高い計算能力を求めるものの，受け手の計算量を抑える本方式は，ユビキタス環境において高速なサーバから携帯端末や組み込みシステムなどの小型計算機にプログラムをダウンロードし実行する場合に有用であると考えられる．

また，このように構成されるプロトコルが保証する安全性は確率的である．すなわち，元のプログラムが安全でない場合，このプロトコルに従った利用者が誤ってそれを安全だと結論づけてしまう確率が微少ではあるが存在する．特に問題となるのは，不正な開発者による戦略的な偽のヒント情報の送信が，検証の誤り確率の増大を招く場合である．しかしその確率も無視できるほど小さい．なぜなら，利用者は各ラウンドで提示する数式を乱数により決定するため，不正な開発者が自分にとって都合のよい検証のシナリオを成立させようと試みても，それが成功することはきわめて稀だからである．より正確には，プログラムのサイズを k とした場合に検証の誤り確率が $1/2^k$ 以下に抑えらえることが，l 次方程式の解はたかだか l 個であるいう事実を利用することにより示される．

5. メモリ安全性のゼロ知識対話型証明

図 3 のプロトコルは，決定性証明者（すなわち乱数を使用しない証明者）による絶対完全性（すなわち確率 1 の完全性）を持つ Arthur-Merlin プロトコル[28]であるため，文献 22) に示される代表的構成法を適用することでただちにゼロ知識化できる．すなわち，共通入力を A とするときの検証者から証明者への転送情報（乱数）列 $x_1, \ldots, x_n$ および証明者から検証者への転送情報（ヒント）列 $y_1, \ldots, y_n$ からなるプロトコル実行を，次のように修正する．

(1) $y_1, \ldots, y_n$ の代わりに，一方向性関数の仮定から存在が保証される秘匿確率暗号方式（secure probabilistic encryption scheme）E と乱数列 $d_1, \ldots, d_n$ を用いたそれらの暗号化 $y'_1 = E(y_1, d_1), \ldots, y'_n = E(y_n, d_n)$ を転送する．

(2) 最後に証明者は検証者に

$$\begin{aligned}&\exists y_1 \cdots \exists y_n \exists d_1 \cdots \exists d_n.\\ &\quad y'_1 = E(y_1, d_1) \wedge \cdots \wedge y'_n = E(y_n, d_n)\\ &\quad \wedge P(A, x_1, \ldots, x_n, y_1, \ldots, y_n)\end{aligned}$$

なる NP 述語をゼロ知識で証明する．ここで述語 P は元のプロトコル実行において転送情報列 $x_1, \ldots, x_n, y_1, \ldots, y_n$ が満たすべき性質を表す．

しかしこの構成法では通信複雑度が著しく増大する．本論文では，文献 23) に基づき，通信複雑度の増加を抑えたゼロ知識対話型証明プロトコルを示す．

情報の内容自体を秘密に保ったまま，内容の事後変更を不可能とする十分な証拠をあらかじめ相手に渡しておくために，コミットメント（秘密証拠供託）関数 commit を本プロトコルにおいて利用する．セキュリティパラメータ l と長さ $\mathcal{O}(l)$ の素数 q に対し，証明者は秘密にしたい情報 $a \in \{0, 1, \ldots, q-1\}$ と乱数 $r \in \{0,1\}^l$ をもとに証拠 $\mathsf{commit}(a, r) \in \{0,1\}^l$ を計算して検証者に送付する（コミットメント・フェーズ）．その後，証明者による a と r の開示を受けた検証者は，これらの情報に基づき先の証拠が確かに $\mathsf{commit}(a, r)$ であることを計算し情報 a が事後変更されていないことを確認する（開示フェーズ）．このような秘密証拠供託の実現は，関数 commit に課せられた次の 2 つの性質により保証される．

(1) $\mathsf{commit}(a, r)$ の値から秘密情報 a に関する知識を得ることは計算論的に困難である．すなわち，異なる 2 つの秘密情報に対するコミットメントの確率分布は多項式時間識別不可能である．

(2) $\mathsf{commit}(a, r)$ の値から a は一意に決まる．すなわち，秘密情報の事後変更は不可能である．

簡単のため適当な r に対する $\mathsf{commit}(a, r)$ を $\mathsf{C}(a)$ と略記する．これにあわせ，コミットメント間の等

情報 1: $A[X_1]$ **を展開して得られた多項式** $\alpha[X_1]$ **の係数のコミット メント** $(\mathsf{C}(3), \mathsf{C}(-6), \mathsf{C}(-6), \mathsf{C}(9), \mathsf{C}(0))$
タスク 1: $\mathsf{C}(A) = \mathsf{C}(A[0] + A[1]) = \mathsf{C}(\alpha[0])\mathsf{C}(\alpha[1])$ **の計算**
情報 2: $A[2][X_2]$ **を展開して得られた多項式** $\alpha[2][X_2]$ **の係数のコミット メント** $(\mathsf{C}(-1), \mathsf{C}(1), \mathsf{C}(7), \mathsf{C}(-1), \mathsf{C}(-6))$
タスク 2: $\mathsf{C}(A[2] - \alpha[2]) = \mathsf{C}(A[2][0] + A[2][1] - \alpha[2]) = \mathsf{C}(\alpha[2][0])\mathsf{C}(\alpha[2][1])\mathsf{C}(\alpha[2])^{-1}$ **の計算**
情報 3: $A[2][4][X_3]$ **を展開して得られた多項式** $\alpha[2][4][X_3]$ **の係数のコミット メント** $(\mathsf{C}(-1), \mathsf{C}(-3), \mathsf{C}(31), \mathsf{C}(63), \mathsf{C}(-90))$
タスク 3: $\mathsf{C}(A[2][4] - \alpha[2][4]) = \mathsf{C}(A[2][4][0] + A[2][4][1] - \alpha[2][4]) = \mathsf{C}(\alpha[2][4][0])\mathsf{C}(\alpha[2][4][1])\mathsf{C}(\alpha[2][4])^{-1}$ **の計算**
タスク 4: $\mathsf{C}(A[2][4][7] - \alpha[2][4][7]) = \mathsf{C}((13)(-1)(-2)(6)(-10) - \alpha[2][4][7]) = \mathsf{C}(-1560)\mathsf{C}(\alpha[2][4][7])^{-1}$ **の計算**
情報 5:
$$\left\{\begin{array}{r} \mathsf{C}(\alpha[0])\mathsf{C}(\alpha[1]) = \mathsf{C}(0) \\ \mathsf{C}(\alpha[2][0])\mathsf{C}(\alpha[2][1])\mathsf{C}(\alpha[2])^{-1} = \mathsf{C}(0) \\ \mathsf{C}(\alpha[2][4][0])\mathsf{C}(\alpha[2][4][1])\mathsf{C}(\alpha[2][4])^{-1} = \mathsf{C}(0) \\ \mathsf{C}(-1560)\mathsf{C}(\alpha[2][4][7])^{-1} = \mathsf{C}(0) \end{array}\right\}$$
すなわち 利用者の計算値がすべて 0 **のコミット メントであること を証明するための開示情報**
タスク 5: **これまでの計算値がすべて** 0 **のコミット メントであることの検証**

図 4 ゼロ知識対話型証明プロトコル実行例（図 3 との差分）
Fig. 4 Example of a protocol execution for zero-knowledge proof-carrying code (which contains only the differences from Fig. 3).

号 $C = C'$ を，C と C' がある a に対し集合 $\{\mathsf{commit}(a, r) \mid r\}$ の要素であることと定義する．$\mathsf{C}(a)$ は同値関係 $=$ による各同値類 $\{\mathsf{commit}(a, r) \mid r\}$ の代表元と見なせる．

一方向性の準同型写像を用いて commit を構成することにより☆，commit に次の性質を持たせることができる[23]．すなわち，与えられた 2 つのコミットメント $A = \mathsf{C}(a)$ および $B = \mathsf{C}(b)$ から $\mathsf{C}(a + b \bmod q)$ および $\mathsf{C}(a - b \bmod q)$ が多項式時間で計算可能である．各コミットメントは乗法群の要素であるため，これらをそれぞれ AB および AB^{-1} と記す．この性質からただちに，コミットメント $A = \mathsf{C}(a)$ が与えられれば，任意の定数 c に対し $\mathsf{C}(ca \bmod q)$ が多項式時間計算可能となる．これを A^c と記す．

一方向性準同型写像を用いて構成された commit を利用することにより，通信複雑度を抑えたゼロ知識化が可能となる．我々のプログラム例に対するゼロ知識対話型証明プロトコルの実行例は，図 3 の破線で囲んだ部分を**図 4** の内容に置き換えることによって得られる．図 3 においては開発者がヒントとして多項式の係数そのものを送付していたのに対し，ゼロ知識対話型証明プロトコルでは係数のコミットメントを送付する点が異なる．準同型写像を用いたコミットメント関数の性質により，利用者は，これら係数のコミットメントのみから，目的とする式のコミットメントを計算することが可能となる．たとえば，第 2 ラウンドで利用者は目的とする式 $A[2] - \alpha[2]$ のコミットメント $\mathsf{C}(\alpha[2][0])\mathsf{C}(\alpha[2][1])\mathsf{C}(\alpha[2])^{-1}$ の計算を行うが，それまでに受け取った係数のコミットメントのみを用いて

$$\begin{array}{c} \mathsf{C}(-1)^0\mathsf{C}(1)^0\mathsf{C}(7)^0\mathsf{C}(-1)^0\mathsf{C}(-6) \\ \mathsf{C}(-1)^1\mathsf{C}(1)^1\mathsf{C}(7)^1\mathsf{C}(-1)^1\mathsf{C}(-6) \\ \big(\mathsf{C}(3)^{16}\mathsf{C}(-6)^8\mathsf{C}(-6)^4\mathsf{C}(9)^2\mathsf{C}(0)\big)^{-1} \end{array}$$

として計算することができる．そして対話の最終段階において，これら目的とする式の値を，開発者はコミットメントの内容を開示することにより利用者に納得させる．

このゼロ知識化によって，各ラウンドで開発者から利用者に送付される情報量は定数倍に抑えられる．また，ラウンド数も定数回の増加に抑えられる．

6. 応　　用

ゼロ知識対話型安全性証明つきプログラム配信方式は，次のような応用可能性を持つ．今，ネットワーク上で競合する複数のエージェント $P_1, P_2, \ldots, P_n$ が顧客 V に安全性証明を添えてプログラムを販売しようとしているとする．このとき，従来の安全性証明つき配信方式によれば，証明は公開されているため盗用される危険性が高い．顧客 V になりすました攻撃者 P_j がプログラムと証明を別の P_i から盗み，自分が開発したと虚偽の主張をして顧客 V に売りつける恐れがある．一方，本論文で提案するゼロ知識対話型安全性証明つき配信方式によれば，証明の中身を把握しているのは正規の開発者に限定される（より正確には，正規の開発者以外の者が証明の中身を把握するには証明を作成するのと同等の高い計算コストを要する）．すなわち，安全性証明がプログラムの著作権として機能し，この著作権がゼロ知識性によりネットワーク上で保護される．

もちろん，ソフトウェアの著作権を保護する手法は数多く提案されている[29)~33)]．本論文が提案する手法の特徴は，プログラムの著作権を保護したいという開発者側の要求だけでなく，プログラムを安全に実行し

☆ このような一方向性準同型写像の作成法は複数知られている．たとえば，離散対数問題に基づく ElGamal 暗号の安全性（すなわち異なる 2 つの平文の ElGamal 暗号方式による確率的暗号化が多項式時間識別不可能であること）を仮定することで具体的に作成可能である[23]．

たいという利用者側の要求も考慮し，両者を同時に満たすという点にある．

7. まとめと今後の課題

本論文では，プログラム流通のセキュリティ向上に資することを目的として，ゼロ知識対話型安全性証明つきプログラム配信方式を提案した．本方式は従来の安全性証明つきプログラム配信方式に比べて次の長所を持っている．

(1) 証明・検証プロトコルの対話的・確率的拡張により，大きなサイズの安全性証明を効率的に検証することができる．

(2) プロトコルのゼロ知識化により，安全性証明の盗用・悪用を防ぐことができる．

特に，従来方式では証明が指数オーダのサイズとなる機械語プログラムのメモリ安全性を例題として，その安全性が提案方式により効率的に検証可能となること，また通信複雑度の増加を抑えてゼロ知識化できることを示した．さらに，安全性証明をプログラムの著作権と解釈する応用について述べた．本論文で提案した方式は，プログラムの安全性を効率的に検証したいという利用者側の要求と，プログラムの著作権を保護したいという開発者側の要求が，同時に満たされるという特徴を持つ．

今後の課題として，まず，本論文で例示した証明・検証プロトコルの実用的視点からの評価が重要である．与えられたプログラムから対応するプロトコルを構成する一般的手順は複雑であり，実用上の問題となりうる．一般には，操作的意味論が定義された仮想的な計算モデルを与え，その計算モデル上でのプログラム実行の諸性質を量化命題論理式の真偽判定問題に還元する手順を与える必要がある．従来研究は計算モデルとして Turing 機械を採用したものがほとんどであるが，実用的なプログラミング言語との隔たりが大きいという問題があった．これを改善するため RAM (Random Access Machine) を計算モデルとするアプローチが文献 34) に見られる．また，最悪時計算量に基づく多項式時間限定という理論的基準も，検証プロトコルが実用レベルで効率的であることを必ずしも保証しない．さらに，安全性証明つきプログラム配信方式の最大の特徴は検証側システムが軽量で済むという点にあり，本論文で提案した方式もその特徴を継承する方針で設計されているが，対話的・確率的な拡張の結果増大した検証側システムの複雑度を，方式実現を通して評価する必要がある．

また，本方式が提供する (1) 複雑な証明の効率的検証と (2) 証明の盗用・悪用防止の機能のそれぞれについて，検討すべき課題が残っている．まず，前者に関しては，メモリ安全性より複雑な安全性への本方式の適用が重要課題である．メモリ安全性に関しては，本論文で提案した理論的アプローチ以外にも，神託を利用した安全性証明の情報圧縮の方法（証明図そのものを送信するのではなく証明図を構成する推論規則のインデックス列を送信する方法[12),13)]）や，段階的アルゴリズムを用いて検証論理式の指数関数的爆発を抑える方法[35)] を適用することで，証明のサイズを抑えることが可能である．しかし，メモリ安全性を超える複雑度が予想されるその他の安全性の検証，たとえばデバイスドライバなどの時相安全性をモデル検査[36)] により検証する場合には，これら既存の方法の有効性は未保証である．近年，証明に基づきプログラムの時相安全性を保証する研究[7),37)~39)] がさかんになりつつあるが，これらの複雑な安全性をともなう例題の中には計算量が PSPACE で抑えられるものも多く，本論文の提案手法の有用性が期待できる．

安全性証明の盗用・悪用防止に関しては，本論文が提案する方式は秘匿できる情報の制約により実際的な効用に限界があり，方式の拡張を検討する必要がある．本論文の定式化においては，プログラムの前条件・後条件の利用者による事前公開に加え，(ループを含むプログラムの場合には) ループ不変条件の開発者による開示を仮定している[19),20)]．これらの不変条件は証明と異なり秘匿されないため，不変条件を利用した攻撃を防ぐことができない．一般に不変条件はプログラムの実行時の振舞いに関する重要な情報を含み，それを用いた解析により難読化[30)] や透かしの挿入[31),33)] などの様々な操作をプログラムに施すことができる．そこで，不変条件も証明とともに秘匿する新たな枠組みを検討する価値がある．この拡張された枠組みにおいては利用者は不変条件を独自に特定しなければならないため，安全なプログラムの全体 Safe は co-NP を超える複雑さを持つ可能性がある．しかし，不変条件のとりうる形の制約などにより，検証の複雑さが依然として PSPACE で抑えらえるならば，本拡張方式の採用により，証明に基づくリバースエンジニアリングの危険性を減少させることができると期待される．

また，確率的検査可能証明[40),41)] をはじめとする他証明概念を採用し，証明に基づく安全なプログラムの配信方式を考案・比較検討することも重要であると考えている．

謝辞 東京工業大学の米崎直樹先生，渡辺治先生，東北大学の小林直樹先生，NTT コミュニケーション

科学基礎研究所の白柳潔氏，真野健氏，櫻田英樹氏，担当編集委員と査読者の方々からは，本研究に関して有益な助言をいただきました．ここに深く感謝いたします．

参 考 文 献

1) Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J. and Minden, G.J.: A Survey of Active Network Research, *IEEE Communications Magazine*, Vol.35, No.1, pp. 80–86 (1997).
2) McGraw, G. and Morrisett, G.: Attacking Malicious Code: A Report to the Infosec Research Council, *IEEE Software*, Vol.17, No.5, pp.33–41 (2000).
3) Lindholm, T. and Yellin, F.: *The Java Virtual Machine Specification*, 2nd Edition, Addison-Wesley, Reading, Massachusetts (1999).
4) Necula, G.C. and Lee, P.: Safe Kernel Extensions without Run-Time Checking, *Proc. Second Symp. Operating Systems Design and Implementation*, pp.229–243 (1996).
5) Necula, G.C.: Proof-Carrying Code, *Proc. 24th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, pp.106–119, ACM Press (1997).
6) Appel, A.W. and Felty, A.P.: A Semantic Model of Types and Machine Instruction for Proof-Carrying Code, *Proc. 27th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, pp.243–253, ACM Press (2000).
7) Bernard, A. and Lee, P.: Temporal Logic for Proof-Carrying Code, *Proc. 18th Int'l Conf. Automated Deduction*, Lecture Notes in Computer Science, Vol.2392, pp.31–46, Springer-Verlag, Berlin (2002).
8) Necula, G.C. and Lee, P.: The Design and Implementation of a Certifying Compiler, *Proc. 1998 ACM SIGPLAN Conf. Programming Language Design and Implementation*, pp.333–344, ACM Press (1998).
9) Morrisett, G., Walker, D., Crary, K. and Glew, N.: From System F to Typed Assembly Language, *ACM Trans. Programming Languages and Systems*, Vol.21, No.3, pp.528–569 (1999).
10) Ohori, A.: A Curry-Howard Isomorphism for Compilation and Program Execution, *Proc. Fourth Int'l Conf. Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, Vol.1581, pp.258–279, Springer-Verlag, Berlin (1999).
11) Colby, C., Lee, P., Necula, G.C., Blau, F., Plesko, M. and Cline, K.: A Certifying Compiler for Java, *Proc. 2000 ACM SIGPLAN Conf. Programming Language Design and Implementation*, pp.95–107, ACM Press (2000).
12) Necula, G.C. and Rahul, S.P.: Oracle-Based Checking of Untrusted Software, *Proc. 28th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, pp.142–154, ACM Press (2001).
13) Necula, G.C.: A Scalable Architecture for Proof-Carrying Code, *Proc. Fifth Int'l Symp. Functional and Logic Programming*, Lecture Notes in Computer Science, Vol.2024, pp.21–39, Springer-Verlag, Berlin (2001).
14) Goldwasser, S., Micali, S. and Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems, *SIAM J. Computing*, Vol.18, No.1, pp.186–208 (1989).
15) Lund, C., Fortnow, L., Karloff, H. and Nisan, N.: Algebraic Method for Interactive Proof Systems, *J. ACM*, Vol.39, No.4, pp.859–868 (1992).
16) Shamir, A.: IP = PSPACE, *J. ACM*, Vol.39, No.4, pp.869–877 (1992).
17) Shen, A.: IP = PSPACE: Simplified Proof, *J. ACM*, Vol.39, No.4, pp.878–880 (1992).
18) Feige, U. and Nissim, K.: On the Use of Interactive Proofs for Formal Program Verification, Technical Report CS97-06, Mathematics and Computer Science, Weizmann Institute of Science (1997).
19) Tsukada, Y.: Mobile Codes with Interactive Proofs: An Approach to Provably Safe Evolution of Distributed Software Systems, *Proc. 2000 Int'l Symp. Principles of Software Evolution*, pp.23–27, IEEE Computer Society Press (2001).
20) Tsukada, Y.: Interactive and Probabilistic Proof of Mobile Code Safety, *Automated Software Engineering* (To appear).
21) Impagliazzo, R. and Yung, M.: Direct Minimum-Knowledge Computations, *Advances in Cryptology—Crypto'87*, Lecture Notes in Computer Science, Vol.293, pp.40–51, Springer-Verlag, Berlin (1988).
22) Ben-Or, M., Goldreich, O., Goldwasser, S., Håstad, J., Kilian, J., Micali, S. and Rogaway, P.: Everything Provable is Provable in Zero-Knowledge, *Advances in Cryptology—Crypto'88*, Lecture Notes in Computer Science, Vol.403, pp.37–56, Springer-Verlag, Berlin (1990).
23) Cramer, R. and Damgård, I.: Zero-Knowledge Proofs for Finite Field Arithmetic or: Can Zero-Knowledge be for Free?, *Advances in*

Cryptology—Crypto'98, Lecture Notes in Computer Science, Vol.1462, pp.424–441, Springer-Verlag, Berlin (1998).
24) Sites, R.L.: *Alpha Architecture Reference Manual*, Digital Press, Burlington, Massachusetts (1992).
25) Milner, R., Tofte, M. and Harper, R.: *The Definition of Standard ML*, The MIT Press, Cambridge, Massachusetts (1990).
26) Tarditi, D., Morrisett, G., Cheng, P., Stone, C., Harper, R. and Lee, P.: TIL: A Type-Directed Optimizing Compiler for ML, *Proc. 1996 ACM SIGPLAN Conf. Programming Language Design and Implementation*, pp.181–192, ACM Press (1996).
27) Floyd, R.W.: Assigning Meanings to Programs, *Mathematical Aspects of Computer Science*, Symposia in Applied Mathematics, Vol.19, pp.19–32, American Mathematical Society (1967).
28) Babai, L.: Trading Group Theory for Randomness, *Proc. 17th Annual ACM Symp. Theory of Computing*, pp.421–429, ACM Press (1985).
29) Mori, R. and Kawahara, M.: Superdistribution: The Concept and the Architecture, *Trans. IEICE*, Vol.E73, No.7, pp.1133–1146 (1990).
30) 門田暁人，高田義広，鳥居宏次：ループを含むプログラムを難読化する方法の提案，電子情報通信学会論文誌 D-I，Vol.J80-D-I, No.7, pp.644–652 (1997).
31) 一杉裕志：ソフトウェア電子すかしの挿入法，攻撃法，評価法，実装法，夏のプログラミングシンポジウム報告集，pp.57–64, 情報処理学会 (1997).
32) Collberg, C. and Thomborson, C.: Software Watermarking: Models and Dynamic Embeddings, *Proc. 26th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, pp.311–324, ACM Press (1999).
33) 門田暁人，松本健一，飯田　元，井上克朗，鳥居宏次：Java クラスファイルに対する電子透かし法，情報処理学会論文誌，Vol.41, No.11, pp.3001–3009 (2000).
34) Batu, T., Rubinfeld, R. and White, P.: Runtime Verification of Remotely Executed Code Using Probabilistically Checkable Proof Systems, *Proc. FLoC Workshop on Run-Time Result Verification* (1999).
35) Flanagan, C. and Saxe, J.B.: Avoiding Exponential Explosion: Generating Compact Verification Conditions, *Proc. 28th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages*, pp.193–205, ACM Press (2001).
36) Clarke, Jr., E.M., Grumberg, O. and Peled, D.A.: *Model Checking*, The MIT Press, Cambridge, Massachusetts (1999).
37) Namjoshi, K.S.: Certifying Model Checkers, *Proc. 13th Int'l Conf. Computer Aided Verification*, Lecture Notes in Computer Science, Vol.2102, pp.2–13, Springer-Verlag, Berlin (2001).
38) Henzinger, T.A., Jhala, R., Majumdar, R., Necula, G.C., Sutre, G. and Weimer, W.: Temporal-Safety Proofs for Systems Code, *Proc. 14th Int'l Conf. Computer Aided Verification*, Lecture Notes in Computer Science, Vol.2404, pp.526–538, Springer-Verlag, Berlin (2002).
39) Xia, S. and Hook, J.: Abstraction-Carrying Code: A New Method to Certify Temporal Properties, *Formal Techniques for Java-like Programs 2003* (*Proceedings*), Technical Report, No.408, ETH Zurich (2003).
40) Fortnow, L., Rompel, J. and Sipser, M.: On the Power of Multi-Prover Interactive Protocols, *Proc. 3rd IEEE Symp. Structure in Complexity Theory*, pp.156–161 (1988).
41) Arora, S. and Safra, S.: Probabilistic Checking of Proofs: A New Characterization of NP, *Proc. 33rd IEEE Symp. Foundations of Computer Science*, pp.2–13 (1992).

(平成 16 年 4 月 16 日受付)
(平成 16 年 11 月 1 日採録)

塚田　恭章（正会員）

1990 年東京工業大学大学院理工学研究科情報科学専攻修士課程修了．同年日本電信電話株式会社入社．現在コミュニケーション科学基礎研究所主任研究員．型理論とロジカルフレームワーク，論理的手法に基づくソフトウェアの安全性検証の研究に従事．日本ソフトウェア科学会，ACM 各会員．