ソフトウェアプロテクションの技術動向（前編）
—ソフトウェア単体での耐タンパー化技術—

門田暁人
奈良先端科学技術大学院大学 情報科学研究科
akito-m@is.naist.jp

Clark Thomborson
Department of Computer Science, The University of Auckland
cthombor@cs.auckland.ac.nz

ソフトウェアプロテクションとは，ソフトウェアの改ざん，解析，コピー，再利用（盗用）などの攻撃からソフトウェアシステムを守る技術の総称であり，難読化（obfuscation），暗号化（encryption），多様化（software diversity）☆1，電子透かし（watermarking），バースマーク（birthmark）☆2などの要素技術がある．難読化と暗号化をまとめて耐タンパー化（tamper proofing, tamper resistance）と呼ぶこともあり，特にタンパー（改ざん）の検出技術をintegrity verification，もしくは，tamper detectionと呼ぶ．

本稿（前編）では，ソフトウェア単体でのプロテクションの技術動向について解説する．特に，難読化を中心に紹介し，それらの技術的課題やその対策，今後の展望を述べる．後編では，より強力なプロテクション技術として，ハードウェアを用いる方式について述べる．

ソフトウェアクラックとプロテクション

ソフトウェアに含まれる秘密

ソフトウェアプロテクションの主目的は，ソフトウェアシステムに含まれる秘密情報を隠蔽する（機密性を保持する）こと，および，ソフトウェアの改ざんや再利用を防ぐことである．この2つの目的は密接に関係しており，ソフトウェアの機密性を保つことができれば，結果的にソフトウェアに対する意味のある改ざんや再利用を防ぐことができる．隠蔽すべき秘密情報として，次のものが挙げられる．

1. システムの安全性にかかわる，もしくは，商業的に価値のあるアルゴリズムやサブルーチン．たとえば，DVD-Videoのディジタルコピー防止用の暗号化規格CSS（Content Scrambling System）の暗号アルゴリズムは本来非公開であり，システムのユーザに知られないように隠蔽することが要求されていた．
2. システムの安全性にかかわるデータ（定数）．たとえば，携帯電話に含まれる端末識別番号やDRMシステムの復号鍵である．CSSの次の世代のディジタルコンテンツ保護規格であるCPPM（Content Protection for Prerecorded Media）/CPRM（Content Protection for Recordable Media）では，暗号アルゴリズム（C2暗号）は公開されているが，コンテンツ再生プレイヤーをソフトウェアで実装する場合に，ソフトウェアに含まれるデバイス鍵（復号鍵の一種），および，S-Boxテーブル（復号に必要な定数の集合）の隠蔽が必須となっている☆3．
3. システム内部の機能分岐点．たとえば，ライセンスチェックを行う分岐文である．ソフトウェア自身のコピーを防ぐための，いわゆるコピープロテクト技術として，フロッピーディスクやCD-ROMメディアの特殊フォーマットを利用したプロテクトやシリアルナンバープロテクトが知られているが，いずれのプロテクト技術においても「正規のコピーか否かをチェックする」分岐文がソフトウェア内部に存在し，この分岐文が発見，改ざんされると，プロテクトが無効化される恐れがある．

☆1 ソフトウェアに個体差を持たせる方式．後編で解説する．
☆2 ソフトウェアの痣（あざ）．http://se.naist.jp/jbirth/
☆3 4C Entity, http://www.4centity.com/

4. 外部インタフェース．たとえば，システムへの完全なアクセス権を与えるメンテナンス用インタフェースである．近年では，Borland Interbase SQLデータベースサーバのメンテナンス用バックドアが露見した事件が知られている☆4．

クラックされるシステム

これまでに，数多くのソフトウェアシステムが“クラック”され，システムの開発者や販売者に深刻な損害を与えてきた．前述のCSS規格について言えば，ある市販のDVDプレイヤーソフトウェアが解析され，コンテンツ復号アルゴリズムと復号鍵が漏洩した．今日ではDVDの暗号解除ツールは容易に入手できるという状態であり☆5，暗号を解除された生コンテンツがP2Pソフトによりインターネットに流出している．

海外では，携帯電話（端末）を大量に盗んで他国へ転売するという犯罪が流行し，大きな問題となった☆6．その主な原因は，端末の識別番号がソフトウェアによって容易に書き換え可能であり，識別番号を変更した端末が国外で使用できたためである．識別番号はソフトウェア内部にハードコーディングされているわけではないが，システムのユーザから隠蔽すべき秘密情報の1つである．

ソフトウェアのコピープロテクト技術は，その大部分がクラックの被害にあってきた．近年のマイクロソフト製品のコピープロテクト方式である「プロダクトアクティベーション」も例外ではなく，プロテクトを無効化するパッチが流通した．

近年では，クライアントサーバシステムのクラックも問題となっている．世界最大の分散コンピューティングプロジェクトであるSETIアットホームでは，クライアントの不正行為が大きな問題となっている．また，オンラインゲームのチート（ユーザデータの書き換えなど）の防止も大きな課題となっており，サーバとクライアント間の通信データを（クライアントから）隠蔽することが求められている．

クラックのためのツール

ソフトウェアのクラックを助長しているのは，多様なソフトウェア解析ツールの存在である．たとえば，逆アセンブラ，逆コンパイラ，デバッガ，バイナリエディタ，メモリダンプツール，アンパッカー，モニタリングツール，ファイルスキャナ等がある[1]．これらは，ソフトウェア開発に役立つ一方で，ソフトウェアクラックにも悪用できる．これらの多くはインターネット上で容易に入手可能であり，ツールを用いてソフトウェアを解析する方法の解説書やWebサイト☆7が数多く存在する．

一方，クラックツールの動作を妨げるための“アンチクラック”ツールも数多く普及している☆8．これらは主にソフトウェアの暗号化／実行時復号化とアンチデバッガ機構を備えている．しかし，これまでほとんどすべてのアンチクラック技術は破られており，万能のアンチクラックツールは存在しないと言われている[1]．

プロテクション技術への期待

今日のソフトウェアプロテクション技術の多くは，ハードウェアの助けを借りることで堅牢な保護の実現を目指している．その1つの方向性は，種々の解析ツールが保護対象のソフトウェアへアクセスできないように物理的に防ぐ方式であり，IBM，Intel，Microsoftなどにより推進されているTrusted Computingプラットフォーム☆9が代表的である．特に，企業ユーザ向けの計算機環境として今後有望である（詳しくは，後編で述べる）．

一方，ソフトウェア単体でのプロテクションは，一般のPC環境で用いられるため，解析ツールによるソフトウェアへのアクセスを防ぐことは困難である．市販されている多くのプロテクションツールでは，解析ツールに情報をなるべく与えないようにプログラムの暗号化，実行時復号を行う機構や，解析ツールを検出してプログラムの動作を停止する機構，解析ツールを誤動作させる機構などを設けている[1]．ただし，これらのプロテクション機構はソフトウェアで実現されるため，それら自身が解析ツールによる攻撃対象となる．

攻撃のしにくさという観点からは，追加のプロテクション機構を持たず，プログラム自身を解析しにくくする方式であるプログラムの難読化が有望である．近年，white-box暗号システム[2]などの有力な難読化法が提案されており，今後の発展が期待されている．

☆4 US-CERT, https://www.kb.cert.org/vuls/id/247371/
☆5 ただし，DVDの暗号を解除することは不正競争防止法に違反し，暗号解除ソフトを開発・配布すること自身も著作権法に違反する（第120条2項）．
☆6 http://www.parliament.uk/commons/lib/research/rp2002/rp02-047.pdf
☆7 Fravia, http://www.woodmann.com/fravia が代表的である．
☆8 ASProtect，ACProtect，Obsidium，FLEXlmなどがある．
☆9 https://www.trustedcomputinggroup.org/

```
int func(int r) {
  int i, k, n=12, p=1;
  for(i = 1; i <= r; i++){
    k = n - i + 1;
    p = p * k / i;
  }
  return p;
}
```

(a)難読化前

```
int func(int r){
  int i, k, n=25, p=3;
  for(i = 1; i <= r; i++){
    k = n - 2 * i + 2;
    p = (p * k - p - k + 1) / 2 / i + 1;
  }
  p = (p - 1) / 2;
  return p;
}
```

(b)難読化後

図 -1　線形変換によるデータ難読化
（$_{12}C_r$ を計算するプログラム）

```
int func(int c) {
  int x = 32;
  return c + x;
}
```

(a) 難読化前

```
int func(int c){
  int c1, c2, x1 = 6, x2 = 13;
  c1 = (c + x1) % 13;
  c2 = (c + x2) % 19;
  return decode(c1, c2);
}

int decode(int a, int b){
   int x = (a+13*(b-a)*3)%247;
   if(x < 0) x+= 247;
   return x;
}
```

(b) 難読化後

図 -2　中国人剰余定理に基づくデータ難読化
（英大文字を小文字に変換するプログラム）

プロテクションの技術動向

一般に，ロバスト（頑健）な安全性設計は，攻撃に対する幅広い防御手段を備える必要がある．攻撃の防止（prevention）だけでなく，攻撃の検出（detection），検出した攻撃に対する応答（response）の機構がそれぞれ必要である．以降では，このうち，攻撃の防止に主に焦点をあてて解説する．

プログラムの難読化

難読化とは，与えられたプログラムをより複雑なプログラムへと変換する技術であり，レイアウト難読化，データ難読化，制御フロー難読化などがある．難読化されたプログラムは，難読化前と同一の機能を持つが，理解や解析がより困難となっている．ここでは，代表的な難読化手法であるデータ難読化と制御フロー難読化について解説する．

データ難読化は，暗号鍵を含むプログラムにおいて鍵データを隠蔽するのに特に有効であり，暗号モジュールに対する電力解析[☆10]への防御手段としても有望である．近年注目されている方法は，プログラム中の隠蔽したい定数xを特定の準同型写像（homomorphism）によって別のドメインの値に変換し，以降のxに対する演算を変換後のドメインで行うようにプログラムを変換する方法である．**図 -1** に $_{12}C_r$ を計算するプログラムにおける値12を線形写像$f(x) = 2x + 1$によって隠蔽した例を示す．この例では，変数nとpの初期値12と1をそれぞれ25と3に変換（符号化）している．また，nとpに対する演算結果も符号化された状態となるように，演算式の変換を行っている．

中国人剰余定理に基づくデータ難読化[☆11]の例を**図 -2** に紹介する．中国人剰余定理とは，互いに素である自然数$m_1, \ldots, m_n$の積$M = m_1 \times m_2 \times \ldots m_n$を法とする剰余類$Z_M = \{0, 1, 2, \ldots, M\text{-}1\}$の各要素が，$m_1, \ldots, m_n$をそれぞれ法とする剰余類$Z_{m1} = \{0, 1, 2, \ldots, m_1\text{-}1\}, \ldots, Z_{mn} = \{0, 1, 2, \ldots, m_n\text{-}1\}$の要素の組に一意的に対応付けられるという性質，および，その解法を表したものである．それぞれの剰余類内で演算（和または積）を行った場合にもこの対応付けが保存されるため，隠蔽したい整数x($x < M$)をn個の整数$x_1 \bmod m_1, x_2 \bmod m_2, \ldots, x_n \bmod m_n$に分割し，分割したまま演算を行い，さらに，必要に応じて分割した値を合成することが可能である．図-2は，アルファベットの大文字A～Zを小文字a～zへと変換する（すなわち，アスキーコード上で$f(x) = x + 32$を計算する）プログラムであり，$x = 32$を剰余変換によって2つの値（$x1 = 6$と$x2 = 13$）に分割することで隠蔽している．

他のデータ難読化法としては，整数をBoolean値の組合せで表現するbit-exploded coding，座標系を変換するcustom base codingなどがある[☆11]．また，適用範囲

☆10 モジュールの消費電力の変化に基づいて鍵や処理内容を解析する手法であり，単純電力解析，電力差分解析，高次電力差文解析がある．
☆11 U.S. Patent 6594761, http://www.freepatentsonline.com/6594761.html

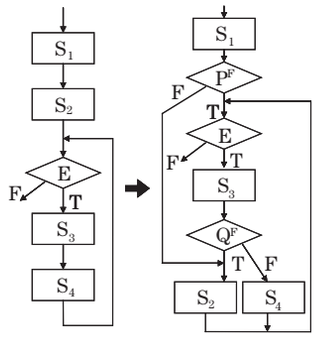

図-3　Opaque Predicate による制御フロー難読化

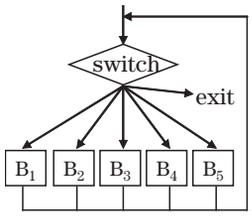

図-4　平坦化された制御フロー

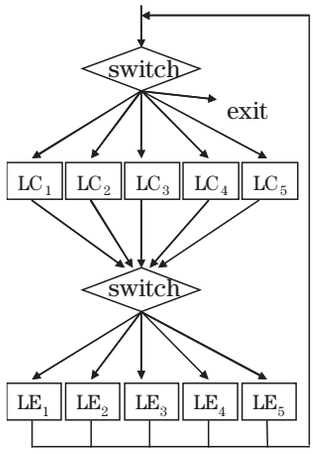

図-5　平坦化された制御フロー（2）

は限られるが，準同型写像を用いない方法として，配列の要素の順序を入れ替える，2個の配列を1つの配列にまとめる，配列の次元を増やす（もしくは減らす）といった方法がある☆12．

一方，制御フローの難読化は，暗号鍵などの秘密データを隠蔽できないため，データの難読化と比べて需要が低いと思われるが，秘密データの位置に関する手がかりを減らしたり，サブルーチンやアルゴリズムの隠蔽に役立つ．以降，代表的な3つの制御フロー難読化法を紹介する．1つめは，フローチャートをより複雑なものへと変換する方式である．Collberg ら☆12は，整数 x と y に対する“if ($7*y*y - 1 == x*x$)”といった恒偽（または恒真）となる条件分岐を追加して制御フローを複雑化する方法を提案している．この方法では，条件分岐が恒偽（恒真）であることを見破られないことが重要となる．ここでの $7*y*y - 1 == x*x$ のような式は，“opaque predicate”と呼ばれる．**図-3** に難読化の例を示す．図中，Q^F と P^F は恒偽の opaque predicate である．Collberg らは，静的解析では恒偽（恒真）となることが特定できない opaque predicate を系統的に作成する方法も提案している☆12．一方，opaque predicate を用いずに制御フローを複雑化する方法も提案されている[3]．また，近年では，より大域的な制御の流れを難読化する方法として，関数やクラス間の関係を複雑化する方法も提案されている．

制御フロー難読化の2つめの方法は，フローチャートを平坦化する方法であり，Wang ら[4]によって提案されている（**図-4**）．Wang らは，分割・平坦化された各基本ブロックの実行制御に変数ポインタを使うことで，静的解析による制御フローの復元（難読化の逆変換）を困難にしている．より高度な難読化法として，Chow ら☆13は，**図-5** に示すように，複数の基本ブロックを融合させ，ダミーの文を混在させた Emulative lump と呼ばれる基本ブロック（図-5 の LE_1〜LE_5）を生成し，さらに，Connective lump と呼ばれる変数変換テーブル（図-5 の LC_1〜LC_5）を導入することで，各 Emulative lump の動作を解析困難にしている．この方法は，実行速度の低下が許容されるならば有望な方法である．

制御フロー難読化の3つめの方法は，Mercaldi[5]によって提案されている例外処理（exception）を用いる方法である．条件分岐を例外処理に置き換えることで，制御フローの静的解析と動的解析がより困難となる．例外処理を用いる方法は，CodeShield や Smokescreen などの一

☆12 U.S. Patent 6668325, http://www.freepatentsonline.com/6668325.html
☆13 U.S. Patent 6779114, http://www.freepatentsonline.com/6779114.html

部の商用難読化ツールでも採用されている．

その他の有力な難読化法としては，メソッドのオーバーライドやオーバーロードを利用したオブジェクト指向言語の難読化法，命令の置き換えや自己書き換えなどを利用した機械語プログラムの難読化法，プログラム中の文字列を暗号化する方法などがある．

今日では，Javaや.NETのバイトコードのための難読化ツールが数多く普及している．また，さまざまな言語のソースコード難読化ツールやx86系バイナリプログラム難読化ツールも存在する．いくつかの最新の難読化手法は，CollbergらのツールSandMark[☆14]に実装されており，難読化の研究に役立つと思われる．

難読化の弱点とその対策

これまで，多くの論文では，難読化の利点や有効性ばかりが強調され，弱点についてはあまり分析されてこなかった．そのため，難読化が実際のところどの程度役立つのかは不明な部分も多い．

一般に，難読化は，ソフトウェアに含まれるあらゆる秘密を隠せるほど強力ではない．プログラムがどんなにうまく難読化されていたとしても，高度な知識と強力なツールを備えた攻撃者は，プログラムからさまざまな情報を得ることができる．理論的な研究として，難読化によってプログラムをブラックボックス化する（プログラムコードが情報を漏らさない状態にする）ことは不可能であることが，ある計算モデル上で証明されている[6]．ただし，現代暗号でさえ，あらゆる攻撃に対して安全であるとは言えない．我々が現実的に目指すべきことは，いくつかの想定された攻撃に対して，現実的な時間では解析に成功しない程度に難読化することである．

以降，難読化の具体的な弱点について，筆者の考えを述べる．データの難読化法は，プログラムファイル，および，実行時のメモリ中に秘密データが現れないように隠蔽できるが，データ復元ルーチンをプログラム中に必ず記述する必要があり，この部分が攻撃の大きな手がかりとなる．たとえば，図-1(b)のreturn文の直前にある“p=(p－1)/2”や図-2のdecode関数である．これらのルーチンを調べることで，攻撃者はデータ隠蔽方法（たとえば，図-1では線形変換$f(x) = 2x+1$）を知ることができる．また，隠蔽したデータに対する演算式も，解析の手がかりを与える．たとえば，図-2(b)の剰余演算から，2つの剰余類の法13と19を知ることができる．また，一般に，プログラムに対する入力は，符号化されていない生の値として引数に渡されるため，解析者に大きな手がかりを与える．たとえば，図-1(b)の引数rは生の値を保持するため，rとの比較演算（i<= r）が行われる変数iもまた生の値を保持することが推測される．これらの問題の対策としては，(1)データ復元ルーチンを他の計算式に紛れさせる，(2)データ変換のドメインを頻繁に変更する，(3)複数のデータ難読化法を併用する，といった処置が考えられる．

なお，一般に，データ難読化によって秘密データが完全に隠蔽できていたとしても，それだけでは不十分である．攻撃者は，秘匿データを含むサブルーチンを丸ごと抽出し，再利用する可能性があるためである．たとえば，DVDプレーヤソフトでは，デバイス鍵を含むコンテンツ復号ルーチンそのものが攻撃者に抽出・再利用される恐れがある．難読化を行う者は，秘密データを含むサブルーチンが攻撃者に抽出されないように，大域的な制御フローの難読化を併用することが望ましい．

一方，多くの制御フロー難読化法の弱点は，本来はプログラムの実行に不要な式であるopaque predicateを必要とする点である．opaque predicateは，たとえ静的解析が理論的に困難であったとしても，攻撃者が一目見て発見できる可能性がある．たとえば，前述の$7*y*y-1==x*x$のような式はきわめて特徴的であるため，攻撃者の目にとまる恐れがある．このような“不自然さ”の問題は，データ難読化におけるドメイン変換式やデータ復号ルーチンにも当てはまる．防御者は，アンチ逆コンパイルやアンチ逆アセンブル技術を併用することで，攻撃者の解析を妨げたり，できるだけ不自然さのないopaque predicateを作成することが重要となる．なお，難読化ツールを公開した場合は，適当なプログラムを難読化して元のプログラムとの差分を調べることで，ツールが生成するopaque predicateを容易に特定されてしまうであろう．

難読化は，標準ライブラリ関数やAPIの呼び出し，ファイルへのアクセスといった外部環境とのインタフェースを隠蔽できない点に注意する必要がある．たとえば，クラス名やメソッド名を隠蔽するJava難読化ツールは数多く存在するが，プログラム中で使用される基本クラス（rt.jarなどに含まれるもの）やそのメソッドの名前を隠蔽することはできない．呼び出すメソッドを動的に（実行時に）決定するようにプログラムを変換することで静的解析は防げるが，動的解析に対しては効果がない．この問題の完全な解決は難しいが，1つの解決方法とは，あらかじめ難読化された標準ライブラリを自前で用意することである．

☆14 http://www.cs.arizona.edu/sandmark/

難読化の定義と評価

ソフトウェア保護を行う者は，攻撃者から隠蔽したい具体的な秘密情報（定数，アルゴリズム，サブルーチンなど）に見合った方式を採用する必要がある．しかし，これまで提案されてきた難読化法の多くは，「ソフトウェアの解析を困難にする」という漠然とした目的しか述べられておらず，その評価もごく限定的であった．たとえば，制御フロー難読化の主目的はアルゴリズムやサブルーチンの隠蔽であると思われるが，「難読化によってアルゴリズムを推測する手がかりがどの程度減ったか」という観点での評価はほとんど行われていない．

妥当な評価が行われていない原因は，評価がそもそも難しいことである．隠蔽すべき情報は多様であり，攻撃者の取り得る手段も無数にある．また，難読化や解析（攻撃）の妥当な定義を与えることも容易でない．たとえば，Barakら[6]は，プログラムへのオラクルアクセス（任意の入力を与えて出力を観測すること）により得られる以上の情報をプログラムコードが漏らさないとき，そのプログラムはvirtual black boxである（難読化されている）と定義している．ただし，これは難読化の定義というよりはむしろ暗号化の定義に近く，プログラムコードがホワイトノイズといえる状態でない限り，この定義を満たすことはできない．この定義は，既存の難読化法の評価には役立たないといえる．

また，門田ら[3]は，プログラムPと，Pに関する知られたくない命題Qについて，「攻撃者がQを論理的に証明できた」状態を「Qに関してPを解析した」と定義し，「解析により時間がかかるようにPを変換する」ことを難読化の要件としている．小規模なPに対してはループ不変式などを用いて任意の命題Qを数学的に証明できるため，証明の困難さを数値化することで難読化の評価が可能となる．ただし，現実には，攻撃者にQは与えられないため，「Qを証明すること」ではなく，「Qを発見すること」を解析の定義とする方がより現実に即している．この場合には，Qを発見することの困難さが評価できる必要があり，そのためには，Qを発見する手段についての定義，すなわち，攻撃のモデルが必要となる．

評価の手がかりとして利用できる攻撃モデルとしては，Mondenらのモデル[7]がある．このモデルでは，攻撃者の能力を，(A)採用されている保護方式に関する知識，(B)攻撃者がシステムから観測できる情報，(C)攻撃者がシステムに対して行うことのできる行動，の3つのカテゴリについて，それぞれ5つのレベルに分類している．ただし，これは具体的な攻撃目標を含まないモデルであるため，定量的な評価を行うためには，攻撃目標（秘密情報）に対する攻撃者の具体的なアクションを定義することが別途必要となる．

現実的な評価を行うためには，保護の対象や攻撃方法を限定したモデルが必要となる．理想的な評価方法としては，

1. 攻撃者のモデル（攻撃者が行うことのできる行動，行えない行動），および，
2. 攻撃者の目的（攻撃者から隠したい情報と攻撃者のリソースに関する命題．たとえば，サイズnのプログラムから情報yを平均$O(nlogn)$回の試行で取り出すなど）を明確にし，
3. 1.の攻撃者が2.を達成できない（もしくは達成できる）ことの証明，を行うことである．

評価の一例として，松本ら[8]は，隠蔽すべき情報としてプログラム中の復号鍵を対象とし，攻撃者モデルとして，プログラム実行時のスタックメモリから復号鍵を探す手順を提案し，攻撃者が鍵を探せることを実験的に示している．

被験者による実験的評価も行われている．松岡ら[9]は，ソフトウェア技術者を被験者として，難読化されたDES復号プログラムを解析して復号鍵を導出する実験を試みている．また，ソフトウェアにプロテクトを施して公開し，クラックできた者に賞金を出すことで攻撃に対する耐性を評価するという試みや，クラックのエキスパートを雇って評価するという試みも一部では行われている．

難読化の今後の展望

これまで，汎用的な難読化法が数多く提案されてきたが，狙いや効果が不明確になりがちであった．今後は，隠蔽するべき情報やプログラムの種類を限定することで，より高い効果を持つ難読化法を開発していくことが必要とされるだろう．

1つの有力な難読化法は，Chowら[2]が提案した，暗号方式と難読化を融合させた“white-box”暗号システムである．文献2）では「DES復号プログラムに含まれる復号鍵を隠蔽する」ことを目的としている．この方法は，データ難読化の一種であり，復号時に必要となる定数の集合S-Box（データ変換テーブル）と復号鍵を混ぜ合わせ，新たな（1つの復号鍵に特化した）データ変換テーブルを作成する．そして，この変換テーブルを用いて復号を行うようにアルゴリズムを変形する．新たなアルゴリズムは，計算の過程で元の復号鍵が現れないことが保証される．また，作成された変換テーブルから復号鍵を分離することも困難である．近年，このChowらの方法への攻撃方法，および，その改良方法が次々と提案されており，今後目が離せない分野といえる．

プログラム暗号化

プログラムの暗号化は，ハードウェアベースのプロテクション技術の主流となっており，基本的には「プログラムを暗号化して流通させ，実行時に（ハードウェアにより）復号を行う」ものである．復号ルーチンをソフトウェアで実装することで，ソフトウェアのみによる保護が行えることになる．これまで，数多くのプログラム暗号化／実行時復号ツールが開発されている☆8, ☆15．

この方式では，プログラム本体は暗号化されているが，最初に実行される復号ルーチンは暗号化されていない．そのため，復号ルーチンが格好の攻撃対象となる．また，復号したプログラムを置くメモリはそれ以上の攻撃対象となる．熟練した攻撃者は，デバッガやメモリダンプツールによってプログラム実行時の動的解析（メモリの除き見）を行うのが一般的であり，復号後のプログラムを隠蔽することは容易でない．

この動的解析に弱い点が，難読化と比べて暗号化の大きな弱点となっている．いくつかのツールでは，少しでも安全性を高めるために，プログラムを多数の部分に分割（断片化）し各断片を異なる鍵で暗号化／復号する，同じメモリを繰り返し再利用する，復号後のプログラムの再暗号化を行う，といった方法が用いられている．ただし，ハードウェアやOSによって保護されたメモリにプログラムを復号できない限り，弱点を完全に解決することはできない☆16．前述のように，攻撃のためのツールが数多く存在し，攻撃方法が知られすぎているのも，プログラム暗号化の大きな弱点である．

攻撃の検出・応答

ソフトウェアプロテクションでは，攻撃の検出，および，応答の機構も重要である．たとえば，デバッガの動作を検知するルーチンをプログラム中に設置し，デバッガによる解析を妨げることが一般に行われる．この場合，「デバッガ検知ルーチン」の改ざんを防ぐことが別途必要となる．そこで，改ざんを検出するためのルーチンを(攻撃に対する応答ルーチンとともに)ソフトウェア内部に数多く設置することが必須となる．攻撃者は，すべてのルーチンを除去した後でないと，デバッガ検知ルーチンを改ざんして無効化することができなくなる．典型的な応答ルーチンは，ソフトウェアの実行を停止させるものであるが，理想的には，攻撃検知後の応答をできる限り遅らせることが望ましい．とはいえ，これらの技術は十分に強力とは言えないのが現状である．プログラムの暗号化と同様，攻撃ツールや攻撃方法がよく知られているためである．防御者は，現在普及している攻撃ツールの特性をよく理解した上で，防御，検出，応答の機構を実装する必要がある．

むすびにかえて

ソフトウェアプロテクション技術は，この5年間で急速に需要が高まり，数多くの特許が成立している．本稿で紹介した技術のいくつかはすでに特許化されており，誰もが自由に使える技術ではなくなっている．ただし，成立した特許の中には，出願前より公知の事実であったと思われるクレームが含まれているものがある．プロテクション技術を採用する者は，どの技術が特許化されているかを見極める，および，どのクレームが有効であるのかを見極める必要がある☆17．

参考文献

1) Červeň, P. : Crackproof Your Software − The Best Ways to Protect Your Software Against Crackers, No Starch Press, San Francisco, ISBN 1-886411-79-4(2002).
2) Chow, S., Eisen, P., Johnson, H. and Oorschot, P. V. : A White-box DES Implementation for DRM Applications, ACM Workshop on Digital Rights Management (DRM2002), Lecture Notes in Computer Science, Vol.2696, pp.1-15, Springer-Verlag(2003).
3) 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(July 1997).
4) Wang, C., Hill, J., Knight, J. and Davidson, J. : Protection of Software-based Survivability Mechanisms, Proc. International Conference on Dependable Systems and Networks, pp.193-202 (July 2001).
5) Mercaldi, M. A. : Using Exceptions to Obstruct Analysis of Control Flow Structure, Bachelor's Thesis, Harvard College, Cambridge, Massachusetts(Apr. 2002).
6) Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S. and Yang, K. : On the (im)possibility of Obfuscating Programs, Lecture Notes in Computer Science, Vol.2139, pp.1-18, (2001).
7) Monden, A., Monsifrot, A. and Thomborson, C. : A Framework for Obfuscated Interpretation, 2nd Australasian Information Security Workshop (AISW2004), Conferences in Research and Practice in Information Technology, Vol.32, pp.7-16(Jan. 2004).
8) 松本　勉，赤井健一郎，中川豪一，大内　功，竹脇和也，村瀬一郎：Java対応ランダムデータ補足ソフトウェア，情報処理学会論文誌，Vol.44, No.8, pp.1947-1954(Aug. 2003).
9) 松岡　賢，赤井健一郎，松本　勉，竹脇和也：鍵内臓型暗号ソフトウェアの人手による耐タンパー性評価，2002年暗号と情報セキュリティシンポジウム(SCIS2002)(Jan.-Feb. 2002).

(平成17年1月19日受付)

☆15 Programmer's tools, http://www.programmerstools.org/
☆16 Červeň[1] によると，2002年時点での最も強力なツールはASPack Software社のASProtectであり，プログラムの暗号化に加えて数多くのアンチクラック技術が実装されている．一方では，ASProtectに対する攻撃方法が解説されている文献も存在するのだが．
☆17 ただし，特許を侵害していることの立証もまた容易でないため，特許の実質的な価値は不明な面もある．