RSAアルゴリズム鍵生成モジュールの実装問題(ROCA)

2.1 はじめに

2017年10月、ACM CCS 2017[*1]の論文発表予定をトリガーとして、暗号技術に関連する大きな報道がありました。1つはKRACKsと呼ばれるWPA/WPA2の仕様上の問題に起因する攻撃の報道です[*2]。プロトコルそのものの問題であったことから、大きな問題になると予想され、SNSを通して過剰な反応が見られましたが、比較的軽微な修正で問題をフィックスできたため、少々肩透かしを食らう事例でした。この一件はJNSAセキュリティ十大ニュース[*3]の第4位にランクインするなど、不確かな情報流通や報道の在り方について一石を投じるものとなりました。

もう1つはROCA(The Return of Coppersmith's Attack)と呼ばれるRSA暗号アルゴリズムにおける鍵生成モジュールの実装不備の問題[*4]です。ROCAはKRACKs攻撃とほぼ同じ時期に報道されたにもかかわらず、国内では大きく取り上げられていません。一方で、より現実的な時間とコストでRSA公開鍵の素因数分解が可能になる攻撃であったことから、速やかに対策が行われました。本来持つべき暗号機能が十分に発揮されないまま機能低下を引き起こす攻撃であると認識されたため出荷ベンダーにより修正パッチが展開され、かつ脆弱な鍵生成モジュールで作成されたRSA鍵ペアの更新が促されています。プログラムのバグもしくは設計の不具合により、本来持つべきもしくは持っていると考えられていた暗号機能の実装がはるかに破られやすくなっている事例[*5]はこれまでにも露呈しており、ROCAもその1つとしてリストされたことになります。本フォーカスリサーチでは、過去の失敗事例を紹介しながら、ROCAと同様に鍵や各種パラメータの空間を狭めることで生じる暗号実装の脆弱性について、その要因を整理・俯瞰していきます。また、ROCAを含む一連の研究結果が今後どのような影響を及ぼすかについても触れます。

2.2 ROCAの概要

SSL/TLSなどのセキュリティプロトコルを利用する際には暗号技術が使われています。鍵マークが表示されていることで安全であることを一般ユーザが確認できるブラウザなどのアプリケーションにおいて、暗号化(機密性の確保)とデジタル署名(完全性の確保)を目的に公開鍵暗号方式が使われています。その1つであるRSA暗号方式は素因数分解の難しさを安全性の根拠に置いた方式であり、サーバ証明書の多くはRSA公開鍵が格納されており、例えばSSL/TLSにおいてサーバの確からしさを保証する仕組みとして広く流通しています。最近ではPerfect Forward Secrecy[*6]の観点からサーバ証明書に格納されている公開鍵を機密性確保の用途に用いずに、その都度DHやECDHアルゴリズムでEphemeral鍵(一時鍵)を生成して暗号化を行う方法が推奨されています。そのためブラウザもしくはユーザがサーバの確からしさを確認する際には、署名専用のアルゴリズムも利用可能になっています。その代表例としては楕円曲線暗号をベースとしたECDSA署名[*7]があります。実際RSA公開鍵ではなくECDSA鍵を含むように発行されたサーバ証明書の割合が増えており、主要ブラウザでもサポートされています。一方で、現在でもRSAベースのサーバ証明書が広く利用されており、ROCAの影響を受けることとなります。

2017年10月、チェコのMasaryk大学の研究チームによってInfineon Technologies AG製のRSA鍵生成モジュールの脆弱性とその影響が報告されました。RSAアルゴリズムは鍵生成時には2つの素数を生成してそれらを秘密鍵とし、2つの素数をかけ合わせた合成数を公開鍵とする暗号方式です。公開鍵である合成数を素因数分解するために要する計算量が膨大で解読が現実的でないことで安全性を担保しています。今回RSA鍵生成モジュールにおいて実装上の脆弱性が発見され、生成される鍵に偏りがあることを利用すれば、想定されるよりもはるかに短い

＊1 ACM Conference on Computer and Communications Security 2017(https://ccs2017.sigsac.org)。CCS 2017 - Accepted Papers(https://acmccs.github.io/papers/)。
＊2 Key Reinstallation Attack(https://www. krackattacks.com)。
＊3 JNSA、「セキュリティ十大ニュース」(http://www.jnsa.org/active/news10/)。
＊4 Centre for Research on Cryptography and Security(CRoCS), Masaryk University、"ROCA:Vulnerable RSA generation"(CVE-2017-15361)(https://crocs.fi.muni.cz/public/papers/rsa_ccs17)。
＊5 Internet Infrastructure Review vol.17「1.4.1 SSL/TLS、SSHで利用されている公開鍵の多くが他のサイトと秘密鍵を共有している問題」(https://www.iij.ad.jp/dev/report/iir/017.html)。
＊6 Internet Infrastructure Review vol.22「1.4.2 Forward Secrecy」(https://www.iij.ad.jp/dev/report/iir/022/01_04.html)。
＊7 NIST、"FIPS 186-4 Digital Signature Standard(DSS)"(https://csrc.nist.gov/publications/detail/fips/186/4/final)。6章にてECDSAが規定されている。同文書は4章にDSA、5章にRSAによる署名方式も記載がある。

時間で素因数分解が可能になることが報告されています。本来よりも狭い空間から鍵やパラメータを導出するために脆弱だと認識された事例がこれまでにもいくつか報告されていますが、その多くが疑似乱数生成モジュールの不具合によるものでした。今回も研究結果だけを見ると同様の問題としてカテゴライズされかねませんが、問題の本質は擬似乱数生成部にはなく、実際には素数生成に関わる実装部分の不具合にありました。

2.3 鍵のライフサイクルと過去の失敗事例

公開鍵暗号方式では、デジタル署名や復号時に使用される秘密鍵とサーバ証明書や鍵リポジトリを通じて公にされる公開鍵の2種類の鍵が利用されます。そのため利用者はまず初めにそれぞれの暗号方式で規定された鍵ペアを生成しますが、その際には擬似乱数生成モジュールから安全に利用できる乱数列をソースとして鍵ペアが生成されます。この疑似乱数生成モジュールは鍵生成時だけではなく、鍵利用時(例えば署名時)に必要に応じて参照され乱数列を得ることができるように配備されています。公開鍵を用いた暗号化処理や秘密鍵を用いた署名処理、その対としての署名検証などが行われる鍵利用の際には公開鍵に有効期限が設定されることが多く、期限切れのあとは廃棄され新たな鍵を生成するという一連のライフサイクルが存在します。この鍵管理フローの中には、呼応する秘密鍵が漏えいまたはその可能性が生じた時点で、有効期限内であっても鍵を強制的に無効にするといったパスも用意されます。SSL/TLSにおけるサーバ証明書では有効期限が設けられており、その前に証明書を廃棄する仕組みを考えると、これらの一連のライフサイクルを理解することができるかと思います。

次に、前述の鍵管理ライフサイクルにおいて、どの段階で問題が生じたかを理解するためにいくつかの失敗事例を見ていきます。今回と同様にRSA鍵生成時に問題が生じていた事例の1つとして、2008年に露呈したDebian OpenSSLにおける鍵生成問題が挙げられます[*8]。Debianの特定バージョンにおけるOpenSSLを使って鍵生成を行った場合、極端に少ない鍵空間からしか秘密鍵を導出していないというバグが生じていたため、脆弱な鍵生成モジュールから生成しうる公開鍵リストが公開され、チェックできるような体制が取られました。

同様に鍵生成の問題としては、台湾市民カードの不具合[*9]が2013年に指摘されています。FIPS140-2と呼ばれる暗号モジュールに対する認定基準をクリアしたICカードでしたが、生成される素数に大きな偏りが見られ、素因数分解が可能な公開鍵が生成されている例が報告されています。これはROCAとは異なり疑似乱数生成モジュールの不具合としてカテゴライズされています。ここで報告された脆弱な鍵の中には2012年に報告された、意図せず秘密鍵を共有してしまう問題[*10]の一例として複数含まれており、はるかに少ない計算量で素因数分解が可能となっていました。意図せず秘密鍵を共有してしまう事例は、生成される鍵空間が少ないことに起因しており、あるICカードではたった36通りのRSA公開鍵しか生成できない実装があるなど、非常にインパクトの大きい報告がなされていました[*11][*12]。

また、鍵生成時ではなく鍵利用時における同様の失敗事例もありました。ビットコインウォレット機能を持つAndroidアプリにおいてECDSA署名に用いられるパラメータを再利用したために同一エンティティによる2つの署名から秘密鍵が同定されてしまう事故です[*13]。署名アルゴリズムとしてパラメータを使い回ししてはいけないという制約条件を無視した実装で、具体的にはAndroidアプリが利用する擬似乱数生成モジュールのエントロピーが低いために当該パラメータが偶然重なってしまったことが原因でした。このように様々なフェーズにおいてROCAと同様の問題が起こっていることが分かります。

---

*8 JVNVU#925211、「DebianおよびUbuntuのOpenSSLパッケージに予測可能な乱数が生成される脆弱性」(http://jvn.jp/vu/JVNVU925211/)。
*9 Daniel J. Bernstein et.al, Factoring RSA keys from certifed smart cards:Coppersmith in the wild、"Cryptology ePrint Archive:Report 2013/599" (https://eprint.iacr.org/2013/599)。
*10 PKIDay2012 須賀、「公開鍵の多くが意図せず他のサイトと秘密鍵を共有している問題」、PKI Day 2012講演資料(http://www.jnsa.org/seminar/pki-day/2012/data/PM02_suga.pdf)。
*11 Arjen K. Lenstra et al., Ron was wrong、"Whit is right"(https://eprint.iacr.org/2012/064)。
*12 Nadia Heninger et al., Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices、"Proceedings of the 21st USENIX Security Symposium"(https://factorable.net/paper.html)。
*13 Bitcoin Project、"Android Security Vulnerability"(https://bitcoin.org/en/alert/2013-08-11-android)。

2.4 ROCAにおける問題の本質

Infineon Technologies AG製の暗号ライブラリで見つかったこの問題は、生成される鍵空間の狭さが原因という観点からいえば、Debian OpenSSLなどと同じ要因であるとも言えます。しかし特筆すべきは、これが疑似乱数生成モジュールから吐き出されるランダムデータに偏りがあることで起こるのではなく、鍵生成アルゴリズムに問題があるという点にあります。

報告者らによると、脆弱性の見つかった鍵生成モジュールはソースコードのレビューやリバースエンジニアリングによる発見ではないと主張されています。鍵生成モジュールをブラックボックスとして扱い、複数の鍵ペアを生成させて大量の鍵を取得した後、鍵データの偏り（バイアス）を観測することで取りうる鍵空間が狭いことが明らかになりました。RSA の公開鍵は2つの素数の積で得られます。一般的な鍵生成モジュールの処理はランダムデータから素数候補となる奇数を生成した後、その数が素数であるかどうかを判定する素数判定部を有しています。一般に素数判定は時間を要することから、演算速度もメモリ空間も制約されたICカードなどの環境下で鍵生成を行った場合にはボトルネックとなる可能性があります。今回見つかった脆弱な鍵生成モジュールで発生される素数は特徴的であり、すべての素数空間と比較するとはるかに小さい空間からしかピックアップしていないことが分かっています。発見者も論文中で指摘しているとおり、この特徴的な素数の形式に制約があるのは素数生成を高速化する意図があったと見られます。設計者や実装者はこのような高速化を施すことが結果的に脆弱になってしまうことに気がついていなかったと考えられます。レスポンス時間に対する要求事項のレベルが高く、処理目標よりもはるかに性能が低かったために、本来行うべき処理を省略してしまったとも想像できます。

同じような問題としてはAndroidアプリにおいてバックグラウンドでSSL/TLS通信を行う際に証明書検証を省いているという脆弱性報告が毎年数十件のレベルで報告され続けているという事実もあります。一般的なブラウザであればSSL/TLSサーバと通信する際の証明書検証結果をURL入力エリアやセキュリティインディケータを通じてユーザに知らせていますが、Androidアプリにてバックグラウンドで行われているSSL/TLS通信においてはそのような表示やユーザアクションを必ずしも必要としていないため、証明書検証モジュールを省略して高速化を図るという選択をしていると考えられます。

ROCAで指摘された脆弱性を持つモジュールで生成される素数pは以下のような特徴を持つことが判明しています。

$$p = k*M + (65537^{a} \bmod M)$$

ここでMは2から連続するn個の素数の積であり、生成したい素数の長さによって定められます（256ビットの場合にはn=39、512ビットの場合にはn=71など）。nが固定されるとMが自動的に固定されるためパラメータkとaを適当に動かすことで素数候補を選択していくことになります。例えばRSA-512公開鍵を生成するためには256ビットの素数を2つ掛け合わせることで実現されますが、どのくらいの密度で素数が存在するかを示す素数定理によると$2^{248.5}$個程度の候補があることが知られており、非常に大きな素数空間からたった1つの素数をランダムに選択することができることが分かります。一方で今回の脆弱な素数生成モジュールではn=39つまりM = 2*3*5*...*167は約219ビット長であるためパラメータkとしては37ビット分しか可動せず、パラメータaも62ビット分しか選択できないため結果的に99ビット分しかエントロピーを持っていない、つまり本来よりもかなり狭い鍵空間からしか素数生成していないことが分かります。

RSAは素因数分解の困難性を安全の根拠とするアルゴリズムであり、NISTによりどのくらいのRSA鍵長を利用することが共通鍵暗号での何ビット鍵による暗号化に匹敵するかどうかが見積もられています。例えばRSA2048は112ビット安全性を保有するなどの対応表が記載されており、先に挙げたECDSAなど楕円曲線暗号では鍵長の丁度半分のビットセキュリティを確保するとされています[*14]。2010年に768ビットRSA鍵が素因数分解されるなど、現在は2048ビット長以上の

*14 NIST、SP 800-57 Part 1 Rev. 4、"Recommendation for Key Management, Part 1：General"（https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final）。

RSA鍵を利用することが推奨されており、PKIでのRSA署名利用の際には1024ビット鍵や昨年コリジョンが見つかり脆弱なアルゴリズムとして認識されるようになったハッシュ関数SHA-1[*15]を利用しないよう移行指針がCA Browser Forumなどで規定され、証明書発行の現場では実際にそのような運用が実施されています。CRYPTRECでは毎年公開される報告書[*16]にスーパーコンピュータの持つ能力との比較が記載されており、現時点では2048ビットRSAが十分に安全であることが裏付けられています。

2.5 ROCAの影響

前節で述べたように、素数に制約があることで鍵空間が大幅に狭まっているために素因数分解を行う探索空間が狭まることは容易に理解できます。同じように秘密鍵の制約条件を用いることで素因数分解を容易にする研究があり、ROCAのタイトルにもなったCoppersmithによる方式[*17]がよく知られています。Coppersmithアルゴリズムは RSAで用いられる2素数p、qの積N(=p*q)と片方の素数pの下半分データからpを効率的に復元し結果的に素因数分解を成功させることができます。OpenSSLにおけるHeartBleedバグの脅威の度合いを推し量るために行われたコンペティションで、効率的にSSL/TLSサーバの秘密鍵を導出した手法の1つでもあります[*18]。

表-1 素因数分解に必要なクラウドリソース利用時のコスト

RSA鍵長	素因数分解に必要なCPUリソース	クラウド利用による素因数分解コスト
512 bit	1.93 CPU hours	$0.06
1024 bit	97.1 CPU days	$40〜$80
2048 bit	140.8 CPU years	$20,000〜$40,000

Coppersmithによる方式はそれをROCA向けに拡張したことにより、素因数分解に必要なクラウドリソース利用時のコストを表-1のように見積もっています。これによると現在広く利用されている2048ビットRSA鍵でも現実的なコストで解読可能であることが分かります。この結果の発表を受けてわずか1週間後に5-25%程度効率よく素因数分解ができることが示唆されています。これは原論文の見積りよりも容易にかつ安価に素因数分解が可能であることを意味しています[*19]。

今回問題となった暗号モジュールで生成された鍵であるかどうかをチェックするツールが著者らによって公開されています。その中にはオフラインでチェックできるPythonコード[*20]のほか、公開鍵をブラウザからポストすることでチェックできるオンライン版[*21]やS/MIME署名をメール送信することで結果を得るなど、様々な検証手段が用意されています。ROCA脆弱性チェックの仕組みは非常に単純で簡潔に記載されています[*22]。著者らによると、誤検知はなく2^{-154}の確率で偶然ROCA脆弱な鍵が生成できてしまうと見積もられており、これは無視できるほど小さいものです。

エストニア政府によって発行されたe-Residency IDカード[*23]で利用される証明書はROCAの影響を受けることがアナウンスされており[*24]利用者に対してファームウェアのアップデートと鍵の再生成を促しています[*25][*26]。原論文によれば、調査対象としたe-Residency IDカードは4400程度ありましたが、そのすべてで影響を受けることが示されています。更にROCAは既に2012年からエンバグしており、過去に遡って調査すると現在は有効期限切れではあるものの、当時から素因数分解可能で

＊15 CRYPTREC暗号技術ガイドライン(SHA-1)改定版(https://www.cryptrec.go.jp/topics/cryptrec_20180427_eval_gl_2001_2013r1.html)。Security Diary、SHAttered attack(SHA-1コリジョン発見)(https://sect.iij.ad.jp/d/2017/02/271993.html)。
＊16 CRYPTREC報告書(https://www.cryptrec.go.jp/report.html)。
＊17 Don Coppersmith, "Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known", EUROCRYPT96, 178–189.
＊18 IIJ-SECT Security Diary、「Heartbleed bugによる秘密鍵漏洩の現実性について」(https://sect.iij.ad.jp/d/2014/04/159520.html)。
＊19 The cr.yp.to blog、"2017.11.05:Reconstructing ROCA"(https://blog.cr.yp.to/20171105-infineon.html)。
＊20 ROCA:Infineon RSA key vulnerability(https://github.com/crocs-muni/roca/)。
＊21 ROCA Vulnerability Test Suite(https://keychest.net/roca)、Test your RSA Keys(https://keytester.cryptosense.com)。
＊22 Key fingerprinting(https://github.com/crocs-muni/roca/blob/master/roca/detect.py)。
＊23 Republic of Estonia、"e-Residency"(https://e-resident.gov.ee/become-an-e-resident/)。
＊24 Politsei.ja Piirivalveamet、"Possible Security Vulnerability Detected in the Estonian ID-card Chip"(https://www2.politsei.ee/en/uudised/uudis.dot?id=785151)。
＊25 Politsei.ja Piirivalveamet、"For the user of ID-card and mobile ID"(https://www2.politsei.ee/en/nouanded/isikut-toendavad-dokumendid/id-kaardi-ja-mobiil-id-kasutajale.dot)。
＊26 Politsei.ja Piirivalveamet、"Renewal of document certificates-frequently asked questions"(https://www2.politsei.ee/en/teenused/isikut-toendavad-dokumendid/sertifikaatide-uuendamine/)。

あったにもかかわらず、その脆弱性を知らずに5年間以上使い続けられていた鍵が存在していたと考えられます。日本のベンダーも含め、TPM（Trusted Platform Module）が搭載されている製品群について各社が詳細な情報を提供しています[*27][*28]。推奨されるアクションはファームウェアのアップデートと共に、脆弱なTPMチップで生成されたRSA鍵ペアを廃棄し、新たに生成し直す作業が求められました。TPMは耐タンパー性を持つチップで、秘密鍵への物理的な攻撃を防いでおり、メモリなど記憶装置に鍵データが格納されるよりも安全であるとされてきました。しかし今回のROCAの事例が発覚したことで、鍵を使うモジュールをブラックボックス化することによるデメリットが新たに発生しうることが露呈しました。管理が大変な部分をアウトソースして管理下から追いやる一方で、コントールできないところで新たな脅威が生まれる可能性があるとも言えます。

2.6 ROCA発見に致るまでの経緯

2.4節で見たようにROCAは特殊な形式を吐き出す素数生成モジュールに着目できたことで脆弱性の発見に至りました。リバースエンジニアリングも行わずソースコードにもアクセスせずに大量の素数を観測することで偏りを発見したそのプロセスは過去の同チームによる研究結果がベースとなっています。2016年8月に開催されたUSENIX Security 2016にて38種類の暗号ソフトウエアやICカードで生成させたRSA鍵に関する考察がそれにあたります[*29]。素数p、qの最上位2から8ビット目までの7ビット分の出現状況を示すp、q 2軸のヒートマップから各暗号ライブラリごとに特徴があることが露呈しました。更に素数の最上位2から7ビット目、素数の最下位2ビット目、素数mod 3、公開鍵N mod2という全部で9ビットの最も特徴的と考えられる部分のみを抽出して傾向を調べることで38種類からのモジュールを13カテゴリに分類しています。ROCAの標的となったInfineon製のモジュールはクラス12に分類されており、他のカテゴリに属する暗号ライブラリと比べ突出した特徴を持つことがこの時点で指摘されていました。

更にACM CCSのあとに開催されたACSAC2017でも同じ研究チームから関連する研究結果が発表されています[*30]。これまでの暗号ライブラリに関する知見を生かして公開鍵情報からのみでその公開鍵が生成された暗号ライブラリを特定するという試みです。USENIX Security 2016でも同様のアプローチがありましたが公開鍵Nのmod3、mod4の剰余を計算して偏りがないかを検出する方式を取ることで暗号モジュールを同定しています。研究者らによると誤検知は1パーセント以下であると主張しており、例えばTorで用いられる公開鍵情報を観測することで同じユーザや地域のノードであることが露呈するなどのプライバシの影響も指摘されています。

2.7 ROCAが他の暗号アルゴリズムに影響する可能性

ROCAそのものについてはRSA暗号方式における鍵生成モジュールの脆弱性、つまり素数生成ロジックの問題であるため、素数を生成・利用はするものの秘密裏に保持する必要のないRSA以外の暗号アルゴリズムに影響を及ぼす可能性は低いと思われます。RSA暗号方式では秘密鍵の候補として、例えばRSA2048ビット公開鍵を生成する際には1024ビット長の素数が2つ必要となります。一方で、ビットコインで利用されている前述したECDSA署名では、署名に用いられる秘密鍵は整数（256ビット長）を導出するのみで生成できるため複雑なロジックを必要としません。つまり擬似乱数生成モジュールの確からしさのみが鍵生成モジュールの安全性に影響を及ぼすこととなり、ROCAと同じような脆弱性が横展開される可能性は低いと言えます。

鍵のライフサイクルにおいて鍵生成ではなく鍵利用フェーズを考えます。このとき先に挙げたようなAndroidにおける擬似乱数生成モジュール実装の問題で見たように、本来ならば毎回異なるパラメータを生成して署名する必要がありますが、ここにも素数生成などに見られるような特殊なロジックは必要とされ

＊27 Infineon Technologies AG、"Information on TPM firmware update for Microsoft Windows systems as announced on Microsoft's patchday on October 10th 2017"（https://www.infineon.com/cms/en/product/promopages/tpm-update/?redirId=59160）。
＊28 CERT、"Vulnerability Note VU#307015, Infineon RSA library does not properly generate RSA key pairs"（https://www.kb.cert.org/vuls/id/307015）。
＊29 Centre for Research on Cryptography and Security（CRoCS）, Masaryk University、"The Million-Key Question - Investigating the Origins of RSA Public Keys [Usenix Sec 2016, Best Paper Award]"（https://crocs.fi.muni.cz/public/papers/usenix2016）。
＊30 Centre for Research on Cryptography and Security（CRoCS）, Masaryk University、"Measuring Popularity of Cryptographic Libraries in Internet-Wide Scans [ACSAC 2017]"（https://crocs.fi.muni.cz/public/papers/acsac2017）。

ていません。ただし秘密鍵にせよ各種パラメータにせよ、必要とされているだけのランダムデータを導出せずに先頭がゼロで埋め尽くされているケースや、短いビット列を繰り返し埋めることでデータを確保するケースなど、ECDSA署名アルゴリズムにおいても秘密鍵空間の狭さからくる脆弱性の存在は無視できません。このとき、ビットコインの公開鍵情報は過去のブロックチェーンを参照することによって同じ鍵ペアを導出していないかチェックすることができます。つまり、ほかの第3者と秘密鍵を共有してしまっているか判断することができます。このことからも、あたかも正しい手順で鍵生成を行っているように見えますが、実際には鍵空間が狭められているというバックドアが仕掛けられている実装が存在しうることが想像できます。

ROCAと同じように意図せず鍵生成モジュールが脆弱であることが露呈した場合には、鍵生成を何度も繰り返し行うことで想定されるよりも高い確率で偶然他の誰かが所有するものと同じ鍵ペアを導出する攻撃が考えられます。この場合、コールドウォレットと呼ばれる署名鍵や署名モジュールをネットワークに繋がないことで安全を確保する技術に対しては効果がありません。このようにビットコインに限らず仮想通貨における鍵生成フェーズはとても重要であることが分かります。前述した台湾市民カードの例で見たように、FIPS140-2などによってお墨付きを得た暗号ライブラリやHSM(Hardware Security Module)であっても、脆弱な製品が存在しうることを想定した鍵管理の運用が求められます。ビットコインでは所有者IDとして用いられるビットコインアドレスが存在します。ビットコインアドレスはSecp256k1で識別される楕円曲線上のECDSA署名方式において鍵ペア生成を行い、公開鍵データから2つのハッシュ関数を用いてダイジェストを算出した後でバージョン情報やチェックサム用データを付与し、Base58符号化を用いることで最終的に26〜35文字の可読文字に変換するという手順で生成されます。この手順において、ビットコインアドレスにユーザの指定する文字が出現するように「お好みのアドレス」を導出する方法が知られています。ここで用いられるハッシュ関数はSHA-2とRIPEMD160であり、その出力データを意図したように導出することは困難であることから、異なる鍵ペアを何度も試行錯誤しながら導出することになります。この方式において相当な回数の鍵ペアを生成することから、ここでも安全なランダムデータが必要となります。このような生成ツールが多く出回っていますが、これらを信用する手立てはなく、特にソースコードではなくバイナリで配布されているケースもあるため、利用する際には注意が必要です。

今回RSAアルゴリズムが実装された暗号モジュールの脆弱性と今後起こりうるプライバシ問題を取り上げました。素数生成、素数判定という少々難解なロジックを要するために実装上のバグが入り込みやすい暗号方式であることが再認識されましたが、RSAアルゴリズム自体にはほとんど傷がついておらずこれまでの実装に関する知見が共有されていれば安全に利用することができます。量子コンピュータが出現することで素因数分解が容易となり今後利用できなくなるという予想もあり耐量子暗号と呼ばれる次世代暗号も開発されるようになりました[*31]。NISTでは現在標準化のためのコンペティションが行われており今後3-5年程度の暗号解析と検討ののち2年程度で標準化ドラフトが共有されるスケジュールで進められる見込みです[*32]。次世代の暗号技術を実装する際には、設計者・実装者にとって更に複雑と考えられる仕組みや理解のために膨大な知識が必要な状況が考えられます。RSAなどの現代暗号で起こったROCAのような問題の本質を理解することで次の世代への教訓として受け継いでいくことが期待されます。

執筆者:
須賀 祐治(すが ゆうじ)
IIJ セキュリティ本部 セキュリティ統括室 シニアエンジニア。
2008年7月より現職。暗号と情報セキュリティ全般に関わる調査・研究活動に従事。CRYPTREC暗号技術活用委員会 委員。
暗号プロトコル評価技術コンソーシアム 幹事。電子情報通信学会 ISEC研究会 幹事補佐。IWSEC2018 Organizing committee member。
ECC2018 Organizing committee member。Virtual Currency Governance Task Force(VCGTF) Security WG member。

*31 Internet Infrastructure Review vol.31「1.4.3耐量子暗号の動向」(https://www.iij.ad.jp/dev/report/iir/031/01_04.html)。
*32 Dustin Moody、"THE SHIP HAS SAILED The NIST Post-Quantum Crypto "Competition""(https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/asiacrypt-2017-moody-pqc.pdf)。