TLSの動向

最近、IETF（Internet Engineering Task Force）ではTLS（Transport Layer Security）が盛んに議論されています。議論が活発になった理由は、何と言っても2013年にエドワード・スノーデン氏によって暴露されたアメリカ政府によるネット・電話の極秘監視・情報収集プログラムPRISMでしょう。PRISMに代表される広域監視（pervasive monitoring）の存在が明らかとなり、IETFはプライバシの問題を再考する必要にせまられました。RFC 7258で宣言されているように、今後IETFで策定されるプロトコルは、広域監視をやりにくくするよう設計されることになります。

HTTPに関して言えば、TLSの利用が強く推奨されていくことになるでしょう。事実、2015年に策定されたHTTP/2（RFC 7540）を利用する場合、主要ブラウザがTLSを要求するため、TLSの利用が実質的に必須となっています。もちろん、現在の主流であるHTTP/1.1でも、TLSの利用が強く推奨されています。TLSを利用するにはHTTPサーバに証明書が必要です。これまで証明書の発行は有償であり、そのせいでTLSの利用を思いとどまった方も多いでしょう。現在では、Let's Encryptというプロジェクトによって証明証を無償で発行してもらうこともできます。

TLSの最新のバージョンは1.2であり、策定後8年が経過しています。この間、TLSに対する様々な攻撃手法が発見されました。RFC 7457は、攻撃手法をまとめた素晴らしい文章です。新たな攻撃手法や暗号技術の老朽化に伴い、推奨されるTLSの利用方法も変わってきました。現時点でのお薦めの利用方法は、RFC 7525でまとめられています。これらの知見をもとに、現在IETFでTLS 1.3の策定が進められています。この記事では、TLSの仕組みを知っている方を対象に、TLSの動向について説明します。

バージョン	仕様	制定年	利用
SSL 2.0	ID 止まり	1995年	RFC 6176により利用禁止
SSL 3.0	RFC 6101	1996年 （RFC発行は2011年）	RFC 7568により利用禁止
TLS 1.0	RFC 2246	1999年	△
TLS 1.1	RFC 4346	2006年	△
TLS 1.2	RFC 5246	2008年	○
TLS 1.3	ID	策定中	

表-1 SSL/TLSのバージョン

3.1 バージョン

TLSの前身はNetscape Communications社が世に送り出したSSL（Secure Socket Layer）です。1995年にSSL 2.0、1996年にSSL 3.0の仕様が公開されました。SSL 2.0は設計上の様々な問題があり、RFC 6176により利用が禁止されました。SSL 3.0も、脆弱性POODLEに代表される攻撃や設計上の問題のため、RFC 7568により使用しないよう求められています。SSLはIETFに持ち込まれて標準化されTLSとなりました。策定されたTLSのバージョンは、1.0、1.1、1.2です。詳しくは後述しますが、現在ではデータの認証と暗号化のためにはAEAD（Authenticated Encryption with Associated Data）という方式を使うことが推奨されています。AEADは、TLS 1.0と1.1では使用できません。端的に言うと、現在TLSを安全に使うには、TLS 1.2を適切な利用方法で使用する必要があります。

SSL/TLSのバージョンに関する情報を表-1にまとめます（IDはInternet-Draftの略記です）。今回、TLS 1.3についても説明しますが、仕様は現在策定中ですので、最終的には少し変わるかもしれないことをご了承ください。

3.2 適切な暗号スイート

TLS 1.2を定めたRFC 5246で実装が必須とされている暗号スイートは、TLS_RSA_WITH_AES_128_CBC_SHAです。これは以下のような意味です。

- 鍵交換はRSA
- サーバ認証もRSA
- 通信の暗号化はAESのCBCモード
- MACを生成する関数がSHA1

HTTP/2で必須とされ、RFC 7525で第一候補にすべきとされているTLS 1.2の暗号スイートは、TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256です。これは、以下のように解釈します。

- 鍵交換は使い捨て楕円Diffie Hellman（ECDHE：Elliptic Curve Diffie Hellman, Ephemeral）
- サーバ認証はRSA
- 通信の暗号化はAES 128のGCM（Galois/Counter Model）モード
- ハッシュ関数がSHA256

TLS 1.3では、TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256に加えてTLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256の実装も必須としています。サーバ認証が、RSAからECDSA（楕円暗号を用いたDSA）に変わっただけですね。

次に、このように推奨される暗号スイートが変化した理由を説明します。

3.3 公開鍵暗号と鍵交換

前述のように推奨される鍵交換の方法は、RSAから使い捨てDiffie Hellman系に変わりました。理由は、RSAが持たない前方秘匿性（forward secrecy）を使い捨てDiffie Hellman系が持つからです。前方秘匿性とは、将来に渡ってデータの秘匿性が守られることです。

鍵交換にRSAを用いると、前方秘匿性がない理由を考えましょう。クライアントがサーバにTLSで接続すると、サーバはクライアントに証明書を送ります。証明書とは、認証局の署名が付いたサーバの公開鍵のことです。クライアントは、サーバと共有すべき秘密を生成し、サーバのRSA公開鍵で暗号化してサーバに送ります。この暗号文を復号できるのは、RSA秘密鍵をもっているサーバのみです。これで、クライアントとサーバは秘密が共有できましたので、この秘密から生成される鍵を使い、共通鍵暗号で通信路を保護します。

この時点では、この暗号路は安全です。第三者が中身を盗聴することはほとんど不可能です。しかし、次のようなことが現実に起こると、盗聴されてしまいます。

ある広域監視が、この暗号路を流れるデータをすべて保存しています。そして、サーバの入れ替えに伴い、サーバを破棄する際に誤ってディスクの内容を消去しませんでした。広域監視がこのディスクを手に入れれば、秘密鍵が取り出せるので、保存していた暗号路のデータを順に復号できます。

一方、使い捨てDiffie Hellman系では、クライアントとサーバが、お互いに使い捨ての公開鍵と秘密鍵を生成します。これらの秘密鍵はディスクに保存されることはないので、前述のようなことは起こりません。

使い捨てDiffie Hellman系では、オリジナルのDHE（Diffie Hellman, Ephemeral）よりも楕円暗号で実現したECDHE（RFC 4492）の方が普及しそうです。その理由は、次のとおりです。

- DHEに比べて、ECDHEの方が交換するデータの量が少ない。
- DHEに比べて、ECDHEの方が計算速度が速い。
- ECDHEには、厳選されたパラメータがあらかじめ定義されている。DHEでも定義しようとするIDはあるが、まだRFCとなっていない。
- 前述のようにRFC 7525で第一候補だと定められている。

前方秘匿性については、「IIR Vol.22、1.4.2 Forward Secrecy」も参考にしてください。

3.4 共通鍵暗号の老朽化

TLS 1.1以前では、暗号文の形式は次の2つがあります。

- ストリーム暗号
- CBC（Cipher Block Chaining）モードのブロック暗号

ストリーム暗号として実質上唯一の選択肢であるRC4には、様々な攻撃方法が見つかっており、利用が禁止されています（RFC 7465）。

TLS 1.0以前のCBCモードのブロック暗号に関しては、BEASTという攻撃方法が有名です。また、TLS 1.2以前のCBCモードのブロック暗号は、「MAC後暗号化」という手法を取っています。MAC（Message Authentication Code）とは、データが改ざんされてないことを保証したり、認証したりするための補助データです。「MAC後暗号化」では、平文からMACを生成し、平文とMACを連結した後に全体を暗号化します。「MAC後暗号化」には、パディングオラクル攻撃という攻撃手法が見つかっています。そのためRFC 7366では、「MAC後暗号化」の代わりに「暗号化後MAC」という書式を提案しています。

TLS 1.2では暗号文の第3の形式として、AEAD（Authenticated Encryption with Associated Data）が定められました。AEADとは、暗号化と認証を同時に実行する方式です。現在では、ストリーム暗号とCBCモードのブロック暗号ではなく、AEADを利

用することが推奨されています。AEADで利用できる共通鍵暗号のモードは、次のとおりです。

- AESのGCM（Galois/Counter Model）モード
- AESのCCM（Counter with CBC-MAC）モード

TLS 1.3では、ストリーム暗号やCBCモードのブロック暗号の書式は削られ、AEADのみが定義されています。

3.5 ハンドシェイク

この節では、TLSの実際のやりとりについて説明します。

3.5.1 フルハンドシェイク

クライアントがサーバに初めて接続するときは、フルハンドシェイクをする必要があります。TLS 1.2でTLS_RSA_WITH_AES_128_CBC_SHAが選ばれると、図-1のようなやりとりになります。

- クライアントは、ClientHelloで暗号スイートの候補を提示します。
- サーバは、TLS_RSA_WITH_AES_128_CBC_SHAを選んだことをServerHelloで伝えます。Certificateには、サーバのRSA証明書が入っています。
- クライアントは、秘密を生成、サーバのRSA公開鍵で暗号化し、ClientKeyExchangeに格納して送ります。その後ChangeCipherSpecを送って、通信路を暗号路に切り替えます。この暗号路は、AESのCBCモードで暗号化されます。暗号路に切り替えた直後に、ハンドシェイクがうまくいった証拠としてFinishedを送ります。また、今後アプリケーションから受け取ったデータも、この暗号路を用いて送られます。図-1の灰色は、暗号路を表現しています。
- サーバは、サーバの秘密鍵を使って秘密を取り出し、クライアントと同様にChangeCipherSpecを送って、通信路を暗号路に切り替えます。

次に、TLS 1.2でTLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256が選ばれたときを説明します（図-2）。TLS_RSA_WITH_AES_128_CBC_SHAの場合と異なる点は、次のとおりです。

- ClientHelloを受け取ったサーバは、TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256を選択します。そして、ECDHEの使い捨て公開鍵と秘密鍵を生成します。この公開鍵は、ServerKeyExchangeの中に入れて送ります。
- クライアントも、ECDHEの使い捨て公開鍵と秘密鍵を生成します。自分の秘密鍵とサーバの公開鍵から、秘密を生成します。ClientKeyExchangeには、自分の公開鍵を入れて送ります。
- サーバは、自分の秘密鍵とクライアントの公開鍵から秘密を生成します。

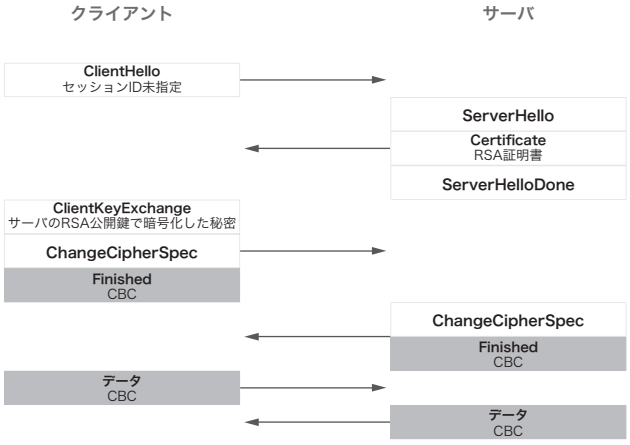

図-1 TLS 1.2フルハンドシェイク
TLS_RSA_WITH_AES_128_CBC_SHA

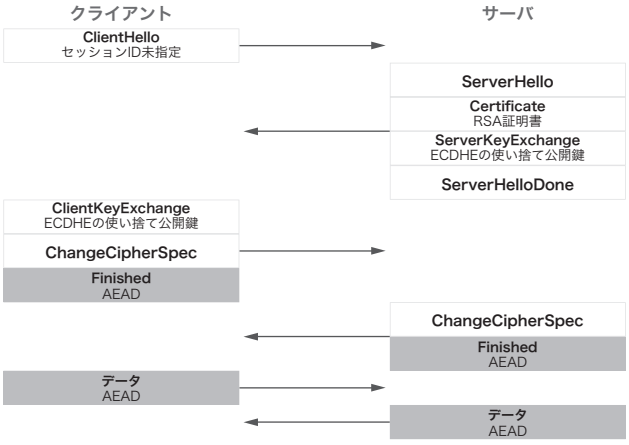

図-2 TLS 1.2フルハンドシェイク
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

フルハンドシェイクは、TLS 1.0、1.1、1.2では変わり映えしません。しかしながら、TLS 1.3のフルハンドシェイクは根本的に再設計されています。なんと言っても、Helloに鍵交換の役割を持たせることで、RTT(Round Trip Time)を1つ減らしているのです。

- クライアントは、ECDHEの使い捨て公開鍵と秘密鍵を作り、ClientHelloのオプションに公開鍵を格納して送ります。
- サーバも、ECDHEの使い捨て公開鍵と秘密鍵を作り、ServerHelloのオプションに格納して送ります。また、ここからすぐ通信路が暗号化されます。サーバの証明書を格納するCertificateやFinishedは、暗号化されて送られます。Finishedを送ったあとは、更に安全な暗号路へと切り替わります。図-3の灰色の濃さの違いは、この暗号路の違いを表現しています。
- クライアントは、現在の暗号路でFinishedを送った後、更に安全な暗号路へと切り替えます。

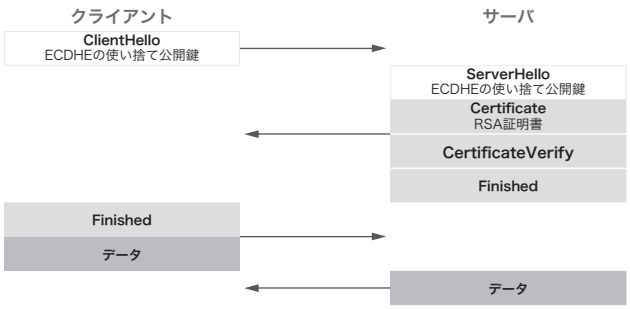

図-3 TLS 1.3フルハンドシェイク
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

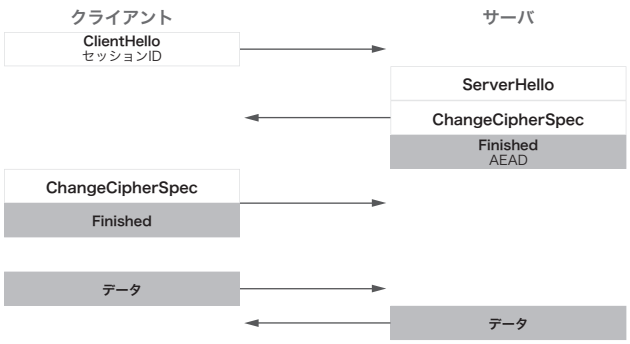

図-4 TLS 1.2セッション再開

3.5.2 セッションの再開

クライアントとサーバが、以前TLS 1.2でフルハンドシェイクしていれば、そのセッションを再開することで、鍵交換を省略できます。図-4をご覧ください。

- クライアントは、ClientHelloで再開したいセッションのIDを指定します。
- サーバは、指定されたセッションIDに対する状態を保存していれば、それを使って暗号路に切り替えます。

セッションの再開は、公開鍵暗号の重い計算を省略するばかりではなく、RTTも1回減らせます。しかし、この方法ではサーバがセッション情報を保持する必要があります。クライアントの数に比例して、保持すべき状態の数も増えます。サーバの負担が増えるこの方法は、あまりうまいとは言えません。

サーバの負担を減らす方法として、RFC 5077でセッションチケットが定義されています。セッションチケットとは、サーバのみが復号できる暗号化されたセッション情報のことです。セッションチケットを用いると、サーバはセッション情報を保持する必要がなくなります。

TLS 1.2でセッションチケットを使うためには、まずセッションチケットのためのフルハンドシェイクをする必要があります(図-5)。

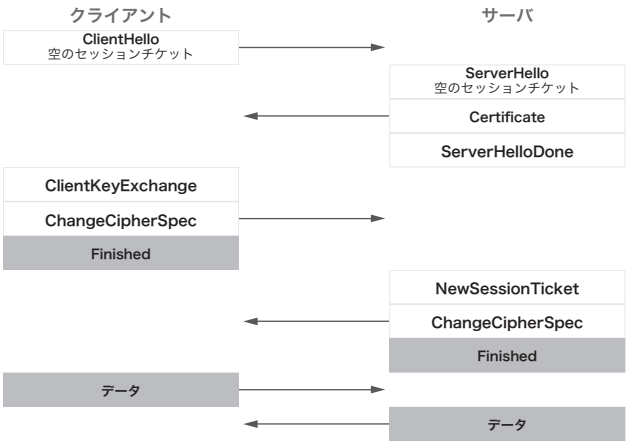

図-5 TLS 1.2セッションチケットのためのフルハンドシェイク

- クライアントは、ClientHelloの拡張として空のセッションチケットを送り、セッションチケットに対応していることをサーバに知らせます。
- サーバも、空のセッションチケットをServerHelloのオプションに指定することで、セッションチケットに対応していることをクライアントに知らせます。
- サーバは、ChangeCipherSpecを使って暗号路に切り替える直前に、生成したセッションチケットをNewSessionTicketに入れて送ります。
- クライアントは、現在のセッション情報と送られてきたセッションチケットを対応付けて保存します。

次に、TLS 1.2でセッションチケットを使ってセッションを再開する方法を説明します(図-6)。

- クライアントは再開するセッション情報とセッションチケットを取り出し、ClientHelloのオプションにセッションチケットを格納して送ります。
- サーバは、セッションチケットを復号して、セッション情報を得ます。必要であれば、新しいセッション情報をNewSessionTicketで送ります。その後、暗号路に切り替えます。
- クライアントは、前述のセッション情報を使って、暗号路に切り替えます。

TLS 1.3のセッションチケットは、RFC 4297で定義されているPSK(Pre-Shared Key)と統合されています。PSKとは、サーバやクライアントの認証のために、公開鍵ではなく、あらかじめ共有している秘密を用いる方式のことです。TLS 1.3のPSKハンドシェイクをセッションチケットの機能に限って使えば、TLS 1.2のそれとあまり変わりません(図-7)。細かな違いは、次のとおりです。

- フルハンドシェイクの後、サーバがいつでもNewSessionTicketを送信できます。
- TLS 1.3のフルハンドシェイクと同様、暗号路が2回切り替わります。

3.5.3 証明書を用いたクライアント認証

TLSを用いて、あるサーバのあるページにアクセスしている場合を考えましょう。そのページにあるリンクの先は、同じサーバにあるが、証明書を使ったクライアント認証が必要なコンテンツだったとします。

TLS 1.2では、途中で証明書を使ったクライアント認証が必要となった場合、再ネゴシエーションを用います。これは文字通り、ハンドシェイクを再び実行するのです。フルハンドシェイクと違うのは、暗号路の中でハンドシェイクすることです(図-8)。

- サーバは、HelloRequestを送ってクライアントに再ネゴシエーションを要求します。
- クライアントは、ClientHelloを送ります。
- サーバは、ServerHelloを送る際に、CertificateRequestも送ってクライアントの証明書を要求します。
- クライアントは、ClientKeyExchangeを送る際にクライアントの証明書をCertificateに入れて送ります。

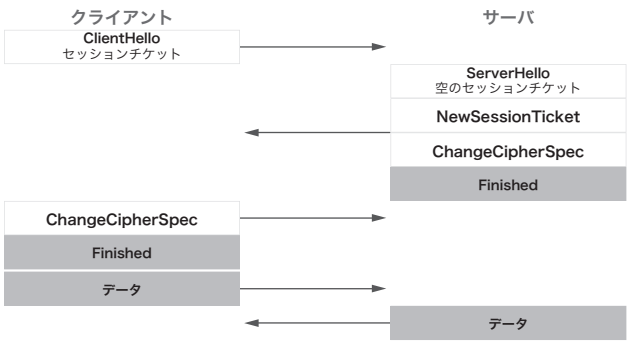

図-6 TLS 1.2セッションチケットを用いたセッションの再開

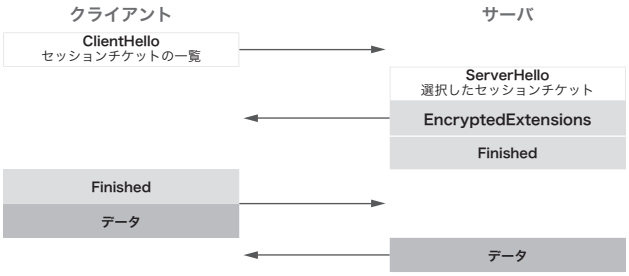

図-7 TLS 1.3セッションチケットを用いたセッションの再開

再ネゴシエーションの本来の目的は、暗号路をリフレッシュして暗号路の寿命を延ばすことです。証明書を用いたクライアント認証にも使われるのは、CertificateRequestをServerHelloの直後に送らなければならないというTLS 1.2の制約からです。

TLS 1.3では、暗号路のリフレッシュと証明書を用いたクライアント認証を明確に分けます。CertificateRequestは、いつでもサーバからクライアントに送れるようになります(図-9)。

3.5.4 0-RTT

TLS 1.3では、ClientHelloを送る際にアプリケーションのデータも暗号化して送る0-RTTというハンドシェイクが検討されています。少し複雑なので今回は説明を割愛します。興味がある方はTLS 1.3のIDを参照してください。

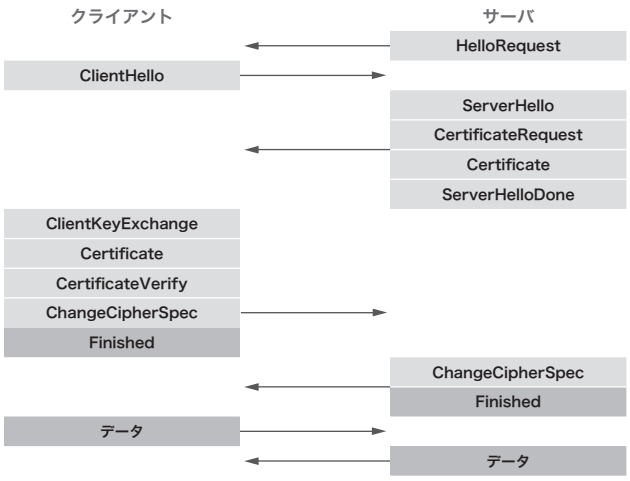

図-8 TLS 1.2での証明書を用いたクライアント認証

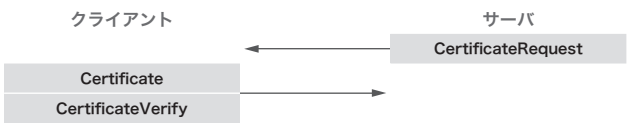

図-9 TLS 1.3での証明書を用いたクライアント認証

3.6 圧縮

TLS 1.2以前では、平文と暗号文に加えて、圧縮文という書式があります。圧縮機能を使う場合、平文が圧縮されて圧縮文となり、更に暗号化されて暗号化文となります。残念ながら、圧縮の機能を利用するとCRIMEやBREACHという攻撃にさらされることになります。

ですから、TLS 1.2を利用する場合、圧縮機能を利用してはいけません。平文を圧縮文を介さず直接暗号化し、暗号文にします。TLS 1.3では、圧縮文という書式は削られ、平文と暗号文のみが定義されています。

3.7 Let's Encrypt

Let's Encryptは、サーバの証明書を無償で自動発行するプロジェクトです。発行できる証明書の種類は、ドメイン認証(DV: DOmain Validation)のみで、企業認証(OV:Organization Validation)やEV(Extended Validation)認証の証明書は発行できません。現時点では、ワイルドカード証明書は発行できません。複数のサーバ名がある場合は、DV証明書をサーバ名の数だけ発行してもらうか、代替名(SAN:Subject Alternative Name)を利用するとよいでしょう。Let's Encryptが提供するコマンド群は、IETFが標準化を進めているACME(Automatic Certificate Management Environment)というプロトコルを実装しています。Let's Encryptについては、「IIR Vol.30、1.4.2 Let's Encryptプロジェクトと証明書自動発行のためのACMEプロトコル」も参考にしてください。

3.8 おわりに

今回は、攻撃手法に関しては名前だけ示して、具体的方法については説明しませんでした。検索すれば、それぞれの攻撃手法に対する詳しい解説が簡単に見つかります。興味を持たれた方は、ぜひ読んでみるといいでしょう。

執筆者:
山本 和彦(やまもと かずひこ)
株式会社IIJイノベーションインスティテュート 技術研究所 主幹研究員。
プログラミング言語Haskellの並行技術をネットワークプログラミングへ応用することに興味を持つ。
最近取り組んでいるプロトコルはHTTP/2やTLS 1.3。
翻訳書に「プログラミングHaskell」「Haskellによる並列・並行プログラミング」がある。