第５章 暗号

5.1 暗号の基礎

5.1.1 暗号とは

暗号とは通信の内容が当事者以外には理解できないように、普通の文字･記号を一定の約束で他の記号に置き換えたものである。

図5.1 暗号化と復号

平文を暗号文に変換することを**暗号化**、暗号文を平文に変換することを**復号**という（一般には復号化とは呼ばない）。暗号化を行うための鍵（暗号化鍵）と復号を行うための鍵（復号鍵）に同じものを使用する暗号を**共通鍵暗号**と呼ぶ（秘密鍵暗号、慣用鍵暗号、対称鍵暗号などとも言う）。暗号化鍵と復号鍵に別のものを使用する暗号は**公開鍵暗号**と呼ぶ。公開鍵暗号は 20 世紀後半に発見されたもので、暗号の歴史の中では比較的新しい技術である。

なお平文（ひらぶん、へいぶん）は暗号化されていないデータのことで、英語の Plain Text の直訳であり元々日本語にある言葉ではない。

5.1.2 暗号の目的

暗号技術の利用の目的は暗号方式によって異なる。共通鍵暗号ではデータの秘匿のみを目的にするのに対して、公開鍵暗号はデータの秘匿以外に**デジタル署名**を用いてデータの完全性を保証する目的などに使われる。この公開鍵暗号の技術によって新しい認証方式や EC（Electronic Commerce: 電子商取引）での各種の応用が可能になる。

共通鍵暗号の使用目的 （暗号化と復号に同じ鍵を使う）

機密性：安全でない通信路での盗聴などからメッセージの秘密を守ること.

公開鍵暗号の使用目的 （暗号化と復号に違う鍵を使う）

機密性：安全でない通信路での盗聴などからメッセージの秘密を守ること.

完全性：文書の改竄を検出し文書が本物であることを確認すること.

認証：その人が本人であるとこを確認すること．なりすまし防止.

否認防止：情報を送信したことを否認できないこと.

公開鍵暗号の使用目的である完全性・認証・否認防止には、公開鍵暗号のデジタル署名機能を利用する。

5.1.3 ストリーム暗号とブロック暗号

暗号化を行う際に、1 文字単位で暗号化を行うものを**ストリーム暗号**と呼ぶ。一方平文を一定の長さのブロックに区切り、ブロックごとに暗号化するものを**ブロック暗号**と呼ぶ。ストリーム暗号では処理スピードが速くなるが、暗号化が単純になる可能性がある。ブロック暗号ではブロック長が長くなればなる程、攻撃時の探索空間が大きくなるので、セキュリティ的には強固になるが処理のコストは高くなる。

なおストリーム暗号はブロック長が１のブロック暗号と見なすこともできる。また公開鍵暗号ではその性質上、ブロック暗号でないと実用的ではないのでストリーム暗号が使われることはない。

5.1.4 ブロック暗号のモード

通常平文が同じで暗号化鍵も同じ場合、暗号文も同じなる（ECB)。このことは機密性の観点から言えば好ましくない事態である。何故ならば、通信内容を予測できるような環境（例えばプロトコル上“OK”または“NG”の２通りしかサーバが応答しないような場合）では、暗号文から鍵も予測できてしまうからである。

このことを防止するために、平文が同じで暗号化鍵も同じ場合でも暗号文が同じにならないような暗号化の仕方（モード）がいくつか考案されている。以下にその内容を示す。なお図中の実線は暗号化を表し、点線は XOR（排他的論理和）を表す。

【**ECB** (Electronic Code Book) mode】 従来の基本モードである。平文をブロックに分割した後、各平文ブロックを暗号化鍵で暗号化する。暗号化される平文ブロックが同じであれば、暗号文も同じになる（図 5.2)。

ECB モードは現在では使用してはいけないモードとなっている。例えば画像データを ECB モードで暗号化した場合，各ピクセルの色（輝度値）がそれぞれ別の値に変化するだけなので，暗号化しても画像パターンが識別可能になる恐れがある。

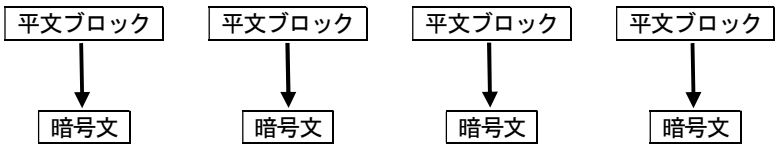

図 5.2 ECB (Electronic Code Book mode)

【**CBC** (Cipher Block Chaining) mode】 暗号化された前ブロックと、まだ暗号化されていない現平文ブロックとの **XOR** をとり、これを暗号化する。最初の暗号化では前暗号化ブロックがないため、**初期ベクトル**(IV)が必要である。よく使われる手法だが、通信などにおいて途中で通信エラーを起こすと、そのエラーが次々に伝播し復号不可能になるという欠点がある（図 5.3)。

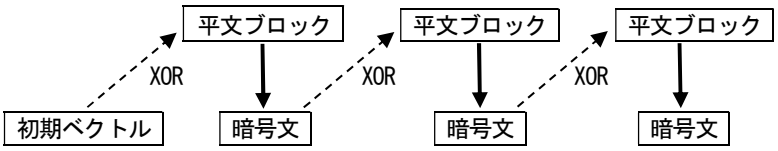

図 5.3 CBC (Cipher Block Chaining mode)

【**OFB** (Output Feed Back) mode】 前ブロックの暗号化ブロックをさらに暗号化し、次の平文ブロックとの XOR をとる。元の平文ブロックは直接暗号化されないのが特徴である（図 5.4)。

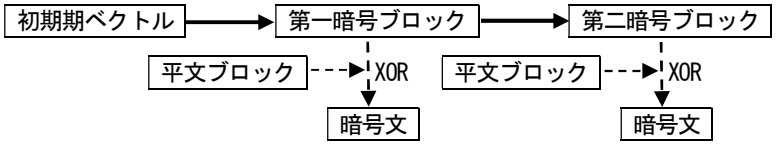

図 5.4 OFB (Output Feed Back mode)

【CFB (Cipher Feed Back) mode】 前段の暗号結果をさらに暗号化して次の暗号ブロックとし、次の平文ブロックと XOR をとる。

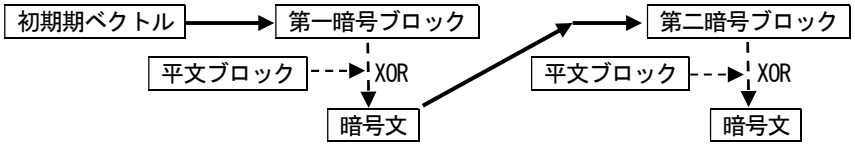

図 5.5 CFB (Cipher Feed Back mode)

XOR (排他的論理和) は暗号処理の中で多用される。それはあるデータに対して同じデータで XOR を 2 回取ると、データが元に戻る性質があるからである。

例えば、元のデータが 11010111 とする。このデータと 10110010 の XOR を計算すると 01100101 となる。この 01100101 に対して、もう一度 10110010 との XOR を計算すると、元の 11010111 に戻る (図 5.6)。

```
  11010111          01100101
XOR 10110010      XOR 10110010
----------        ----------
  01100101          11010111
```

図 5.6 同じ XOR を 2 回計算すると、データは元に戻る

5.1.5 Base64

バイナリデータをテキストデータに変換する最も一般的な手法に Base64 がある。Base64 ではバイナリデータを 6bit 毎に区切り、2^6 = 64 個の文字 (キャラクタ) で表現し直す。この場合 3Byte のデータが 4 個のキャラクタ(4Byte) で表現されることになる。例えば、0x00, 0x10, 0x83 のバイナリデータは、6bit

ずつ区切ると 0x00, 0x01, 0x02, 0x03 となるため、Base64 によるエンコードでは“ABCD”というテキストデータに変換される（図 5.7）。

暗号化ではバイナリデータを取り扱う場合が多いが、そのデータを表示・交換する場合はこの Base64 で符号化する場合が多い。なお、Base64 は符号化方式であって暗号化方式ではないので、この点を良く注意すべきである。

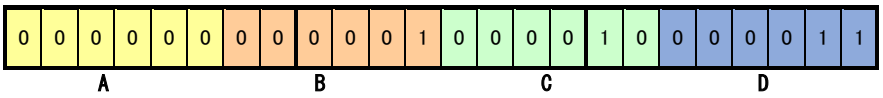

図 5.7 Base64 によるエンコード例

5.1.6 メッセージダイジェストと一方向ハッシュ関数

メッセージダイジェスト（MD）とはメッセージ（データ）の特徴を抽出したもので、元のメッセージを一文字（1bit）でも変えるとメッセージダイジェストは大きく変化するという性質を持つ。メッセージダイジェストは（**暗号学的**）**ハッシュ値**または**フィンガープリント**とも呼ばれ、メッセージからメッセージダイジェストを計算する関数を**一方向ハッシュ関数**などと呼ぶ。

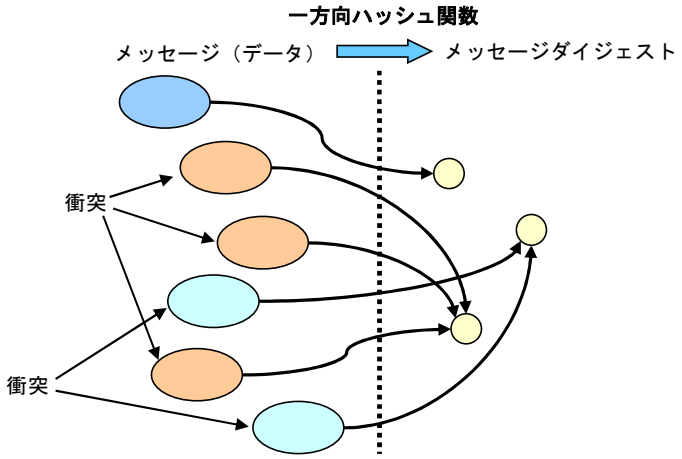

図 5.8 一方向ハッシュ関数

一方向ハッシュ関数は多対１対応であるので、メッセージダイジェストから元のメッセージを復元することは不可能である。ただし、多対１対応であるので、理論上は同じメッセージダイジェストを持つ複数のメッセージを探し出すことは可能であり、このことを「**衝突問題**」と呼ぶ（図5.8)。一方向ハッシュ関数では、「衝突問題」が発生しにくいと言うことが関数の性能の重要な部分を占める。

メッセージダイジェストはデジタル署名などで大きな役割を果たしている。

以下に主な一方向ハッシュ関数を示す。

【DES】

共通鍵暗号の DES に基づいたハッシュ値化方法である。初期の Unix/Linux のパスワードのハッシュ値（MD）化で使用されていたが、現在では機能不足であり、デジタル署名は勿論、パスワードのハッシュ値化でも使用されることは無くなった。

【MD4,5 (Message Digest 4, 5)】

MD5 は以前は Unix/Linux のパスワードのハッシュ値化やデジタル署名に使用される手法であったが、これも現在では機能不足および脅威に対する脆弱により、使用は避けられるようになった。MD4, MD5 は共に、ある条件下で衝突問題が発生することが知られている。

【SHA-1 (Secure Hash Algorithm 1)】

1995 年に米国の**国立標準技術局**（NIST）によって、米政府標準の一方向ハッシュ関数として採用された手法である。しかしながら、2005 年に効果的な攻撃方法が発見され、現在では使用を禁止しているシステムも存在しており、SHA-2 への移行が強く推奨されている。

【SHA-2 (Secure Hash Algorithm 2)】

SHA-1 の改良型(2001 年)である。 SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256 の６つの種類があり、最後の数字が出力するハッシュ長を表している（各 224, 256, 384, 512bit)。**SHA-256**, **SHA-512** が基本的な手法で、それ以外の手法では 256bit または 512bit を切り詰めてハッシュ値

を出力している。出力は通常はバイナリとなるので、表示する場合はBase64で符号化される。

【SHA-3 (Secure Hash Algorithm 3)】

2012年にNISTによる次世代暗号コンペティションの結果、KeccakがSHA-3として選出された。それまでのSHA-1/2とは内部構造が大きく変わっている(内部構造がSHA-1/2と異なっていることがコンペティションの要求条件であった)。しかしながらSHA-2が現状でまだ十分な強度があると考えられているため、あまり普及はしていない。

【Bcrypt (Blowfish)】

暗号化アルゴリズムであるBlowfishに基づいたハッシュ値化方法(1999年)である。

5.1.7 暗号の強度

暗号を使用する上で最も気がかりになるのはその強度（有効性）である（鍵を知らない状態での暗号の解読の困難さ)。しかしながら（実装された）暗号の強度を明確に決定することは困難である。暗号の理論とその暗号の実装（実際に動作するようにシステム上に組み込むこと）は別の話であり、理論的は特に問題はない場合でも、実装に脆弱性があって暗号が簡単に解けてしまうこともある。

また暗号のアルゴリズムを隠蔽することによって暗号の強度を上げる手法もあるが、その手法は最も下策であると言われている。なぜならば、アルゴリズムが漏洩した瞬間にその暗号は無価値なものとなり、対応の時間さえ取ることが難しいからである（ゼロデイの脆弱性)。

暗号の強度（有効性）は、暗号アルゴリズムを公開した状態で、「最新のシステムを使用しても、その時点で誰もその暗号を（効率的に）解くことができない」という事実の上にのみ立脚している。

5.2 共通鍵暗号

5.2.1 共通鍵暗号の種類

以下に主な共通鍵暗号について説明する。

【シーザ暗号（シフト暗号)】

シーザ暗号は古代ローマの政治家であり、将軍もであったジュリアス・シーザが考えたストリーム暗号であり、文字を決まった数だけずらすことにより暗号化を行う。シーザ自身は3文字ずらしていたとされるが、一般に文字をずらす暗号は、ずらす文字数に関わらずにシーザ暗号と呼ばれる場合が多い。ただし3文字の場合とそれ以外を区別して、3文字に限らない一般的な場合は**シフト暗号**と呼ぶ場合もある。

例えば ABC（平文）を3文字後ろにずらすと DEF（暗号文）になる。また IBM（平文）を一文字前にずらすと HAL（暗号文）になる。HAL の例は良く引き合いに出されるが、元ネタは SF 作家のアーサー・C・クラークの「2001 年宇宙の旅」などに出てくる人工知能コンピュータ HAL9000 である。名作なので、是非一読をお勧めしたい（作品は4部作)。

なおシーザ暗号は最古の暗号との記述も良く見かけるが、紀元前6世紀ごろの古代ギリシャのスパルタでは、布を棒に巻き付けて縦読みにするという暗号（**スキュタレー暗号**）が使われていたとされる。さらに遡って紀元前 20 世紀ごろの古代バビロニアでも既に暗号が使われていたとの報告もある。

【換字式暗号】

全ての文字を別の文字に対応させて暗号化する手法を**換字式暗号**と呼ぶ。特に文字を１対１に変換する方法を**単一換字式暗号**と呼ぶ。例えば、A→X, B→C, C→P など変換する。

シーザ暗号よりは複雑だが、**文字出現頻度**などの統計情報から解析可能である。例えば、英語の文章では統計的に“e”が最も出現頻度が高いので、暗号文中で最も多く使用されいる文字は“e”であると推測できる。エドガー・アラン・ポーの小説である「黄金虫」では、キャプテン・キッドの宝の隠し場所としてこの暗号が使われている。

暗号化は通常 1 文字単位で行われるので、ストリーム暗号である。

【ヴィジュネル暗号】(5.3 節参照)

16 世紀に考案されたシフト暗号の一種（従ってストリーム暗号）で、300 年間破られることが無かったが、**チャールズ・バベッジ**（歯車と蒸気機関でコンピュータを作ろうとした人物）により解析された。ヴィジュネル方陣と呼ばれるテーブルを使用しており、換字式暗号の一種とも見なせる。

【バーナム暗号】

19 世紀に考案された暗号である。十分に長い乱数を鍵とする暗号方式で、鍵の長さが平文を超える場合は、理論的に解読することは不可能である（どの様にでも解釈が可能なため)。

例えば、「ABCDEFGHIJ」という暗号文があり鍵も 10 文字で、復号は暗号文と鍵との XOR を取ることであるとする（符号化については別の話なので省略)。その場合、鍵が分からなければあらゆる 10 文字の文章が平文である可能性があり、その中からどれが正解であるかを判断することは不可能だからである。ただし同じ鍵は使用できないため、鍵の生成や受け渡しに問題があり実用的な暗号ではない。

【エニグマ】

第二次世界大戦の時期にナチスドイツが開発したロータ方式の暗号である。鍵が 159,000,000,000,000,000,000 通りもあるとされ、当時としては最高の強度を誇った。しかし、運用の拙さや**アラン・チューリング**（コンピュータの理論的な基礎を築いた人物）らのチームが開発したコロッサス２(1944 年)によって解読された。またエニグマ暗号の解読が、連合軍のノルマンディー上陸作戦を成功に導いたとも言われている。

大戦後はイギリス政府により、コロッサス２の成功については箝口令が敷かれ（大戦後も各国で重要情報の通信にエニグマが使用されており、それを解読して外交的優位に立とうとしたため と言われている)、1970 年までは機密扱いであった。しかしその後の情報公開により、エニグマ解読に対するアラン・チューリングらの功績も明らかとなった。

【RC4】

1987 年に開発されたストリーム暗号である。無線 LAN の WEP などで使用されたが、現在では使用は推奨されていない。

【DES (Data Encryption Standard)】

1972 年に作成された 64bit ブロック暗号で鍵長は 56bit である。かつては米国の標準の共通鍵暗号で、重要技術として輸出規制が掛けられたこともある。現在では DES 自体には既に十分な強度がないとして使用は推奨されていない。

一方 DES を 3 回行う**トリプル DES**（**3DES**：3 種類の鍵を組み合わせる）はまだ十分な強度があるとして、主に処理スピードが要求されるような場面で使用されている。3DES は日本では、SUICA などに採用されている Felica （Sony）で使用されている（つまり SUICA では 3DES が使用されている)。

【Blowfish】

1993 年に作成された 64bit ブロック暗号である。鍵長は 32～448bit まで可変で、ライセンスフリーな暗号化方式である。

【AES (Advanced Encryption Standard)】

AES は 2001 年に、米国の国立標準技術局（NIST）により連邦情報処理規格（FIPS PUB 197）として規定された、米政府の標準共通鍵暗号である。かつては DES が米政府の標準共通鍵暗号であったが、コンピュータ性能の年々の向上により DES が時代遅れになったことから、DES に代わる標準暗号として AES のアルゴリズムの公募が行われた（ただし、現在でも 3DES は十分強度があると考える人々もいる)。

AES の条件は、ブロック長として 128bit, 鍵長として 128, 192, 256bit が利用可能なブロック暗号といった他に、30 年以上の暗号として用いられる強度が見込めるといった条件があった。選考の結果、最終的には**Rijndael**（ラインダール）が採用された。変換の処理数（ラウンド数）は鍵長により、10, 12, 14 段となる。AES は無線 LAN (Wi-Fi) の暗号化アルゴリズムとしても有名である。

5.2.2 共通鍵暗号の問題点

共通鍵暗号の運用において最大の問題となるのが**鍵の共有方法**である。特に離れた場所にいる者同士の鍵の交換方法が問題となる。つまり、暗号は安全な通信路がないときに利用したいのに、その暗号を行う鍵を共有するためには別の安全な通信路が必要になるという矛盾に陥る。

また鍵の管理も煩雑になる。例えばN人のグループ内で共通鍵暗号を用いると、それぞれが(N-1)個の鍵を管理しなくてはいけないので、全体で N(N-1)/2 個の鍵が必要となる。例えば10人のグループでは、全体で実に45個の鍵が必要となる。

5.4 公開鍵暗号

5.4.1 公開鍵暗号の概要

共通鍵暗号の最大の問題は鍵の共有方法であった。そのような状況の中、1976年にWhitfield Diffie と Martin Hellman は第三者に通信を盗聴されている状況下であっても、その第三者に知られることなく、通信者同士がお互いに同じ共通鍵を生成できるアルゴリズムを発見する。これが **Diffie-Hellman 鍵交換法**と呼ばれる手法である。

Diffie-Hellman 鍵交換法自体は完全な公開鍵暗号とは言えないが、公開鍵暗号の考え方を示唆した最初のものである。さらにその翌年の 1977 年には Ronald Rivest らによる **RSA 暗号**が発表され、これが公開鍵暗号の最初のものである。しかしながらDiffie-Hellman鍵交換法がRSA暗号より劣っているわけではなく、使う場面や目的によって長所と短所がある。

公開鍵暗号では公開鍵と秘密鍵の**二つの鍵が生成され、片方の鍵で暗号化したものは、もう片方の鍵でないと復号できない**という性質を持っている。この二つの鍵は理論的には相対的であり、どちらを秘密鍵または公開鍵にしても良い。この性質と**メッセージダイジェスト**をうまく利用すると、公開鍵暗号に「**デジタル署名**」の機能を持たせることが可能となる。さらにこの「デジタル署名」の機能を使用すると、先に述べたように共通鍵暗号では実現できなかっ

た、情報の**完全性、認証性の保障**および**否認防止**の機能を実現することが可能となる。

公開鍵暗号では、ストリーム型で暗号化を行うと、単一換字式暗号（すべての文字を別の文字1 対1 で対応させる暗号）と同等なものとなり、意味がなくなるので必ずブロック単位での暗号化と復号が行われる。

5.4.2 公開鍵暗号による情報の秘匿

公開鍵暗号では公開鍵と秘密鍵の二つの鍵が生成される。**二つの鍵の片方の鍵で暗号化したものは、もう片方の鍵でないと復号できない**という性質を利用して、生成した鍵の一方を公開し（**公開鍵**）、もう一方を厳重に保管する（**秘密鍵**）。なお二つの鍵は前項で述べたように理論的には相対的なので、どちらを公開鍵・秘密鍵にしても良い。ただし暗号化用ツールである OpenSSL などの実装では公開鍵と秘密鍵を明確に区別し、計算を容易にするために公開鍵の長さを短くしている場合もある（OpenSSL の実装では公開鍵の一部のデータはほぼ固定となっている）。

図 5.17 において、Bob が Alice へ公開鍵暗号を利用した暗号化メールを送信する場合を考える。

この場合の暗号化・復号の手順は以下の通りになる。

① Alice は先ず、ツールを用いて自らの**公開鍵と秘密鍵のペアを生成する。**公開鍵は Web ページなどで公開し、秘密鍵は手元に置く。
② Bob は Alice へのメールを Alice の**公開鍵で暗号化**し、送信する。
③ 公開鍵で暗号化したメールは、同じ公開鍵では復号できないので、もし通信路上に盗聴者がいてもこのメールを解読することはできない。
④ Alice は Bob からのメールを自分の**秘密鍵で復号**しメールを読む。

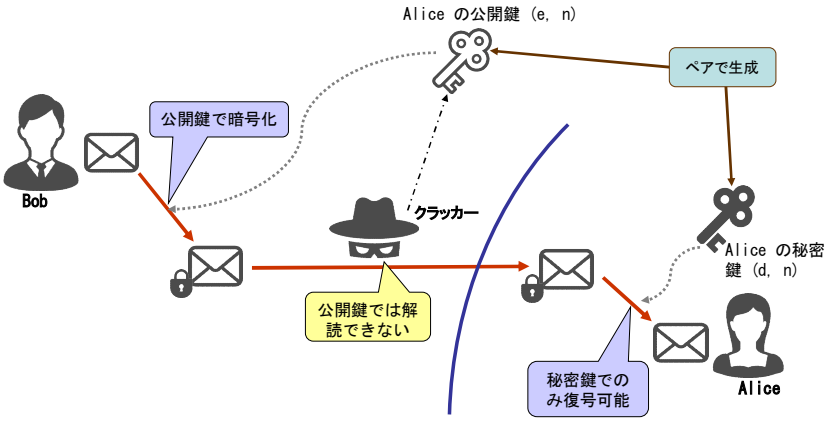

図 5.17 公開鍵暗号よる情報の秘匿

Alice の公開鍵で暗号化したものは、Alice の秘密鍵でしか復号できないので、結局 Bob からのメールは、秘密鍵を持つ Alice 自身しか読むことができないということになる。

5.4.3 公開鍵暗号によるデジタル署名

図 5.18 に Alice から Bob へデジタル署名付きのメールを送信する場合の手順を示す。

① Alice は先ず、ツールを用いて自らの**公開鍵**と**秘密鍵**のペアを生成する。公開鍵は Web ページなどで公開し秘密鍵は手元に置く。
② Alice は自分のメールの MD を計算する。
③ Alice は、計算した MD を自分の秘密鍵で暗号化する。MD を暗号化したものが、Alice の**署名**である。
④ 署名をメールに添付して Bob へ送信する（実際のメールでは、これを Bob の公開鍵で暗号化して送信するが、ここでは簡単化のために暗号化は行わない）。
⑤ 通信路上の改ざん者は、メール本文と添付された MD の辻褄が合うように、それらを改ざんすることは不可能である。
⑥ メールを受け取った Bob は、添付されている署名を Alice の公開鍵で復号して MD を取り出す。Alice の公開鍵で復号できることにより、その MD は Alice によって計算されたことが保障される。

⑦ Bob は Alice のメールの MD を計算する。

⑧ Bob は自分で計算した MD と Alice から受け取った MD を比較して一致していることを確認する。両者が一致していることにより、メールが途中で改ざんされていないことが保障される。

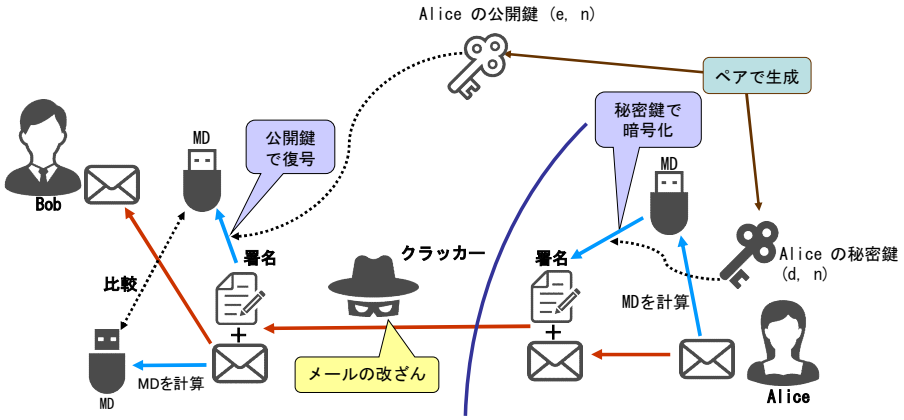

図 5.18 公開鍵暗号によるデジタル署名の概要

つまり、メッセージ（データ）のメッセージダイジェストを自分の秘密鍵で暗号化したものが、**自分のデジタル署名**であり、これによりメッセージ（データ）の完全性と自己の認証性および否認防止を保障することが可能となる。

なお、図 5.18 はデジタル署名の概要を示しているものなので、実際のシステムへの実装では手順等が変更される場合もある。

5.5 Diffie-Hellman 鍵交換法

5.5.1 Diffie-Hellman 鍵交換法とは

Diffie-Hellman 鍵交換法（鍵共有法とも呼ばれる）は、1976 年に Whitfield Diffie と Martin Hellman によって発見された手法で、通信の当事者同士が第三者に知られることなく同じ共通鍵を生成できるアルゴリズムである。同じ共

通鍵を生成した後は、その共通鍵を使用して暗号化通信を行う。共通鍵の生成はリアルタイムで行われるので、メールなどのメッセージの暗号化には向いていない。

なお、このアルゴリズムの特許は 1997 年 4 月に失効している。またこの功績により二人は 2015 年に情報科学界でのノーベル賞とも言われるチューリング賞を受賞している。

5.5.2 Diffie-Hellman 鍵交換法のアルゴリズム

Diffie-Hellman 鍵交換法のアルゴリズムは以下のようになる(表 5.19)。なお、以下で使用する ^ は累乗を、mod は割り算での余りを意味する。

1) 先ず鍵を交換する 2 者の内の一方（以後 Alice）が、十分大きな（通常 512～1024bit）素数 P を決め、その原始根を G とする。ただし実装では計算時間の関係から、先に G を決め（通常 2 や 5 が選ばれる）、それを原始根とする素数 P を定める場合が多い。Alice は 0 =< Xa =< P-1 である秘密鍵 Xa をランダムに生成し、公開鍵 Ya を Ya = (G^Xa) mod P で計算し、鍵を交換する相手（以後 Bob）に（P, G, Ya）の鍵セットを送信する(通常の実装ではこのデータは X.509 の SubjectPublicKeyInfo として ANS.1 の DER エンコーディングを使って送られる)。
2) Bob は Alice から送られてきた鍵セットから、0 =< Xb =< P-1 である秘密鍵 Xb をランダムに生成し、 K = (Ya^Xb) mod P を計算して共通鍵 K を得る。K は K = (((G^Xa) mod P)^Xb) mod P = (G^(Xa*Xb)) mod P とも書ける。さらに Bob は公開鍵 Yb を Yb = (G^Xb) mod P で計算し、鍵セット(P, G, Yb)を Alice に送る（情報としては Yb のみでも可)。
3) Alice は Bob から送られてきた Yb を使って、共通鍵 K を K = (Yb^Xa) mod P = (G^(Xa*Xb)) mode P で計算する。
4) 以後、Alice と Bob は共通鍵 K を使用してデータを暗号化し、通信を行う。

Alice と Bob は結局同じ K = (G^(Xa*Xb)) mode P を計算していることになるが、相手の秘密鍵 Xa, Xb の情報は P で割った余りである Ya, Yb の中に隠されているため、第三者は Ya, Yb キーを見ても Xa, Xb を知ることはできない。

ただし、プログラム中で使用する素数 P があまり大きくない場合には、通信

データ (P, G, Y) から秘密鍵 X を推測することが可能である。また、サーバを認証する方法が（アルゴリズムとして）用意されていないため、第三者が通信に割り込んで、情報を自分のものとすりかえてしまう**中間者** (Man In The Middle) **攻撃**に弱いとされている。

運用によってはＹキーを固定し、公開鍵とみなして相手を認証することも可能である。またDiffie-Hellman鍵交換法を証明書などに用いる場合は、Ｙキーは固定となる。この手法は **static DH** または単に **DH** と呼ばれるが、同じＹキー（即ち同じ素数Ｐと秘密鍵Ｘ）を使い続けることはシステムの危殆化に繋がる。一方その都度パラメータを計算し直す通常の Diffie-Hellman 鍵交換法は static DH と区別して **DHE** (DH Ephemeral) とも呼ばれる。

Alice	公開鍵の交換	Bob
素数Ｐ、 原始根Ｇの生成 秘密鍵 Xa の生成 公開鍵 Ya = G^Xa mod P の計算		
鍵セットの転送	(P, G, Ya) →	秘密鍵 Xb の生成 共通鍵 K = Ya^Xb mod P の計算 公開鍵 Yb = G^Xb mod P の計算
共通鍵 K = Yb^Xa mod P の計算	← (P, G, Yb)	鍵セットの転送

表5.19 Diffie-Hellman鍵交換法 (DHE) のアルゴリズム

5.5.3 Diffie-Hellman鍵交換法の例題

上記の Diffie-Hellman 鍵交換法のアルゴリズムを簡単に書き下せば以下のようになる。

1. AはG, P と秘密鍵Xaを決める。
2. Aは Ya＝G^Xa mod P を計算し、(P, G, Ya) をＢへ送信。
3. BはＰを元に秘密鍵Xbを決める。Yb＝G^Xb mod P を計算し、(P, G, Yb) をＡへ送信。
4. Bは K = Ya^Xb mod P を計算し、共通鍵Ｋを得る。
5. Aは K = Yb^Xa mod P を計算し、共通鍵Ｋを得る。

これを参考に、以下にDiffie-Hellman鍵交換法の例題を示す。ただし、以下の例題は素数Pが非常に小さい場合であることに注意されたい。なお **mod** や ^ の計算には MS Windowsに標準搭載されている電卓（関数電卓）を使用すると便

利である)

【例題１： G=2, P=7, Xa=4, Xb=3 の場合】

A：G, P を決め(2, 7)、さらに秘密鍵 Xa=4 を決める。
Ya = 2^4 mod 7= 16 mod 7 = 2
(P, G, Ya) = (7, 2, 2) を B へ送信

B：G, P から秘密鍵 Xb=3 を決める。
Yb = 2^3 mod 7 = 8 mod 7 = 1
(P, G, Yb) = (7, 2, 1) を A へ送信

Kab = 2^3 mod 7 = 8 mod 7 = **1**
A：Kab = 1^4 mod 7 = 1 mod 7 = **1**

【例題２： G=2, P=13, Xa=5, Xb=4 の場合】

A：G, P を決め(2, 13)、さらに秘密鍵 Xa=5 を決める。
Ya = 2^5 mod 13 = 32 mod 13 = 6
(P, G, Ya) = (13, 2, 6) を B へ送信

B：G, P から秘密鍵 Xb=4 を決める。
Yb = 2^4 mod 13 = 16 mod 13 = 3
(P, G, Yb) = (13, 2, 3) を A へ送信
Kab = 6^4 mod 13 = 1296 mod 13 = **9**

A：Kab = 3^5 mod 13 = 243 mod 13 = **9**

5.5.4 Perfect Forward Security (PFS)

近年、**Perfect Forward Security (PFS)**という考え方が注目されている。PFSとは、ある時刻に使用していた鍵（暗号化鍵、復号鍵）が漏洩しても、それ以前かつそれ以降の暗号の解読に影響を与えないことである。

つまり同じ鍵を使い続ける static DH および次節のRSAはPFSではなく、DHEはPFSであると言える。また楕円曲線関数を利用したDHEの拡張版である**楕円曲線ディフィー・ヘルマン鍵交換法**（**ECDHE**）もPFSである。

5.6 RSA暗号

5.6.1 RSA暗号とは

RSA 暗号は、1977 年に Ronald Rivest, Adi Shamir, Len Adleman らによって開発が行われた、公開鍵暗号方式による最初の暗号である。RSA という名称は開発者の 3 人の頭文字を由来としている。

Diffie-Hellman 鍵交換法とは異なり、メールなどのメッセージ暗号などに向いている。一方、リアルタイムでの通信に使用する場合には、処理に時間がかかるため、共通鍵の交換に RSA 暗号を使用し、その後は交換した共通鍵による共通鍵暗号を移用するのが一般的である（**ハイブリッド暗号方式**）。暗号の安全性（解読の困難さ）は、大きな数の素因数分解の困難さに依存しており、プログラム中で使用する素数が十分に大きくない場合には、有限時間内に暗号を解読することが可能となる。

なお、RSA 暗号のアルゴリズムの特許は 2000 年 9 月に失効している。

5.6.2 RSA暗号のアルゴリズム

RSA 暗号のアルゴリズムを以下に示す。なお、以下で使用する ^ は累乗を、**mod** は割り算での余りを意味する。

1) **ある大きな２つの素数 p, q を選んで n = p×q とする。**
2) **(p - 1)×(q - 1) 以下で (p - 1)×(q - 1) と互いに素の数 e を選ぶ。**
3) **(e×d) mod ((p - 1)×(q - 1)) = 1 となる整数 d を求めると (e, n) が公開鍵、(d, n)が秘密鍵となる**

この時、平文 M を暗号化するには C = M^e mod n とし、暗号文 C を復号するには M = C^d mod n とすれば良い。

5.6.3 RSA暗号の例題

以下にRSAの例題を示す。ただしこれは、非常に簡単化した（使用する素数が非常に小さい）例のため様々な制約が発生し、実際のRSAとはかなり違うことに注意されたい。

例えば以下の例ではブロック長は１であり、また暗号化・復号ではｎ(素数×素数)による余りを使用するため、使用するｎ以上の数を扱うことがでないという制約が発生している。そのため結局以下の例題の暗号化では、換字式暗号と同等になってしまっている。

【例題１ p=5, q=7 の場合】

p=5, q=7 とすると、n=35, (p-1)×(q-1)=24
24 以下で、24 と互いに素となる数 e=5 を決める。
(5×d) mod 24 = 1　となる d は 29　(d=5 でも可だが、公開鍵と同じになってしまう)
よって、**(5, 35) が公開鍵、(29, 35) が秘密鍵**

確認

5 を暗号化 5^5 mod 35 = 3125 mod 35 = 10
10 を復号 10^29 mod 35 = 5

33 を暗号化 33^5 mod 35 = 39135393 mod 35 = 3
3 を復号 3^29 mod 35 = 68630377364883 mod 35 = 33

【例題２ RSA 暗号の解読】

(e, n) が公開鍵として与えられているとする。n を因数分解して n=p×q を満足する p, q を求める。ここで、e×d mod (p - 1)×(q - 1) = 1 となる整数 d を求めると (d, n)が秘密鍵となる。すなわち、n を因数分解することができれば、容易に秘密鍵を求めることができる。

公開鍵が (7, 33) の場合。
33 を因数分解して、33 = 11×3。すなわち p=11、 q=3 となる。
(11 - 1)×(3 - 1) = 20　であるので、(7 × d) mod 20 = 1 となる d は 3 となる。
従って**秘密鍵は (3, 33)** である。

確認

5 を暗号化　5^7 mod 33 = 78125 mod 33 = 14
14 を復号 14^3 mod 33 = 2744 mod 33 = 5

20 を暗号化　20^7 mod 33 = 1280000000 mod 33 = 26
26 を復号　26^3 mod 33 = 17576 mod 33 = 20

5.8 サーバ認証とクライアント認証

5.8.1 HTTPS とサーバ証明書

証明書の例として HTTPS でのサーバ証明書を挙げる。Web サーバで HTTPS（暗号化通信）を使用する場合、サーバ側には**（SSL/TLS）サーバ証明書**をインストールする必要がある。このサーバ証明書を利用することにより、暗号化通信と Web ブラウザによるサーバの身元確認（URL の確認）が可能となる。

Web サーバ側では先ずペア鍵を生成し、それを使用して**証明書署名請求**（Certificate Signing Request：**CSR**）と呼ばれるデータを作り出し、認証局に送付する。この時重要となるのが**コモンネーム**と呼ばれるサーバのドメイン名である（例えば hogebar.jp など）。認証局側ではこの CSR に署名をして証明書を作成し、サーバ側に送り返す。この証明書が**サーバ証明書**（単にサーバ証明書とも呼ぶ）である。この時コモンネームはサーバ証明書の Subject 欄に記載される。

サーバ側ではこの証明書を所定の場所に設置し、Web サーバを稼働させる。もしサーバ証明書を発行した機関が中間認証局であるならば（ルート認証局でないならば）、Web サーバには**中間認証局証明書**も証明書チェーンとして設置しなければならない。

Web ブラウザ側では、HTTPS 通信のリクエストを出した場合に、サーバからこの証明書を受け取る。Web ブラウザはサーバ証明書のコモンネームが自分のリクエストしたドメイン名と同一であるか、また証明書にある認証局の署名が実際の認証局の署名と一致しているかを検証する（図 5.27）。なお実際の認証局の署名（証明書）は Web ブラウザ、もしくは OS に予めインストールされている（図 5.28）。

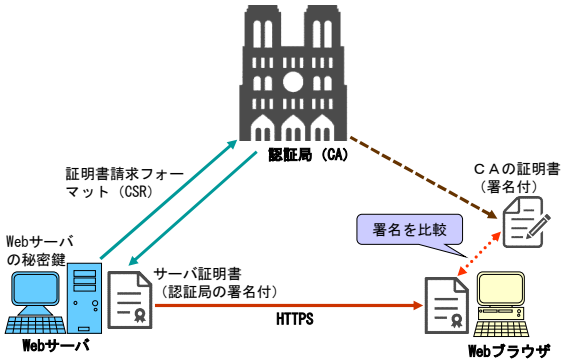

図 5.27 サーバ証明書

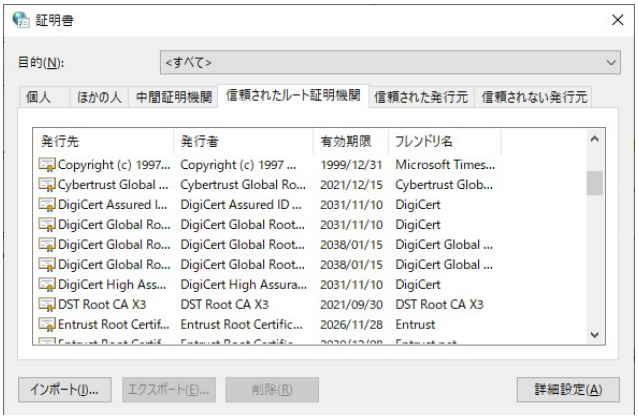

図 5.28 MS Windows に予めインストールされているルート証明書

サイトのドメイン名とコモンネームが一致しない、ルート認証局の証明書が一致しない（または存在しない）などの不備がある場合は、図 5.29 のような警告画面が表示される。初心者はこの画面を良くエラー画面と呼ぶが、これは警告画面であってエラー画面ではない。ユーザがそのサイトのサーバ証明書に不備があることを承知しているのなら、そのまま「詳細設定」のボタンの先からサイトに接続することもできる。

図 5.29 サーバ証明書に不備がある場合に表示される警告画面（Google Chrome）

不備なサーバ証明書としては、認証局ではなく自分で自分を認証する自己証明書（Self-Signed Certificate、俗に**オレオレ証明書**とも呼ばれる）や証明書の期限が切れている場合なども多い。

一方認証局側でサーバ側からの CSR に署名を行う場合、本来であればサーバ側の組織の実態などを検査するべきであるが、実際にはそうなっていない。実情では申請した Web サーバの存在が確認され、料金さえ支払われれば、それだけでどんなに怪しいサイトであってもサーバ証明書を発行してくれる場合が多い。故に正式なサーバ証明書を持っているからと言って、不正なサイトでないとの補償には全くならない。逆に不備のある証明書を使用しているからと言って、不正なサイトとは限らない（単に証明書を買うお金が無いだけかもしれない）。

そのため現在ではサイトの実態をより厳密に調査し、不正なサイトに証明書を発行しないようにする **EV SSL 証明書**というのもある。ただし当然発行手数料は高くなる。

また米国の非営利団体である ISRG（Internet Security Research Group）では **Let’s Encrypt** と呼ばれる有効期間 90 日の SSL サーバ証明書を無料で発行している（ISRG 以外にも、無料のサーバ証明書の発行を行っている団体がい

くつか存在する)。最近では Let’s Encrypt のサーバ証明書の発行は、ほぼ自動化されており、容易に設定が可能となっている（サーバ証明書の更新も自動化できる)。

認証局は発行したサーバ証明書について、有効期間内であっても色々な事情からその効力を取り消す場合がある。その場合、認証局は**証明書失効リスト**（Certificate Revocation List: **CRL**)と呼ばれるリストをユーザに配布し、無効な証明書をユーザに通知する(通常はWebブラウザやOSのアップデート時に同時に配布される)。また **OCSP**（Online Certificate Status Protocol）と呼ばれる通信プロトコルでも証明書の失効状態を確認することができる。OCSPはCRLに比べて取り扱う情報量が少ないので、証明書の失効状態を迅速かつタイムリーに得ることができる。

5.8.2 サーバ証明書の申請

実際にWebサーバ用にサーバ証明書を発行して貰う場合の手順を紹介する。まずWebサーバ上で openssl genrsa コマンドにより公開鍵暗号RSAのペアを作成する（図5.30)。カレントディレクトリに key.pem が生成されるが、key.pemはPEM形式で、この中に素数pとq、およびその積n、 公開鍵（の一部）e、 秘密鍵（の一部）d 等が含まれている。key.pem には秘密鍵が含まれるので、通常は第三者がアクセスできないディレクトリに保存する。なお図5.30の最後の e（65537: 0x010001）はRSAの公開鍵（e, n）の e を表す（5.6.2 項参照)。OpenSSL の実装では高速に計算が行えるように、公開鍵はあまり大きな数を使用しないようになっている。

サーバ証明書の発行依頼手続きとは直接関係ないが、ここで key.pem の中身を確認してみる。図5.31 の openssl rsa コマンドより key.pem の内容を確認できる。図中のコロン : で区切られた数字は 16 進数で、例えば 00:b2:1f は 0x00b21f を表す。またmodulus, publicExponent, ……, coefficientの意味を図5.32に示す。

ここで d は (e*d)mod((p-1)*(q-1) = 1 を満たす数で（d, n)が秘密鍵とな

る。また exponent1, exponent2, coefficient は以後の計算を効率よく行うためのもので、cofficient c は (q*c) mod p = 1 を満たす数である。

ペア鍵の生成が終了したら、次に証明書署名請求 (CSR) の作成を行う。作成は図 5.33 のように openssl req コマンドで行う。図 5.33 で太字かつイタリックの部分が入力値であるが、Common Name (コモンネーム)、メールアドレス以外はあまり重要ではない。特にコモンネームにはサーバのドメイン名 (通常は FQDN) を指定するが、間違えた場合は全く用をなさない証明書が発行されるため注意が必要である。

図 5.33 のコマンドによりカレントディレクトリに csr.pem ができるので、これを認証業者にメールや Web からの投稿などで送れば良い。折り返し証明書が送られて来る筈である。

ここでは自己証明書 (オレオレ証明書) の作成の具仕方は省略するが、この csr.pem に openssl ca コマンド等を使用して自分で署名してサーバ証明書を作成することも可能である。

```
# openssl genrsa -out key.pem 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....................................+++++
.........+++++
e is 65537 (0x010001)
```

図 5.30 openssl genrsa コマンドによる RSA ペア鍵の生成

```
# openssl rsa -in key.pem -text -noout
RSA Private-Key: (1024 bit, 2 primes)
modulus:
    00:b2:1f:de:d7:a3:54:f0:17:d9:be:cb:70:7d:46:
    ..............
publicExponent: 65537 (0x10001)
privateExponent:
    7e:93:8d:3c:79:31:83:87:bf:51:08:aa:40:2b:97:
    ..............
prime1:
    00:dd:3d:58:6e:6b:03:4d:0c:64:ee:7d:cc:c6:5c:
    ..............
prime2:
    00:ce:1c:5b:61:8d:b4:8c:07:28:66:be:e9:f2:5e:
    ..............
exponent1:
```

```
    00:c2:4c:21:e1:b7:31:ca:f4:db:9f:67:f3:f3:31:
    ..............
exponent2:
    1f:08:ce:09:a6:58:a5:2c:fe:bc:59:ca:c8:1f:cc:
    ..............
coefficient:
    00:b3:9d:42:f0:c2:9a:83:dc:fd:a8:8e:c5:dd:0d:
    ..............
```

図 5.31 openssl rsa コマンドによる鍵の確認

modulus	n = p*q
publicExponent	e = 65537 (=0x10001)
privateExponent	d: (e*d) mod ((p-1)*(q-1)) = 1
prime1	p
prime2	q
exponent1	d mod (p-1)
exponent2	d mod (q-1)
coefficient	c: (q*c) mod p = 1

図 5.32 図 5.31 の項目の意味

```
# openssl req -new -key key.pem -out csr.pem
.....
Country Name (2 letter code) [AU]:JP          (国名)
State or Province Name (full name) [Some-State]:Chiba      (県名)
Locality Name (eg, city) []:Chiba          (市名)
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NetSec    (組織名)
Organizational Unit Name (eg, section) []:HogeBar    (部署名)
Common Name (e.g. server FQDN or YOUR name) []:www.hogebar.jp          (FQDN)
Email Address []:iseki@hogebar.jp          (メールアドレス)

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: (enter)
An optional company name []: (enter)
```

図 5.33 CSR の作成. (enter) はエンターキーの入力を意味する.

5.8.3 クライアント認証

図 5.27 では Web ブラウザ（クライアント）が Web サーバを認証（チェック）しているが、逆にサーバがクライアントを認証（チェック）することも可能で

ある（お互いに認証することも可能）。サーバがクライアントを認証する事を**クライアント認証**と呼び、クライアントには**クライアント証明書**をインストールする必要がある。クライアント認証はサーバが特定のクライアントのみに接続を許可したい場合などに用いる。

5.9 TLS（SSL）

5.9.1 TLS（SSL）とは

SSL(Secure Socket Layer)は当初 WWW で暗号化通信（HTTPS）を行うために、Netscape Communications 社（Netscape ブラウザを開発）が開発した、公開暗号を利用した暗号化技術である（現在は HTTPS 以外でも使用可能）。バージョン１は公開前に脆弱性が発見され、バージョン 2（1994 年）も公開後程なくして脆弱性が発見された。1995 年にバージョン 3（**SSL3**）が公開されたが、これを若干修正して 1999 年に **TLS1.0**(Transport Layer Security 1.0)として標準化されている。

その後しばらく SSL と TLS は併用されたが、2014 年頃から SSL3 に実装上の問題点がいくつか見つかり（POODLE, Heartbleed Bug 等）、現在では SSL3 の使用も非推奨となっている。現時点では **TLS1.3**（2018 年）が TLS の最新バージョンであるが、セキュリティ問題に直結するようなシステムでは、その時点での最新版の TLS を使うべきである。

一方 SSL という言葉は、**SSL/TLS** のように未だに色々な用語に残っているが、実際に使用されているのは多くの場合 TLS である（推奨に逆らって SSL3 を使おうと思えば使えないこともない）。先に述べたように HTTPS も TLS の使用が推奨され、SSL3 はほとんど使用されていない。