画像内暗号を用いた C/S 型
ログインモジュールの開発

島根大学　総合理工学部　数理・情報システム学科
計算機科学講座　田中研究室

s99449-F　田辺知里

2003 年 2 月 12 日

目次

第１章　序論

近年、社会の情報化とネットワーク化が進み、Web 上で情報をやりとりすることは日常的となっている。それに伴い、安全性の高い通信システムが望まれるようになってきた。

情報の秘密性を保ったまま通信をするには、暗号を用いることが一般的である。しかし、暗号化した情報をやりとりすることで、通信している情報の内容を解読されることはなくても秘密データの存在を隠すことは出来ず、かえって秘密情報に関する関心をかき立ててしまうことになる。したがって、重要な情報があること自体も秘密にできた方が情報の秘匿性は高いといえる。そこで注目されるのが Steganography（ステガノグラフィ）という技術である。

Steganography とは、他人に見られたくない秘密情報を何か別のデータの中に埋め込み、その秘密の存在そのものを見えなくしてしまう技術のことである。表に見えるデータは秘密を人目にさらさないためのダミー情報であるので、万一他人の手に渡っても、一見すると普通の媒体と変わりなく、秘密が漏れる心配はない。

この情報セキュリティ技術を使用して、本研究では実際に画素置換型 Steganography を用いて画像に秘密データ（パスワード）を隠蔽し、それを使用してサーバにログインするシステムを開発した。

第２章　情報の隠蔽とその目的

本章では、情報の隠蔽の必要性とその方法にはどのようなものがあるかについて述べる。

2.1 Steganography と電子透かし

2.1.1 Steganography とは

ステガノグラフィ（Steganography）とは、情報隠匿技術の一部をなすものであり、“ 他人に気づかれる事なく秘密情報を別の情報データに埋め込んで保存したり、それを特定の相手にこっそりと伝える技術 ” のことである。

ステガノグラフィということばの語源はギリシャ語に由来するといわれており、「こっそり書き込んだもの」との意味合いを持つ。秘密情報を他人に分からないように何か別のデータに埋め込んで伝えることにより、秘密通信の存在自体を隠すことができる。

ステガノグラフィで重要なのは隠された秘密データである。表に見えるデータは、秘密を隠すためのダミー情報であるため、万一誰かに見られても秘密が露見する心配はない。

ステガノグラフィに相当する日本語としては、“ 画像深層暗号 ” や “ 画像内暗号 ”、“ 情報非可視化 ”、“ 情報迷彩 ” などが使われる[1]。

2.1.2 電子透かしとは

電子透かし（Digital Watermark）とは、デジタルコンテンツを不正利用から保護するために、人に気づかれないように著作権情報などを示すマーク（情報）を、画像や音声などのデジタルコンテンツにそっと忍ばせる技術のことである。

例えば、紙幣には “ 透かし ” 入れられており、それがあることによって紙幣の真正性が証明できる。この発想をデジタルコンテンツに取り入れたものが電子透かしである。[3]

デジタルコンテンツに正当性を主張する情報を埋め込んでおくことによって、不正コピーの抑止や、不正利用者に対して埋め込まれた情報を証拠に著作権の主張ができる。

電子透かしの発想は、古くから情報セキュリティの世界で使われていた通信手段を秘密にするステガノグラフィと同じ性格のものであり、それが電子メディアの時代にふさわしい形態で再登場したものと言える[2]。

2.1.3 電子透かしと Steganography の違い

電子透かしもステガノグラフィも、画像や音声などのデータに別の情報を目立たないように埋め込む技術である。しかしながら、その目的は異なっている。

両者の違いを表 2-1 に示す。

表 2-1　ステガノグラフィと電子透かしの違い

	ステガノグラフィ	電子透かし
価値のある情報とは	埋め込まれた情報	外に見えている情報（画像や音楽データなどの作品）
埋め込みデータの頑強さ	外に見える情報に手を加えて情報が壊れても構わない	どのような処理をしても取り除けないことが必要
埋め込みデータが復元できる為の条件	埋め込み前のダミーデータを参照しない	埋め込み前の作品データを参照してもよい
埋め込み容量	なるべく大きいことが望ましい	少量の目印情報が埋め込めればよい

(引用文献[1]より)

電子透かしは、著作権の保護や不正コピーのコントロールが目的であるので、実際に見えるデータ（画像や音楽データなどの作品）に価値があり、また、埋め込んだデータはその著作権などを明示するためのものであるから簡単に取り除かれてはならない。

一方、ステガノグラフィは秘密通信が目的である。表に見えるデータ（これをダミーデータまたはダミー情報と呼ぶことにする）は、秘密情報を人目にさらさないためのダミー情報であるのであまり価値がない。本当に意味があるのは埋め込まれている情報で、これを秘密裏に相手に伝えるためにダミーデータに埋め込んで表面上偽装し通信する。したがって、秘密の隠し場所はできるだけ大きいことが必要とされる。

また、ステガノグラフィでは、たまたま秘密情報が送信中に第三者によって消去されても、再度送信すればよいだけであるのであまり大きな損失にはならない。問題なのは秘密が露見することであり、それよりは消されてしまった方がかえって安全だからである。

このように、ステガノグラフィと電子透かしは、技術的には同じでも目的はまったく逆のものであることが分かる[1][3]。

2.1.4　情報隠蔽の原理

電子透かしもステガノグラフィも同様の技術であることはすでに示した。

本節では、どのような原理に基づいて情報が埋め込まれるのかということを述べる。

① ダミーデータの変更：

ダミーデータの一部を埋め込みデータに基づき変更する。

（ダミーデータとは秘密情報を埋め込む対象のことである）

② デジタルコンテンツの冗長性の利用：

画像や音声データの一部を変更しても、冗長性の高いコンテンツの場合は全体的な構成に影響を及ぼさないことを利用し、データの一部を書き換える。

③ 生理学的特徴の利用：

上記②で変更する場合、さらに人間の感覚の特性を上手く利用して、人が気付き難い部分のデータを変更していくことにより、埋め込む。

（引用文献[3]より）

①の場合、データの一部を変更する際に、人の感覚で気付く範囲で行っては意味がない。なぜならば、電子透かしの場合、画像などの作品上に変更の跡が見られるのは好ましくなく、ステガノグラフィの場合は秘密が埋め込まれているということ自体を隠すことが目的であるため、痕跡が残っては問題外である。

原理②は、もともとデジタル画像やデジタル音声に含まれている冗長部分を利用しようというものである。画像でいえば、冗長性というのは画像の骨格をなす部分ではなく、彩りを添えている部分に相当する。冗長性が高い場合、わずかな量の情報を変更しても、画像全体に決定的な影響を与える怖れはない。

原理③は人間の感覚のルーズさを利用して情報を埋め込もうというものである。例えば、人間は輝度情報に対する感度より色に対する感度が低いという特徴がある。これらの生理学的特徴を利用して埋め込みを行う。

2.2 Steganography の種類

情報を埋め込む方法としては、以下のようなものがある。

- 画素置換型 Steganography
- 周波数特性を利用した Steganography
- 色情報を利用した Steganography

本研究では、画像への Steganography を利用するため、以下の節では画像をキャリアとして各方法について説明する。

2.2.1 画素置換型 Steganography

多値画像（白黒濃淡画像やカラー画像）へのステガノグラフィで有名なのが「画素置換型 Steganography」である。これは明るさ情報を利用した埋め込み方法で、人の視覚の特性と画像の冗長性を利用する。

カラー画像の場合だと、R（赤）、G（緑）、B（青）の各成分の強さをそれぞれ８ビットで表現することが多い。Windows での BITMAP ファイルや UNIX での PPM ファイルなどがその例である[1]。

ここでは、そのような形式での画像データの第１ビット目を最上位、第８ビット目を最下位ビットとして扱うことにする。各画素の LSB（least significant bit）を集めたものをプレーン０（最下位）とし、MSB（most significant bit）を集めたものをプレーン７（最

上位）とすると、画像を８枚のビットプレーンに分けることができる。（図 2-1）

画素置換型というのは、プレーンそのものを秘密情報と置き換える方法である。最上位ビットがほぼ画像の特徴を反映しているのに対し、白黒濃淡画像やカラー画像の最下位ビットは画像全体への視覚的影響が少なく、別のデータと置き換えても変化が少ない。このことを利用して最下位ビットに秘密を埋め込んでも、視覚的な埋め込みの証拠は残らず、秘密データを隠すことができる。この最下位ビットに埋め込む方法は、典型的なステガノグラフィ方式として知られており、この原理は一次元時系列データである音楽データなどへも適用できる[2][3]。

例えばカラーの場合は、先にも述べたように各画素が RGB の３バイトで構成される。この各成分の最下位ビットに埋め込み処理を行うことになる。

この方式には以下のような問題点がある。

① この方式で埋め込めるデータの量は、ダミー画像ファイルサイズの１／８を超えることはない。最下位だけでなくその次のビットに埋め込んでも画質が劣化しない場合があり、その場合の埋め込み容量はダミー画像の１／４程度になるが、いずれにしても埋め込み容量は少ない。
② この方式のように、“ 定まった部分 ” に情報を埋め込む方式は、埋め込みの事実が比較的容易に見破られる恐れがある。
③ 画像データの圧縮などの際に、埋め込み情報が失われる可能性がある。

①は、本研究のように画像に８文字程度のパスワードを埋め込むという場合に使用するには何の問題もない。しかし、秘密通信を行う場合に大量のデータを隠すには、サイズの大きなダミー画像を用意するか、または小分けして隠蔽しなければならなくなり問題となる。②については、ステガノグラフィは本来、秘密が隠されていること自体を秘密にできる技術であるので、心配する必要はないかに思われる。しかし、万一秘密の存在を知られた場合や、無作為に画像を選んで抽出処理が行われた場合にも、容易に抽出を行えないように対策を立てるべきである。例えば、秘密データの埋め込み場所を一定にせず、ビットプレーンに離散的に散布することで対処できる。また、画像の圧縮の対象となる成分は、視覚への影響力の少ない下位のビットプレーンを構成する要素である。よって、下位のビットプレーンに埋め込まれた秘密データは失われる可能性がある。

このように、画素置換型には多くの問題点があるが、埋め込み処理が簡単で分かりやすいために、よく使われる手法のひとつとなっている。

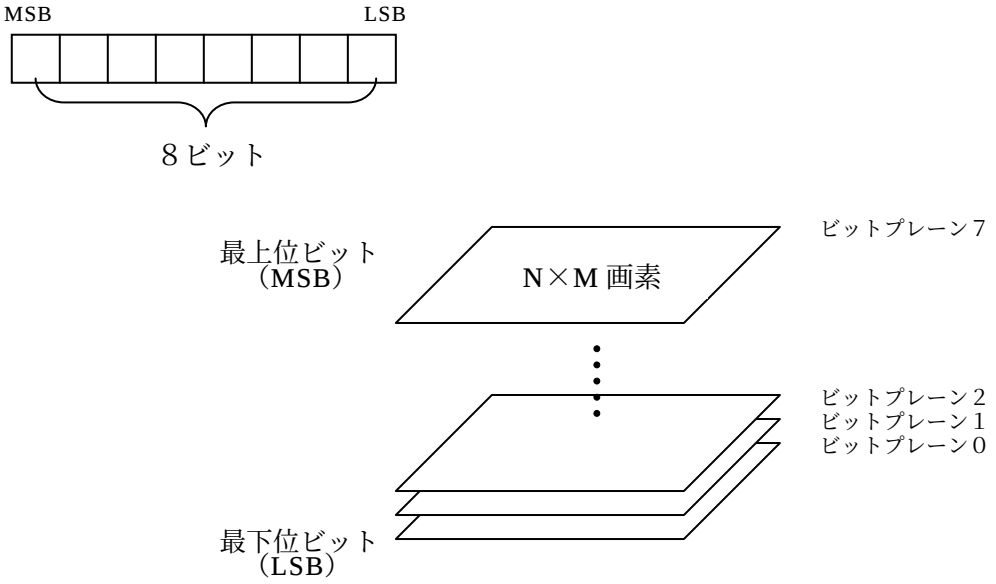


図 2-1　ビットプレーンの考え方（引用文献[3]より）

2.2.2　周波数特性を利用した Steganography

今日の電子透かしは、画像を周波数領域へ変換してから情報を埋め込む方法が主流となっている。これは、画像や音声を周波数成分に変換し、特定の周波数成分に秘密情報を埋め込む方式である。

周波数領域にすると画像の特徴が把握しやすくなり、画像の骨格を形成しているのがどの周波数で、どの部分は無駄であるかということが分かりやすくなる。

画像や音声データは、フーリエ変換（Fourie transform）やスペクトル拡散、離散コサイン変換（Discrete Cosine Transform）などを使い、周波数成分に変換して扱うことができる。そのような変換の際に、画質に影響しないように、ダミーデータの特定周波数帯域の成分を秘密データと置き換える方式である[3]。

低周波成分は画像情報そのものを表し、高周波成分は圧縮などで消えてしまうため、中間周波数成分の一部を置き換えるような方法がとられる。

2.2.3　色情報を利用した Steganography

色情報を利用した埋め込みといえば対象はカラー画像であるが、カラー画像に対する Steganography も、実際には明るさ情報を利用することが多い。その場合には、RGB 信号から輝度信号＋色信号の形の信号を作り、輝度信号を利用する。

先にも述べたように、人間の目は明るさより色に対して感度が低いという特徴がある。この特性と、カラー画像は色成分だけ冗長性が増加している、という特徴により色情報を操作すると秘密情報が比較的埋め込みやすくなる[3]。

具体的な方法としては、

① 全画素について色をわずかに変更する方法
② 秘密情報の部分だけパラメータを変更する方法

がある。①については、全画素では１ビットしかデータを埋め込めないため、いくつかのブロックに分割する方法が考えられている。また②では、表色変換するときに全ての画素に対して同じ変換をするのではなく、秘密情報の部分だけ変換パラメータを少し変えて逆変換処理を行う。これは、基本的に表色変換系の係数を秘密データに基づいて変更する方式となっている。埋め込まれる秘密データは画像のみとなり、文字情報は埋め込む事ができない。本研究では秘密データとしてパスワード（文字列）の埋め込みを行う。よって、この方法は使用することができない。

第３章　画像への情報の隠蔽・抽出方法について

本研究では、画素置換型 Steganography を利用してパスワードが埋め込まれた画像を作成し、ログインの際に使用した。本章では、実際にパスワードを画像に隠蔽する方法について述べる。

3.1 ビットマップファイルについて

パスワードを隠蔽するための画像として、ビットマップファイルを使用した。

ビットマップは Windows が標準でサポートしている画像形式である。白黒（2 値）の画像からフルカラー（1677 万 7216 色）までの色数を指定できる[4]。

通常、圧縮されていないため、画像ファイルの大きさはかなり大きくなるが、編集や加工がしやすく、プログラムも組みやすい。（まれに、イメージデータが圧縮（ランレングス圧縮）されているものもある）

3.1.1　DIB と DDB

Windows のビットマップには２種類あり、デバイス依存型を DDB（Device Dependent Bitmap）といい、デバイス独立型を DIB（Device Independent Bitmap）と呼ぶ。

DDB とはデバイスの仕組みや性能に依存するビットマップのことである。デバイスの性能や現在の状態によって、扱うことのできる色の種類や数などが制限されてしまう。また、ピクセルのビット値がどのような意味を持つのかに関してもデバイスに依存するため、ピクセルのビット値を取得しただけでは、それが特定の色を一意的に表すと決め付けることはできない。DDB は古い形式のビットマップであり、同一のコンピュータ間では転送可能だが、デバイス依存性により Disk、Modem での転送は不可能である。このため DIB に比べ柔軟性に欠ける[5]。

DIB はその名のとおり、デバイスに依存しないビットマップである。DIB には画像情報に加えて、そのビットマップで利用されているカラーパレットが組み込まれているため、どのような環境下でも同じ表示が得られるようになっている。機器に依存しない画像形式であり、現在の BMP 形式は実際にはそのほとんどが DIB 形式である[4]。

DIB には、イメージデータの格納の仕方で bottom-upDIB と top-downDIB とがあり、通常はイメージをしたから格納する bottom-upDIB が使用される。

DIB は、ピクセルの色が単純な配列としてまとめられており、プログラムの中で直接配列を扱うのと同じような感じで扱うことが可能である。Windows の標準グラフィック形式である BMP ファイルは、DIB の前にヘッダをつけた構造になっている[5]。

3.1.2　BMP ファイルの構造

Microsoft 社の MSDN ライブラリ[6]によると、ビットマップファイルは以下のような構造になっている。

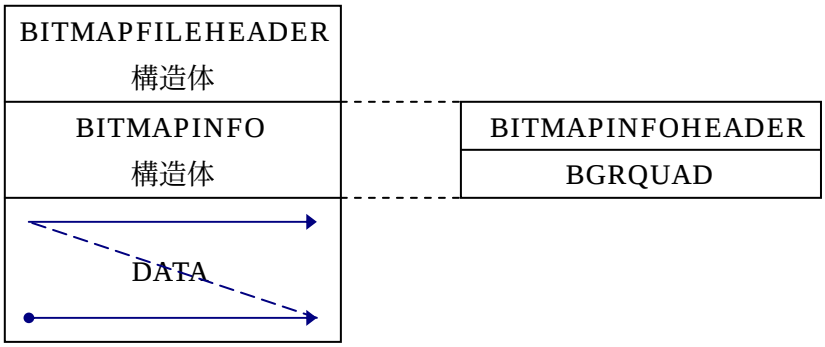


図 3-1　ビットマップの構造

●BITMAPFILEHEADER 構造体

ビットマップファイルである証明など、このファイル自体に関する情報が格納されている。先頭 2byte には、BMP ファイルであることを示す文字列“ BM ”（0x4d42）が入っている。

●BITMAPINFO 構造体

BITMAPINFOHEADER 構造体と BGRQURD からなる。

・BITMAPINFOHEADER 構造体

画像のサイズ、色深度など、画像自体に関する情報が格納されている。

・RGBQUAD

色深度が 2、16、256 色の場合、パレットが存在し、その連続番号に対応する RGB 値が記録されている。今回使用するのはフルカラー（24bit）である。フルカラーの場合、RGB の輝度がそれぞれのピクセルについて格納されているため、パレットは存在しない。

●DATA（ピクセルデータ）（フルカラーの bottom-upDIB の場合）

フルカラービットマップは、上位から BGR の順に、１バイトずつ計３バイトで１ピクセルを形成しており、このピクセルデータが、左から右、下から上へとボトムアップ方式で格納されている。（実際のイメージが上下さかさまに格納されている。

また、１行のサイズは DWORD（4byte）単位でなければならない。行サイズが４の倍数でない場合は、１行が４の倍数になるようにビットを末尾に追加する。この部分は０で埋められる。（この部分は表示しない）

※ top-downDIB の場合、イメージの高さを示す biHeight がマイナス値になっている。

（例）パディングの例

５×５ピクセルの場合、

１行のバイト数は

５×３＝１５byte

である。１行のサイズは４バイト単位でなければならないため、１バイト分ビットを追加する必要がある。

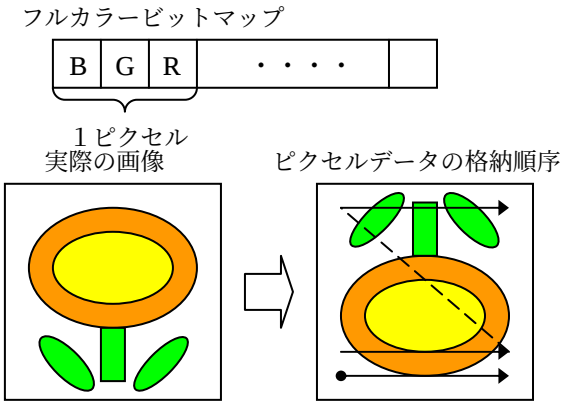


図 3-2　ピクセルデータの並びと格納順序
（フルカラーbottom-upDIB の場合）

3.1.3　各構造体の要素

ビットマップの各構造体に含まれる要素について説明する[6]。

● **BITMAPFILEHEADER（ファイルヘッダ）**

WORD	bfType	ファイルタイプ（“ BM ”が入っている）
DWORD	bfSize	ファイルサイズ （byte 単位）
WORD	bfReserved1	０で予約
WORD	bfReserved2	０で予約
DWORD	bfOffBits	ビットマップデータへのオフセット（ファイルの先頭からビットマップデータの先頭までのバイト数）

● **BITMAPINFOHEADER（情報ヘッダ）**

DWORD	biSize	この構造体のサイズ （byte 単位）
LONG	biWidth	イメージの幅 （ピクセル単位）
LONG	biHeight	イメージの高さ （ピクセル単位）
WORD	biplanes	プレーン数。常に１
WORD	biBitCount	色深度（１ピクセル当たりの bit 数）
DWORD	biCompression	圧縮タイプ
DWORD	biSizeImage	イメージデータのサイズ（byte 単位）
LONG	biXPelsPerMeter	水平解像度
LONG	biYPelsPerMeter	垂直解像度
DWORD	biClrUsed	実使用カラーインデックス数
DWORD	biClrImportant	重要カラーインデックス数

● **RGBQUAD（パレット）**：色深度が１６bit 以上ならば、RGBQUAD はない）

BYTE	rgbBlue	青輝度
BYTE	rgbGreen	緑輝度
BYTE	rgbRed	赤輝度
BYTE	rgbReserved	０予約

BYTE：符号なし 8bit 値　　LONG：符号付き 16bit 値
WORD：符号なし 16bit 値　　DWORD：符号なし 32bit 値

※　実際には、アライメントの問題でパティングが入っている可能性があるので、これらの構造体の長さは単純にそれぞれのデータのバイト数を足したものにはならないことがある。

以上の要素のうち、プログラムで使用したものについて説明する。

bfType：BMP ファイルを読み込むときに、まずファイルタイプを調べる。
ここが“ BM ”なら BMP ファイルである。

biBitCount：フルカラーかどうかを調べる。
ここが 24 ならばフルカラーである。

biWidth と biHeight：隠蔽できる容量を調べるために使用する。
これらの掛け算でイメージサイズが分かる。

bfOffBits：ファイルの先頭からイメージデータまでのビット数。
イメージデータの正確な位置を調べるために使用する。

3.2 画素置換型 Steganography を利用した隠蔽方法について

3.2.1 画素値への隠蔽・抽出方法

ここでは、実際にどのように秘密情報を画像へ埋め込むのか、その方法と実装について説明する。なお、隠蔽には画素置換型 Steganography を用いる。

秘密情報として埋め込むものは、人の名前や ID、日記、画像など何でもよいが、本研究ではサーバにログインするためのパスワードを隠蔽する。

3.2.2 隠蔽アルゴリズム

まず、秘密データを画像へ隠蔽するアルゴリズムについて説明する。

埋め込む情報は、文字であれ画像であれビット０か１のいずれかとなる。例えば大文字の“ A ”はアスキーコードで 97（10 進数）であり、2 進表記では 01100001 である。これを最下位ビットから１ビットずつ埋め込んでいくことになる[3]。

秘密データの埋め込み規則で最も簡単な方法は、画像のデータを秘密データの０、１に

よって変更するというものである。ここでは、以下のような規則を用いてダミー画像にデータを隠蔽することとする。

[埋め込み規則]

・ 秘密データをダミーデータの最下位ビットに１ビットずつ埋め込む。（ダミーデータの最下位ビットを、埋め込む秘密データのビット値に書き換える）

以下に隠蔽アルゴリズムの流れ図を示す。

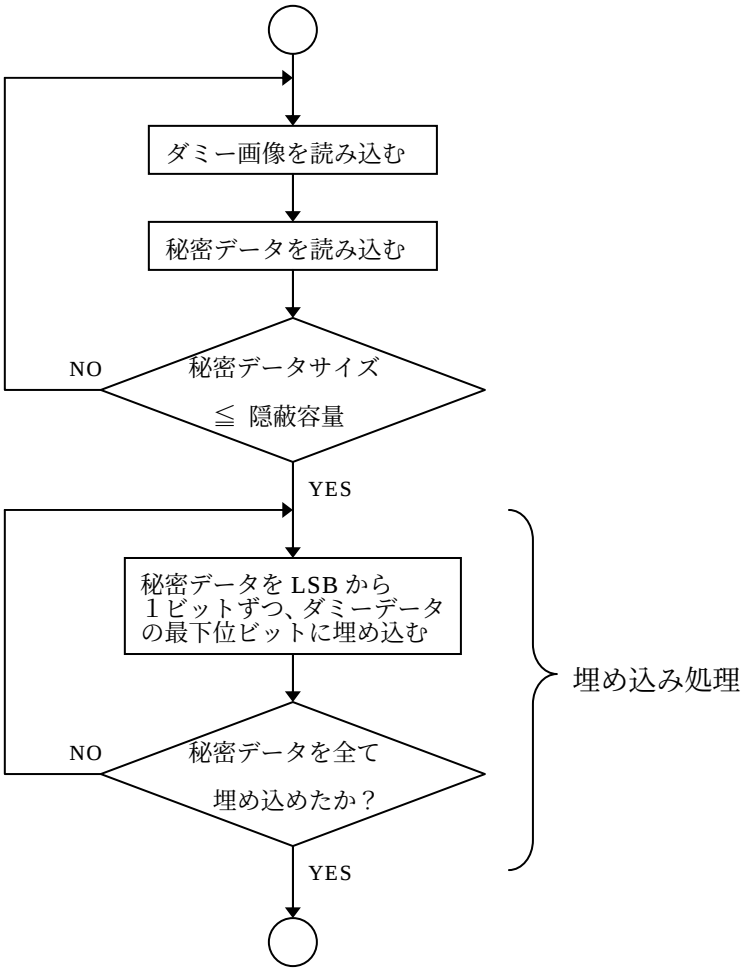


図 3-3　隠蔽処理の流れ

１．ビットマップ画像（ダミーデータ）を読み込み、画像のヘッダ情報（データのサイズ等）を取得する。ここで、BMP ファイルかどうかとフルカラーかどうかを調べておく。違ったら別の画像に変更する。（今回使用するのはフルカラービットマップ）

２．読み込んだ画像に隠蔽できる秘密データの容量（byte 単位）を求める。

隠蔽容量 = ダミー画像の幅 × ダミー画像の高さ × ３ ／ ８

（biWidth）　（biHeight）

biWidth、biHeight は BITMAPINFOHEADER を見ることにより分かる。これらの値はピクセル単位である。今回使用するのはフルカラーBMP であり、各ピクセルが RGB の３バイトで構成されているので、３倍することにより隠蔽容量がビット単位で求まる。これを８で割り、バイト単位で求めておく。

３．秘密データを読み込み、隠蔽容量と比較。
読み込んだ秘密データが隠蔽容量以下なら４へ。それより大きければ埋め込めないので別のダミーデータを探す。（１へ戻る）

４．埋め込み処理。
秘密データを８ビット単位にし、最下位から１ビットずつ右シフトをしながらダミーデータの RGB 値の下位１ビットに書き込んでいく。
（ダミー画像のイメージデータ部分に埋め込みを行う）
この処理を秘密情報を全部書き込む（隠蔽する）まで繰り返す。

3.2.3　抽出アルゴリズム

隠蔽された秘密データの抽出方法について説明する。

ダミーデータから秘密データを抽出するには、埋め込み処理の逆を行えばよい。秘密情報は隠蔽後画像の RGB 値の最下位ビットに埋め込まれている。これらを取り出し、８ビットずつをまとめてデータを構成していく。

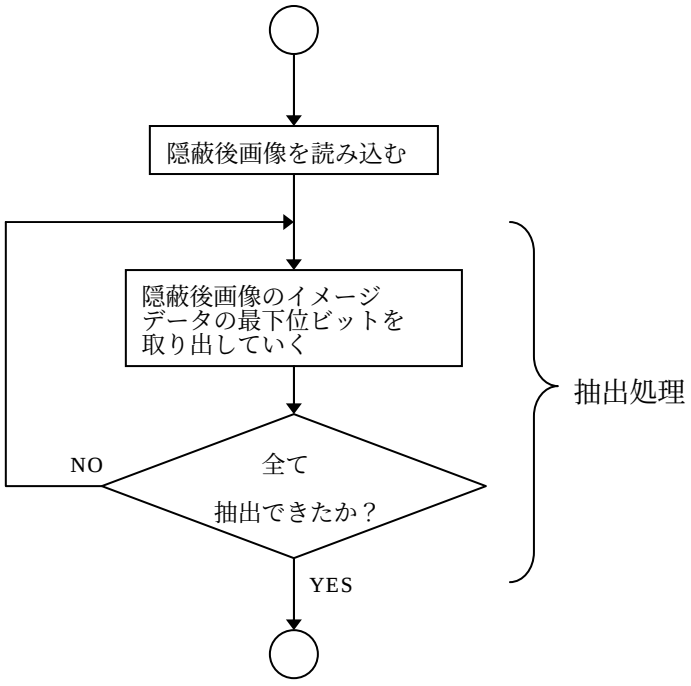


図 3-4　抽出処理の流れ

１．隠蔽後画像を読み込む。

２．抽出処理。

画像データの最下位ビットを取り出し、８個まとめてデータを構成する。隠蔽の逆で左シフトをしながら秘密（抽出）データを格納していく。

秘密データは最下位ビットから埋め込まれているため、取り出されるのも最下位ビットからとなる。データを構成する際にこのことに気をつける必要がある。

3.3　作成した隠蔽・抽出プログラムについて

本節では、実際に作成した隠蔽・抽出プログラムについて説明する。

3.3.1　概要

画像へのパスワードの隠蔽を行うために、隠蔽・抽出処理を行うプログラムを作成した。パスワード長は８文字に固定されている。隠蔽方法には 3.2 節で紹介した方法を用いる。

画像には、秘密情報とそれを抽出するために必要になる情報を、以下の順で隠蔽する。

隠蔽キー	パスワード長	パスワード

図 3-5　埋め込む情報

隠蔽キーとは、ダミーデータにパスワードが埋め込まれている事をサーバに知らせるための制御信号である。サーバはこのキーワードが埋め込まれていることを確認して、パスワードの抽出処理に移る。隠蔽キーは統一のものを使用し、今回は“ testpass ”とした。

隠蔽キーの後にはパスワードの長さが埋め込まれている。埋め込まれている情報を抽出する際に、抽出処理を行う範囲を知る必要がある。今回はパスワード長を８文字に固定しているためこの部分は必要ないが、文字列を隠蔽するプログラムと考えた場合に、長文でも隠蔽できるように拡張することを前提にこのような方式をとった。また、画像を隠蔽する場合には、この部分は画像のファイルサイズを隠蔽するために使用される。そのあとにパスワードを埋め込む。

必ずしもこのような方式をとる必要はなく、例えば、秘密データを埋め込んだ後になんらかの終わりを示す印を埋め込んでおくことによって、隠蔽範囲を知るということもできる。この場合は、抽出処理を行いながら、常に抽出処理の終了判定を行う必要がある。

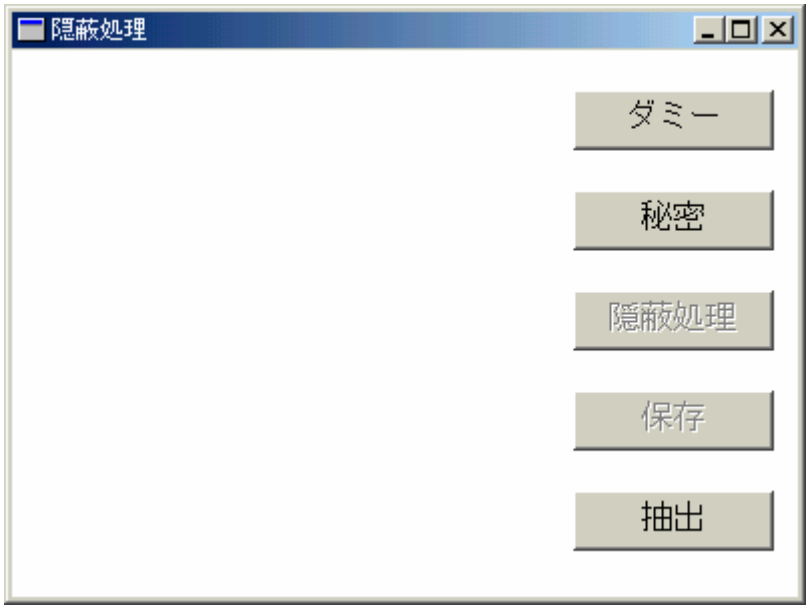
図 3-5 の情報を、3.2.2 節で説明した方法によってダミー画像に埋め込んでいく。

3.3.2　実行結果

作成したプログラムで実際に画像にパスワードを隠蔽してみる。ここで使用する環境はネットワークに接続されている必要はない。

隠蔽処理手順

１．隠蔽・抽出プログラムを起動すると、以下のようなウインドウが開く。



各ボタンの機能は以下のようになっている。

ダミー　：秘密データを埋め込む画像（ダミー画像）読み込みボタン
秘密　　：秘密データ（文字列）読み込みボタン
隠蔽処理：隠蔽処理を行うボタン
保存　　：秘密を埋め込んだ画像を保存するボタン
抽出　　：秘密が埋め込まれた画像を読み込み、抽出を行うためのボタン

２．ダミー画像（パスワードを埋め込む画像）を読み込む。

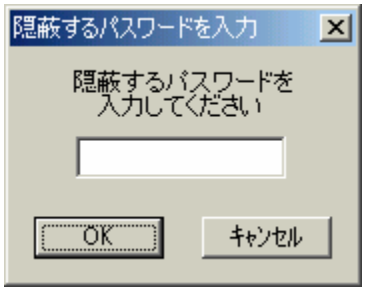
ダミー画像の読み込みには以下の２つの方法がある。

① ダミーボタンを押すと「ファイルを開くダイアログ」が表示される。ダミー画像として使用する画像を選んで OK を押す。

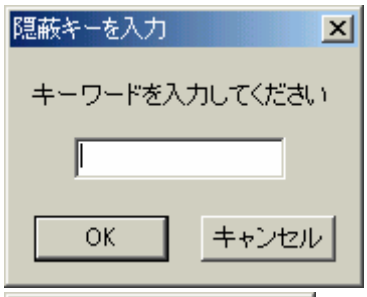
② ウインドウにダミー画像をドラッグ＆ドロップして読み込む。

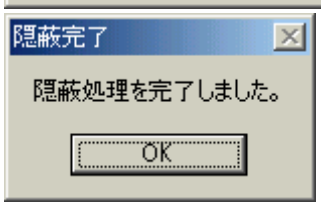
読み込んだ画像はウインドウに表示される。

3．次に、秘密データを読み込む。秘密ボタンを押すと、以下のようなダイアログが表示されるので、ここで隠蔽するパスワードを入力し、OK を押す。ここで隠蔽処理ボタンが有効になる。



4．隠蔽処理ボタンを押すと以下のダイアログが表示される。
隠蔽キーを入力して OK を押すと隠蔽の完了を通知し、保存ボタンが有効になる。





抽出処理手順

抽出処理はどの段階からでも行うことが出来る。以下は、先ほどの隠蔽処理につづいて、抽出処理を行った場合の図となる。

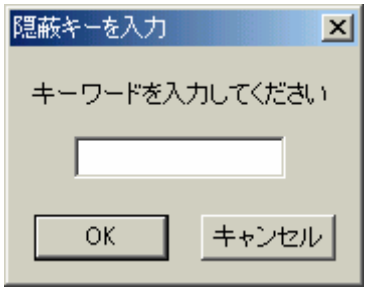
なお、抽出する画像は先ほどパスワードを隠蔽したものとする。

１．抽出処理を行う前（隠蔽処理をし、保存後の画面）の画面は以下のようになっている。

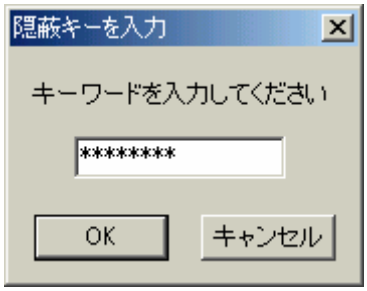
２．抽出ボタンを押すと、「ファイルを開く」ダイアログが表示される。

ここで、先ほどパスワードを隠蔽した画像“ 隠蔽.bmp ”を選択し、開くボタンを押す。

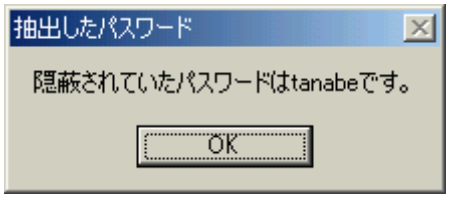
３．開くボタンを押すと、以下のようなダイアログが表示される。



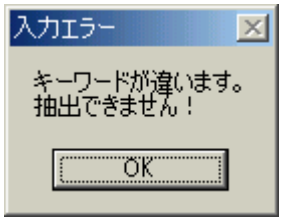
４．ここで、先ほど隠蔽を行ったときに入力した隠蔽キーを入力し、OK を押す。



５．OK ボタンを押した後、抽出が成功すれば以下のようなメッセージボックスが表示される。ダイアログには、抽出できた秘密データ（パスワード）も表示される。
これで抽出処理は終了である。



失敗ならば以下のようなメッセージボックスが表示される。



第４章　ネットワークプログラミング

4.1 ネットワークプログラミングとは

ネットワークプログラミングとは、どこか別の場所で動いているコンピュータとデータを通信するプログラムを作成することである。

ネットワーク上のコンピュータと以下のような手順でデータの通信を行う。

１．ソケットを使用して、ネットワーク上のコンピュータと接続する。
２．ソケット経由でデータの通信を行う。
３．通信が終了したらソケットを閉じて、一連のセッションを終了する。

(引用文献[7]より)

サーバ・クライアント型のシステムを作成するには、サーバプログラムとクライアントプログラムの２つのプログラムを実行する必要がある。

ネットワークプログラミングでは、この２つのプログラムが強調して動かなければならない。そのため、両者の間でサービスの要求・提供方法に関しての約束（プロトコル）をあらかじめ決めておく必要がある。

4.1.1　サーバとクライアント

サーバとはサービス（機能）を提供するプログラムのことであり、クライアントとはサービス（機能）を受けるプログラムのことである[8]。

サーバはクライアントの要求に応じてデータの提供や加工などのサービスを行う。

クライアントは、サーバにサービスを要求しその結果を受け取って結果の表示などを行う。クライアントとサーバでそれぞれ処理を分散して行い、コンピュータへの負荷を分散することが出来る。

C／S 型の仕組みを採用しているシステムの中で、最も有名なものが HTTP である。

HTTP でいうクライアントは Internet Explorer や Netscape Navigater などの Web ブラウザのことである。そしてサーバにあたるのが、WindowsNT の IIS（Internet Information Server）や UNIX でよく使われる Apache などの Web サーバと呼ばれるものである。この２つの関係を簡単に図示すると以下のようになる[9]。

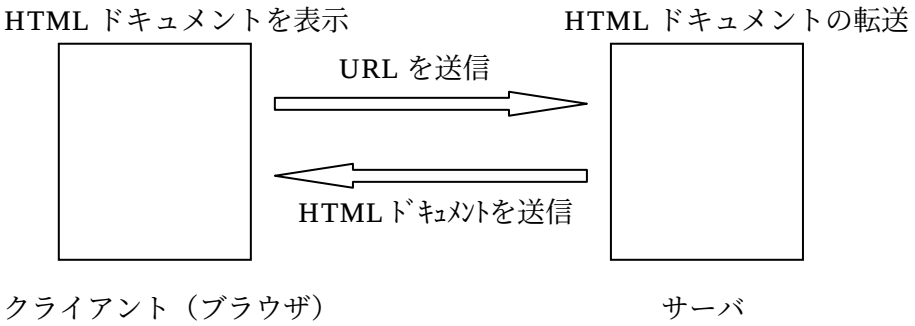


図 4-1　Web サーバとクライアントの関係（引用文献[9]より）

クライアントであるブラウザが「このページの内容を送って欲しい」とサーバに要求すると、サーバは指定された HTML ドキュメントをクライアントに送信する。つまり、クライアントが依頼し、サーバがそれを受けて奉仕するという関係である。

既存のサービスを受けるためのプログラム（例えば「Web ブラウザ」や「電子メール」プログラム）を作るなら、クライアントプログラムさえ作ればよい。しかし、独自のサービスを提供するプログラムを作成するには、サーバプログラムとクライアントプログラムの両方を作る必要がある。

4.1.2　TCP と UDP

ネットワークプログラムを作成するにあたり、TCP による通信か、あるいは UDP かを選択することができる。当然サーバ、クライアントとも同一のプロトコルにする必要がある。

TCP（Transmission Control Protocol）はコネクション型やストリーム型と呼ばれ、サーバとクライアントの 1 対１で接続して通信を行う。通信先との接続制御や、データの再送制御、順序制御はプロトコルで定義されているため、これらの処理をアプリケーションで行う必要がない。信頼性の高い通信方法である。

これに対し、UDP（User Datagram Protocol）はコネクションレス型やデータグラム型と呼ばれ、指定の IP アドレスとポート番号で直接データの送受信を行う。複数のマシンなどに同時にデータを送信することなどができる。TCP とは違い、データの再送制御や順序制御などがプロトコルで定義されておらず、信頼性を向上させるためにはアプリケーション側での対策が必要である。信頼性より通信速度を優先する場合に使用する[8]。

TCP と UDP の特徴をまとめると以下のようになる[9]。

１．TCP が通信に必ず接続が必要な一方で、UDP は必ずしも必要でない。
２．TCP にはエラー訂正・フロー制御などの機能が付いているが、UDP にはない。
３．１度に大量のデータを送受信する場合は、TCP を使用する。
４．UDP は TCP に比べて通信速度が速い。

このようなことから、UDP は「近くのマシンに少量のデータを送りたい」といった場合には向いているが、大量のデータを遠くのマシンに送る必要があるときは、信頼性の高い TCP を利用した方がよい。本研究では、クライアントからサーバに送られるデータには ID の他に画像もあり、送信データ量は多く、また、同じ LAN 内での通信に限られたものではないため、信頼性の高い TCP を使用することとする。

4.1.3 IP アドレスとポート番号

インターネットの中から一台のコンピュータを特定するのために IP アドレスを使い、ポート番号を指定することでサーバプログラムを特定する。（一つのコンピュータでは複数のサーバプログラムを動かすことができるため、どのサーバプログラムとデータをやり取りしたいのかをポート番号で特定する）

サーバは特定のポートを常に見張っていて、クライアントがそのポートに接続するとサーバ側ではリクエストが来たことを知ることが出来る。そこからデータのやりとりが始まる。

4.1.4 ネットワークバイト順序

i386 系 Linux／BSD や Windows 上でネットワークプログラムを製作する場合、コンピュータのバイト順序（ホストバイト順序）とネットワークバイト順序がことなるということを意識しておく必要がある。

Windows 上：リトルエンディアン（上位アドレスに下位バイトが配置される）

(例)　79 → [4F　00]

ネットワーク上：ビッグエンディアン（下位アドレスに下位バイトが配置される）

(例)　79 → [00　4F]

ネットワーク上を伝達される整数値（ポート番号や IP アドレスなど）は必ず関数に渡す前にバイト順序の変換を行っておく必要がある。また、関数から取り出したこれらの値もバイト順序を変換して使う必要がある。これらの処理を行うための標準的な関数として、htons()、htonl()、ntohs()、ntohl()などがある[7]。（関数の詳細は巻末の付録を参照のこと）

4.1.5 アドレスファミリ

BSD ソケット API では TCP／UDP 以外のネットワーク通信機能もサポートできるよう設計されている。BSD ソケットをモデルにして作られた WinSock も、もちろんこの柔軟性を持っている。

ネットワークプログラムを作るためには、データのやり取りを行うコンピュータを識別できる何らかの方法が必要となる。例えば、IP をベースとしたインターネットでは、IP アドレスとポート番号で識別している。

しかし、それぞれのネットワーク規格は独自の方法でコンピュータを識別しているため、

インターネット以外でのネットワークでもこの方法が使えるとは限らない。

IP における IP アドレスのように、そのネットワーク上でのホストアドレス割り当て方法をアドレスファミリ（Address Family）と呼ぶ。アドレスファミリは別のネットワークで互換性を持たせるものである。

このようなアドレスの依存性を和らげるために、sockaddr という 16 バイトの構造体（そのうち 2 バイトはアドレスファミリを示す）でネットワークを表す。残りの 14 バイトには、それぞれのネットワーク基盤が独自の方法で定義したアドレス情報が入れられる。

WinSock プログラムではアドレスがどのアドレスファミリに属するかを指定しなければならない。インターネットの場合は AF_INET となる[7]。

4.2 ソケットとは

ネットワークを介して通信を行うためにはソケットを使用すると述べた。

ソケットとは、一般に TCP/IP において利用するネットワーク用 API のことを指す。ネットワーク間で接続する概念的なものである。インターネットプログラムを作るときには、ほとんどの場合この便利なソケットの機能を利用して接続を確立し、プロトコルに従って通信を行う。ソケットを利用しなくてもある程度のネットワーク機能は利用できるが、柔軟性のある通信プログラムを作ろうと思ったらソケットを利用することになる[7]。

4.2.1 ソケットの使用方法

ソケットにはストリーム型とデータグラム型の 2 種類がある。基本となるプロトコル（TCP か UDP か）の違いが、そのままソケットの違いとなる。ストリームソケットは TCP を使用して通信を行い、データグラムソケットは UDP を使用する。特殊な場合を除き、普通はストリーム型のソケットを使用することが多い。

以下に、ソケットによりサーバ・クライアント間に通信路が開設される手順を示す。通信路の開設には、IP アドレスとポート番号が使われる。

1．サーバがポート番号を指定し、接続要求受付用ポートを作成、クライアントからの接続要求を待つ。（要求受付用ポートではデータの送受信はできない）
2．クライアントが通信用ポートを作る。
　サーバの IP アドレスと、指定されたポート番号を使って接続要求を行う。
3．接続要求が受け付けられると、サーバには新たに通信用ポートが作られる。
　これは特定のクライアントとの通信のために使われる。

こうして一度通信路が開設されると、サーバとクライアントはどちらからでもデータの送受信を開始できる[10]。

4.3 Winsock を利用したネットワークプログラムの作成（TCP）

本節では、簡単なネットワークプログラムの作成手順を示す。どのネットワークプログラムもその処理手順は同じである。まず始めに、Winsock とは何かということを説明し、その後、ネットワークプログラムの作成方法について説明する。

4.3.1 Winsock とは

UNIX 環境においてネットワークプログラムを作成するための方法として、古くからソケットという概念が用いられてきた。

Windows においては、TCP/IP プロトコルの通信インターフェイスを実現するため、1992 年に TCP/IP ネットワークコミュニティに属する 20 社以上のベンダが協力して、API 仕様が策定された。これが Winsock（Windows Sockets）である。Windows でネットワークプログラムを製作する場合、標準的に使用される。

Winsock32 API のプロパティの説明に「BSD Socket API for Windows」と記載されているように、Winsock（Windows のソケット関数）は、カリフォルニア大学のバークレイ校で開発した BSD 版 UNIX（BSD:Berkeley Software Distribution）のバークレイソケット関数を基本に構成されている[7]。

WinSock 関数は大きく分けて４つに分類できる。

- ネットワークバイト順序とローカルホストのバイト順序を変換する関数
- ドメイン名を DNS に問い合わせてホスト情報を取得するデータベース関数
- ソケットを操作してデータ入出力を行うソケット関数
- Windows で拡張された関数（WSA で始まる関数など。UNIX との互換性はない）

4.3.2 Winsock の利用手順

ここで、実際に Winsock を利用したネットワークプログラムの作成手順について説明する。ここでは TCP を用いて通信を行う。なお、各関数の詳しい説明については付録とする。（以下、参考文献[7]より）

―Winsock API の使用方法

Winsock を使用する際は、wsock32.lib をリンクし、winsock.h をインクルードする必要がある。Microsoft Visual C++を使用しているのであれば、winsock.h というヘッダファイルに全ての構造体や関数などが定義されている。Winsock を使用したプログラムを製作する場合、Windows システムディレクトリに存在する wsock32.dll を使用することになる。したがって、プロジェクトの設定でライブラリファイル wsock32.lib を参照する必要がある。メニューバーの[プロジェクト]の中の[設定]を選択、表示されるウインドウの[リンク]タグをクリックし、[オブジェクト／ライブラリモジュール]欄の最後に wsock32.lib を追加する。

ー初期化処理と終了処理

UNIX ではシステムを起動した時点でネットワーク機能が有効になるが、Windows ではプログラムの中でネットワーク機能の初期化処理、終了処理を自分で行わなければならない。このため、WinSock ではいくつかの拡張機能が用意されている。バークレイソケットから Windows ソケットで拡張された関数や定数はすべて WSA という文字で始まることで識別できる。(これらの関数は UNIX との互換性はない)

WinSock プログラミングでは、ネットワーク機能を使用する前に WSAStartup 関数を呼び出して初期化を行う必要がある。そして、終了する前には WSACleanup という関数を呼び出さなければならない。これらを怠るとプログラムで使用しているすべての WinSock 関数でエラーが発生してしまう。

以下にサーバとクライアントの処理の流れを示し、各関数についての説明を行う。

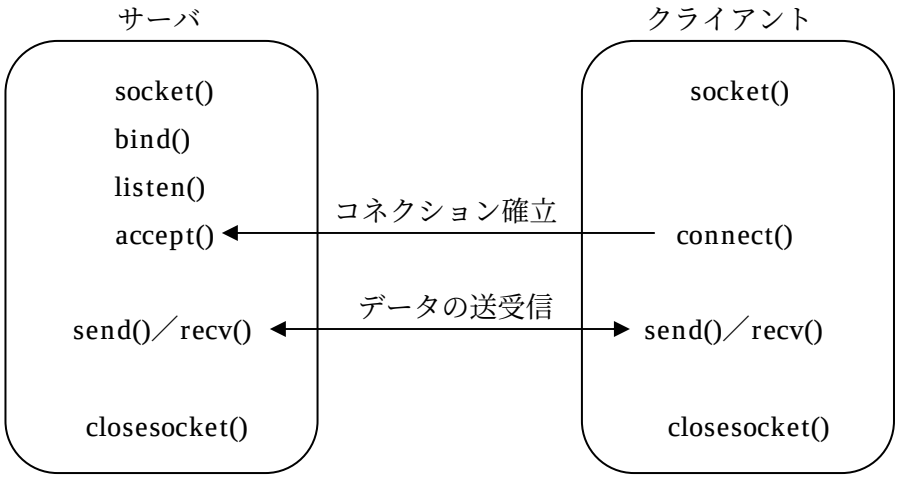


図 4-2　サーバ・クライアントの処理の流れ（TCP）

●ソケット作成（socket）

ネットワーク通信を行うためにはまず、サーバ側及びクライアント側でソケットを作成する必要がある。ソケットの作成には socket()関数を使用する。

関数が正常に終了すると戻り値にソケットディスクリプタ(SOCKET 型)が返ってくる。このソケットディスクリプタはデータ送受信の際に必要となるので保持しておかなければならない。

●ソケットのバインド（bind）

bind 関数はソケットをローカルホストと結合させる作業をする。

後で connect 関数を呼び出すときには bind 関数を呼び出す必要はない。connect 関数は指

定したソケットがバインドされていなければ、自動的にローカルホストのアドレスを取得してバインドを行ってくれる。（bind()関数を呼んでも特に問題はない）

●ソケットの接続待ち（listen）

サーバ側は相手から接続されるのを待っておく必要がある。

ソケットをバインドした後、listen()関数でソケットの接続待ちキュー数（回線数）を指定する。これにより、接続待ち状態になる。

●接続の受け入れ（accept）

接続を確立してデータ通信を行うために、サーバ側では相手から接続要求が来た場合にaccept()関数で接続を受け入れなければならない。

●サーバへの接続（connect）

クライアント側はソケットを生成後、サーバと通信を行うために connect()関数でサーバに接続し、コネクションを確立する必要がある。

●データの送信（send）

コネクションが確立した相手に対し、データの送信をする場合、send()関数を使用する。

●データの受信（recv）

コネクションが確立した相手からデータを受け取る場合、recv()関数を使用する。

●ソケットを閉じる（closesocket）

ソケット通信を終了するときには closesocket()関数を使用する。

使い終わったソケットは必ず閉じる。

以上が TCP を使用した通信の流れである。UDP の場合もほとんど同じだが、コネクションレス型のためサーバ側での listen()、accept()は行わない。またデータの送信にはsendto()を使用し、受信には recvfrom()を使用する。

4.3.3　同期型と非同期型

ソケット関数の多く、具体的には accept、connect、recv、send、closesocket の各関数は基本的にアクションが完了するまで戻ってこない。

例えば、接続先が遠いところにある場合など相手からなかなか応答が来ない時、関数を呼び出したルーチンはずっとブロックされ、相手側から応答があるか何らかのエラーが検出されない限り、ウインドウの移動や再描画などのウインドウメッセージを全く処理でき

なくなる。（アプリケーションの終了すら出来ない）これでは使いやすいプログラムとはいえない。

このような状態を回避するために、Windows Sockets ではノンブロッキング処理（非同期処理）が出来るような関数が用意されている。

非同期処理を行うためには、send()、recv()、connect()、accept()など、通常は相手からのレスポンスがあるまでブロッキングしてしまうようなソケット関連の関数を呼び出す前に、WSAAsyncSelect()を呼び出す。

こうすると非同期処理下でのソケット関数の呼び出しは即座に戻り、実際にアクションが完了した時には任意のウインドウに通知メッセージが発行される。

以下に、関数の使用法について説明する。

＜WSAAsyncSelect 関数＞

int　WSAAsyncSelect（SOKET sock,　HWND hwnd,　UINT wMsg,　long lEvent）;

●第三引数について

ここでは、通知するメッセージを指定する。

WM_USER 以降であれば、自由に設定可能なので、例えば

```
#define　WM_SOCKET　WM_USER + 1
```

のように定義しておき、WM_SOCKET を通知メッセージとして指定する。

第四引数で指定したソケットイベントが発生すれば､WM_SOCKET メッセージが発生する。

●第四引数について

ここで指定できるイベントのタイプには以下のようなものがある。

FD_ACCEPT	相手から接続要求が来たことを通知
FD_CONNECT	ソケットの接続が完了したことを通知
FD_WRITE	送信準備が出来たことを通知 （送信不可能な状態から可能な状態になった）
FD_READ	受信準備が出来たことを通知（相手が何か送信してきた）
FD_CLOSE	ソケットへの接続が終了したことを通知 （相手が接続を切断した）

FD_READ を例にとって説明すると、データを受信する場合に、非同期関数を使用しなければ、相手がデータを送信してきてそのデータの受信が完了するまで処理がブロックされる。相手が送信状態になるまでに何か重い処理を行っていた場合、なかなかデータが送

信されてこないため、recv はずっと待っていることになり、その間、他の処理が何も出来なくなる。

これを防ぐために、非同期関数を使用し、受信準備が出来たという通知（FD_READ）が来てから受信を行うようにすれば、相手が送信してくるまでの間に別の処理をすることが出来る。

send 場合も要領は同じだが、ファイルサイズが大きい場合には、一度の送信処理でデータを全て送信できないため、ファイルを転送中に FD_WRITE が発生した場合は残りのデータを送信しなければならない。

ウインドウにソケットイベントが通知されるときには、第三引数で指定したメッセージが発行される。

ウインドウプロシージャの wParam には、イベントが生じたソケットが入っている。lParam の上位ワードには、エラーコードが入っている。WSAGETSELECTERROR マクロで取得でき、０以外の場合はエラーである。また下位ワードには発生したイベントのタイプが入っている。第三引数で指定したメッセージが発行されたときは、WSAGETSELECTEVENT マクロでイベントのタイプを取得することが出来る。

複数のソケットを扱う場合は、それぞれのソケット毎に WSAAsyncSelect 関数を実行する必要がある。

同一ソケットに対して、複数回 WSAAsyncSelect 関数を実行した場合、最後に実行したものだけが有効になり、それ以前のものは無効となる。

ソケットイベントの通知を取り消すには、第三引数と第四引数に０を設定する。

第５章　Steganography を利用したログインシステム

本研究では、ID とパスワード隠蔽画像でサーバにログインするシステムを開発した。画像にパスワードを隠蔽する方法についてはすでに第３章で説明した。本章では開発したログインシステムの構成と実装方法について述べる。

5.1 システム構成

サーバとクライアントの構成は以下のようになっている。

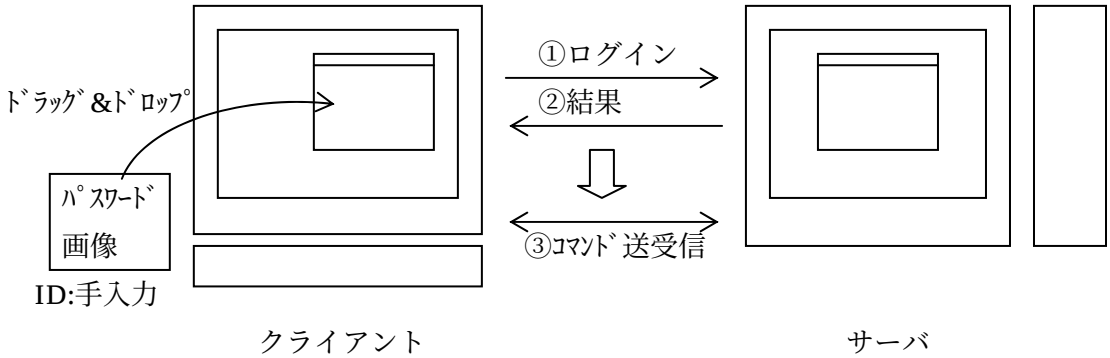


図 5-1　サーバとクライアントの構成

① クライアントで ID を入力し、パスワードの隠蔽された画像ファイルをフォームにドラッグ＆ドロップして読み込み、サーバに接続する。（ファイル名を入力することでも読み込める）
② サーバで ID とパスワードの検索・照合を行い、結果をクライアントに返す。
③ ログインが成功すれば、コマンドの送受信を開始する。失敗ならば接続を切る。

大まかな流れは上記の通りである。処理としては、大きく分けて、クライアントからサーバへの接続、サーバでのログイン内容の検証、サーバ・クライアント間でのコマンドの送受信の３つで構成される。各処理の詳しい内容は次節で説明する。

5.2 サーバ・クライアントプログラムの処理の流れ

サーバとクライアントの処理の流れを具体的に説明する。
以下にサーバプログラムの流れ図を示す。

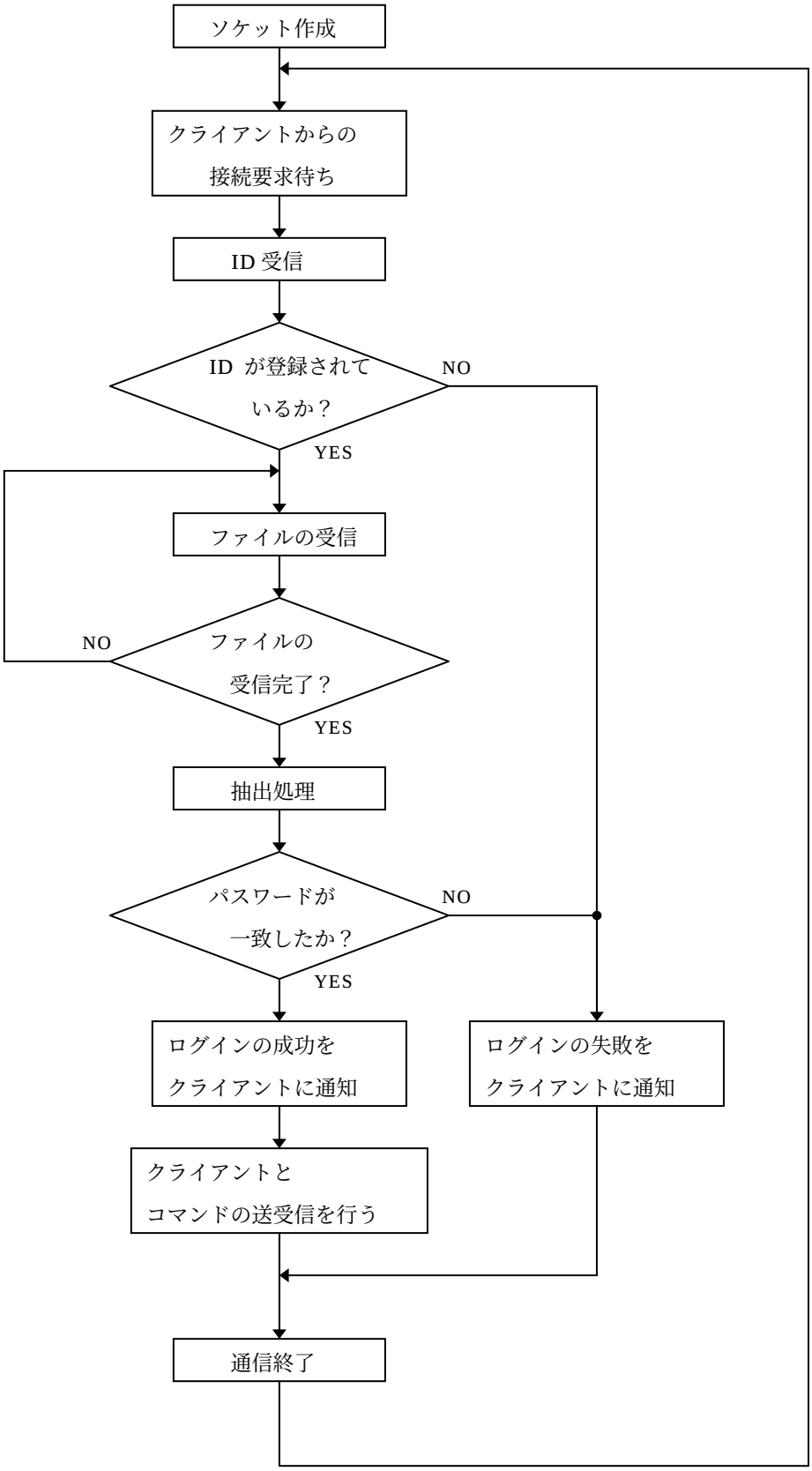


図 5-2　サーバプログラムの処理の流れ

サーバでの処理は

① ソケットを作成し、クライアントからの接続要求を待つ。

② ID を受信し、ID ファイルを開いて検索・照合を行う。

③ ID が登録されていれば、ファイル（パスワード隠蔽画像）の受信を行う。

④ ファイルから隠蔽キーが抽出できれば、続いてパスワードを抽出し照合する。

⑤ ログイン結果をクライアントに伝える。

⑥ ログイン成功ならば、クライアントとコマンドの送受信を行う。

となる。各処理について説明する。（①については省略する）

②では、まずクライアントから送られてきた ID を受信する。ここで、ID ファイルとは以下のような形式になっている。

図 5-3　ID ファイルの形式

ID ファイルには登録されている ID とパスワードが１行に入っている。ID とパスワードは半角スペースで区切られている。

サーバでは、ファイルから１行ずつ読み込み、ID だけを切り出して、受信した ID と比較、合っていれば登録されているパスワードを切り出して配列に格納している。

このパスワードと、画像ファイルから抽出されたパスワードとを照合する。

ID が登録されていればファイルの受信処理に移る。受信処理で注意しなければならないのは、TCP ではストリーム型のソケットを使用するため送信の区切りは受信側では分からないということである。例えば、クライアントから「ABCD」と送ったとしても、サーバでは「AB」と「CD」のように２回に分けて受信されるかもしれない。そこで、まずファイルサイズをクライアントから送ってもらい、その後ファイルの受信を行う。先に送られてきたファイルサイズと比較し、受信バイト数が等しくなったら受信処理を終了する。

次に抽出処理に移る。（抽出方法については第３章で説明済み）

受信ファイル（パスワード隠蔽画像）に隠蔽キーが埋め込まれているかをチェックする。隠蔽キーとは画像にパスワードが埋め込まれている事を示すものである。隠蔽キーは“ testpass ”に統一されている。これが抽出できれば、続いてパスワードを抽出し、先ほど、ID を検索した時に保存しておいた登録パスワードと比較、結果をクライアントに送信する。

サーバからクライアントへのログイン結果の通知は以下のようになっている。

ログイン成功　→　“ login success ”

ログイン失敗　→　“ login failure ”

ログインが成功すれば、接続を継続してクライアントとコマンドの送受信を行い、失敗の場合は接続を終了し、接続要求待ち状態に移る。

クライアントから送られてくるコマンドに対して、サーバから返されるメッセージは、

Hello!	：こんにちは
Good-Morning	：おはよう
Good-Bye	：さようなら
その他	：Not Found

となっている。これらの対応表も ID ファイルと同じ形式でファイルに格納されており、ID の検索と同様の方法で調べることが出来る。

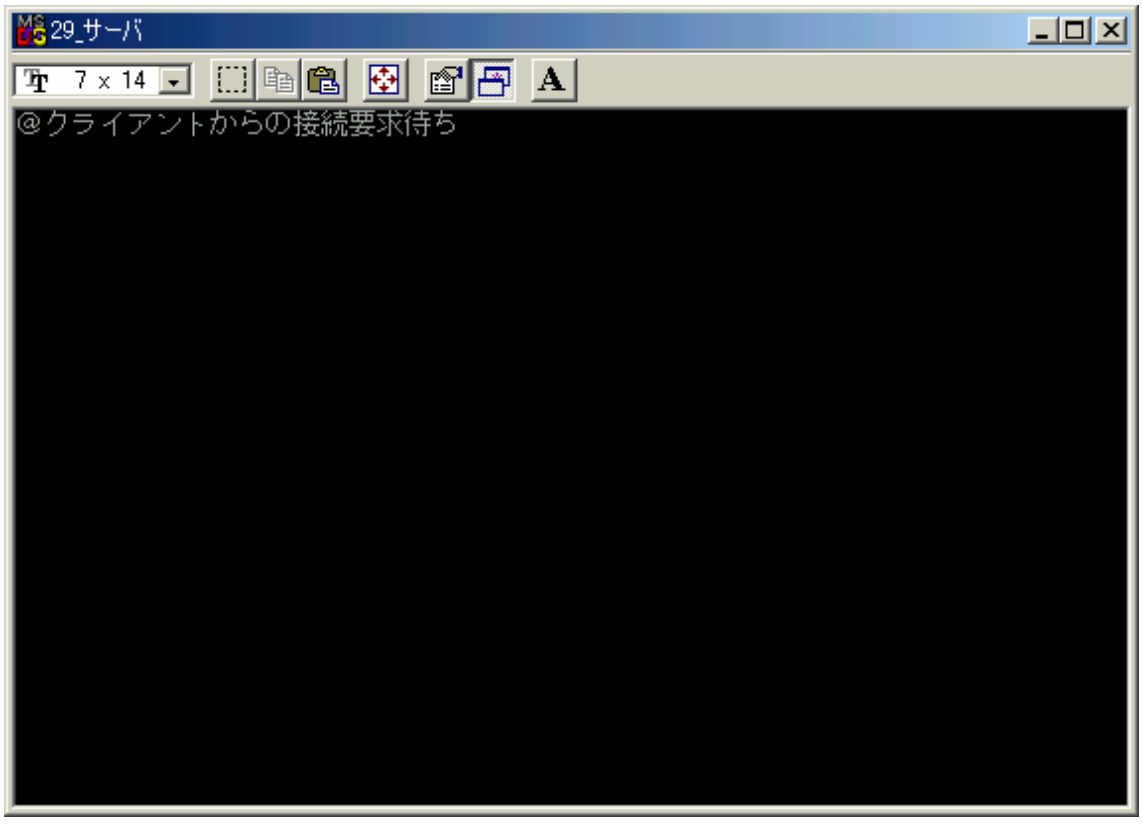
続いて、クライアントの処理についてであるが、クライアントは Windows プログラムであるため、起こったイベント（ボタンを押すなど）によって処理が分かれる。そこで、クライアント側の処理については、次節のログイン処理の流れの中で説明することとする。

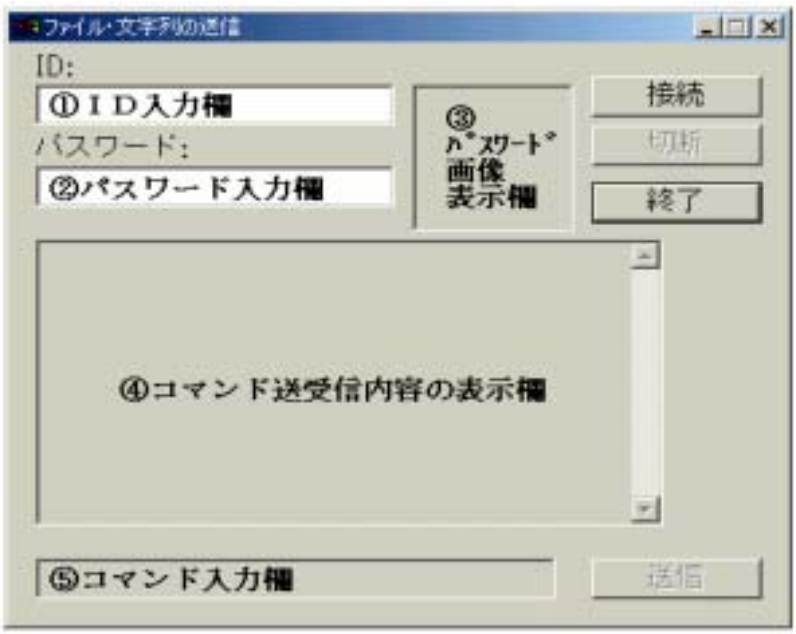
5.3 実行結果

実際に、クライアントからサーバへログインを行ってみる。

以下に実行画面を示す。

１．サーバとクライアントの起動直後の状態は以下のようになっている。





クライアント画面の各構成要素について説明する。

①ＩＤ入力欄：

ユーザＩＤをキーボードから入力する。

②パスワード入力欄：

クライアント画面内に画像をドラッグ＆ドロップすると、そのファイル名が表示される。（キーボードからの入力もできる）

③パスワード画像表示欄：

パスワード入力欄に入力したファイルのプレビューが表示される。

表示欄よりも画像が大きい場合は、縮小表示される。

④コマンド送受信内容の表示欄：

サーバ・クライアント間でのコマンドの送受信内容が表示される。

⑤コマンド入力欄：

コマンドを入力する。

接続ボタン：サーバに接続する。

切断ボタン：サーバから切断する。

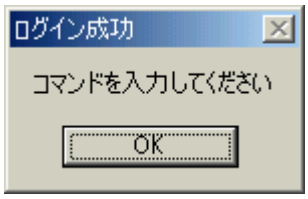
終了ボタン：プログラムの実行を終了する。（サーバとの切断処理も行う）

送信ボタン：サーバにコマンドを送信する。

2．クライアントで ID とパスワードを隠蔽した画像“ 隠蔽.bmp ”を入力したあとの画面は以下のようになる。ここで、画像ファイルは、フォームにドラッグ＆ドロップすることで読み込むことが出来る。ドラッグ＆ドロップすると、パスワード欄にファイル名が入力され、画像のプレビューが表示される。

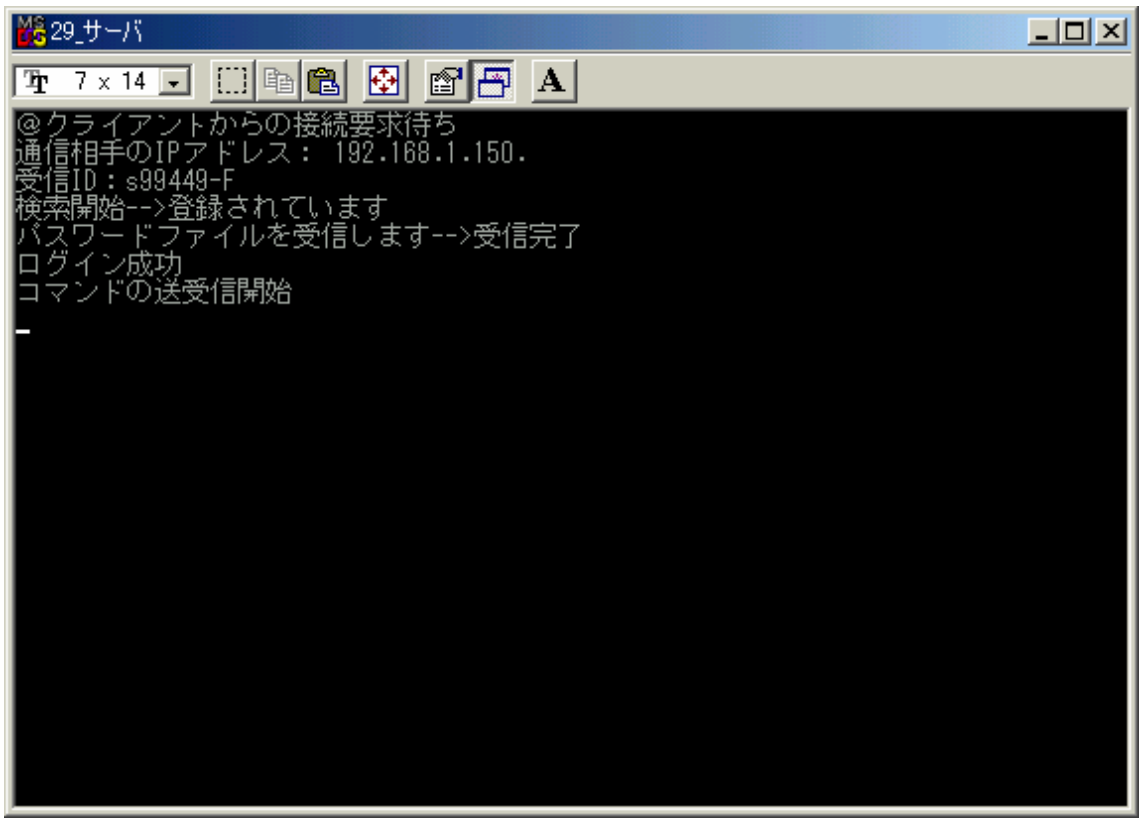
３．クライアントで接続ボタンを押して、サーバに接続する。

接続が成功すると、クライアントでは以下のようなメッセージボックスが出る。

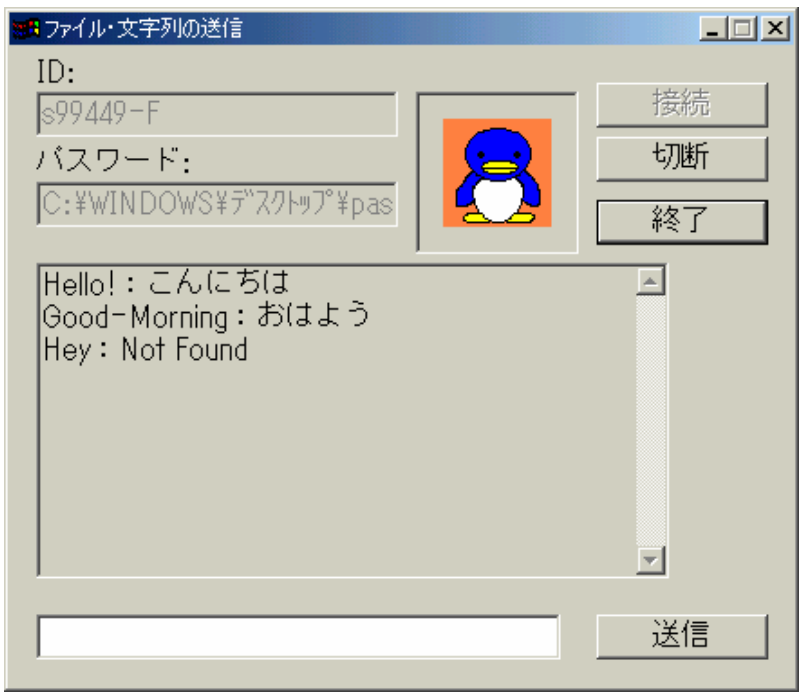


OK を押すと、以下のようにコマンド入力欄がキーボードからの入力を受け付けるようになる。

サーバの状態は以下のようになる。

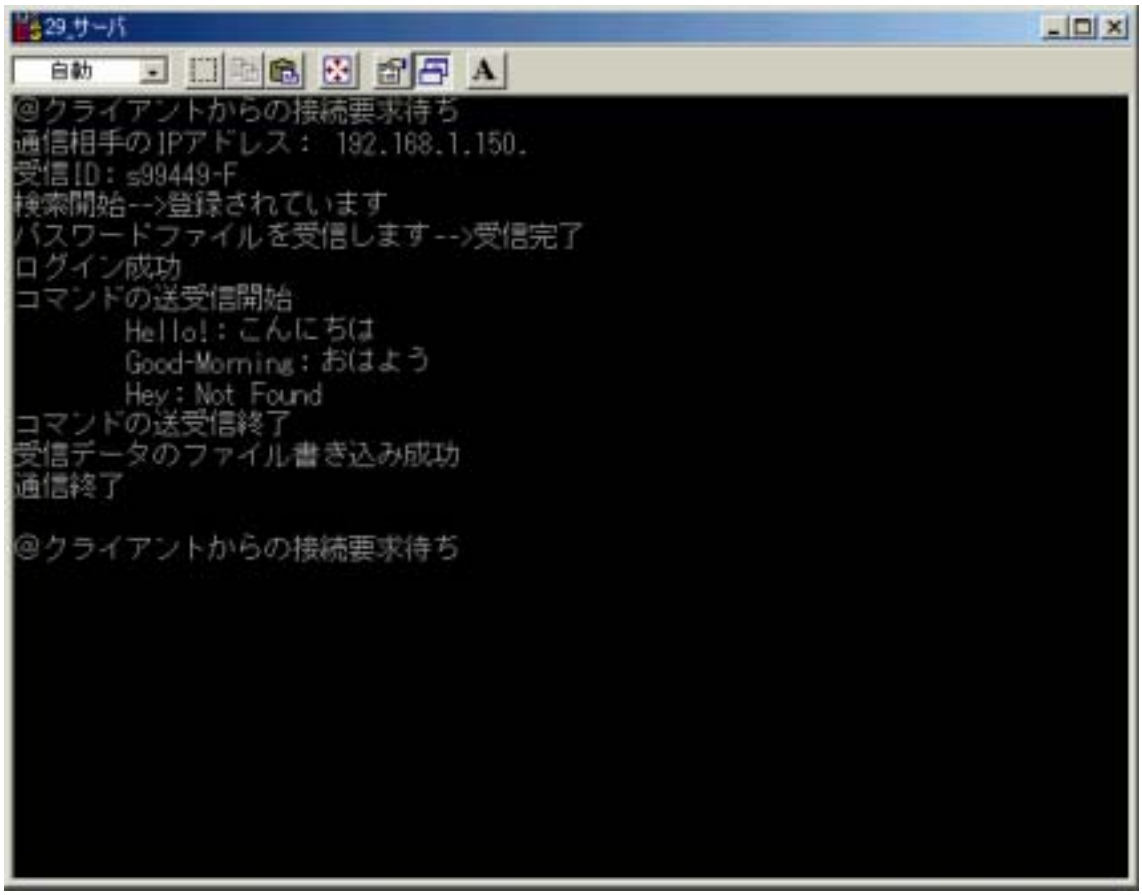


４．コマンド入力欄に入力を行い、送信ボタンを押すと、入力したコマンドに対する応答がサーバから返される。コマンドの送受信内容は中央のエディットボックスに表示される。



５．接続を終了するには、切断ボタンを押す。
　すると、以下のようなメッセージボックスが表示されるので、ここで OK とするとサーバから切断される。（終了ボタンを押すと、接続終了とクライアントプログラムの実行の終了を一度に行う）
　クライアントが接続を終了すると、サーバは次のクライアント待ちに移る。

クライアントとの接続終了後の画面（クライアントからの接続要求待ち状態に移る）



第６章　終論

本システムのログイン方法では、クライアントからのログイン処理の際のパスワード入力の手間が省け、また、パスワードを忘れる心配がなくなった。さらに、他人が画像を見ても、それがパスワードであることに気づきにくいため、パスワードの管理が容易になった。しかし、今のところ ID は手入力となっており、この部分も同様に隠蔽することでさらなる改良ができると思われる。

また、セキュリティ面でのことを考えた場合、現在使用している隠蔽方法では弱い。本研究では、隠蔽アルゴリズムとして最も単純な画素置換型を用いており、隠蔽する部位も各画素の最下位ビットに限定されている。ステガノグラフィは本来、隠蔽されていること自体が秘密にできる技術であるので、第三者によっての攻撃（隠蔽内容の抽出）を受ける可能性は少ないが、万一攻撃を受けても、容易に抽出を行えないよう対策を立てるべきである。また、画素置換型では、画像の非可逆圧縮などを行った場合に、隠蔽した内容が消えてしまう恐れがある。

この解決として、周波数領域へ隠蔽する方法がある。周波数領域へ変換する方法としては、フーリエ変換や離散コサイン変換（DCT）などがある。特に、DCT は JPEG などの圧縮技術の中心をなす変換であり、これを使用することによって、画像の圧縮が行われた場合にも隠蔽内容が消える心配が少なくなる。

さらに大きな問題として、現システムの実行環境がある。本研究では、クライアントはアプリケーションプログラムであるため、各クライアントに実行プログラムをインストールしておく必要がある。この方法では、サーバにログインできるコンピュータはプログラムがインストールされているものに限られるため使い勝手がよいとは言いづらい。よって、Internet Explorer や Netscape Navigator などのブラウザからログインできるように改良する方がよい。

現在、この問題を解決するために、Java アプレットを使用して実行を試みている。まず、クライアントから URL とポート番号を指定してサーバに接続を行う。サーバはログイン処理を行うための Java アプレットを、接続要求のあったクライアントにダウンロードして実行する。以下、現システムと同様に処理が行われる。

しかしここで問題となるのは、Java アプレットのセキュリティ制限である。ブラウザから本システムと同様の処理を行うためには、クライアントマシン内の画像ファイルの読み込みが行えなければならない。しかし、これはアプレットのセキュリティによって、通常は実行することが出来ない。このため、各クライアントマシンに Java Plug-in をダウンロードしておき、その中にある java.policy ファイルのセキュリティ設定を書き換える必要があり、現在、どのような方式をとって実行するのがよいか思考中である。

謝辞

本研究にあたり、最後まで熱心な御指導をいただきました田中章司郎教授には、心より御礼申し上げます。また、田中研究室の貫目洋一さん、辰己圭介さん、高木明さん、長田昌訓君、喜代吉容大君、鷲見明君、垰田千帆さん、堀隆志君には、本研究に関して数々の御協力と御助言をいただきました。厚く御礼申し上げます。本研究の特許出願に当たり、田辺特許商標事務所の田辺義博弁理士には多大なるご尽力を賜りました。心より感謝申し上げます。

なお、本論文、本研究で作成したプログラム及びデータ、並びに関連する発表資料等のすべての知的財産権を、本研究の指導教官である田中章司郎教授に譲渡致します。

引用文献・資料

[1] 九州工業大学　知識工学研究室
http://www.know.comp.kyutech.ac.jp/BPCS/

[2] 松井甲子雄　著
「電子透かしの基礎 -マルチメディアのニュープロテクト技術-」
森北出版,1998
ISBN4-627-82551-X

[3] 小野 束　著
「電子透かしとコンテンツ保護」
オーム社,2001
ISBN4-274-06401-8

[4] 情報・通信事典 e-Words
http://e-words.jp/

[5] Windows における画像処理について
http://lacom2.ice.ous.ac.jp/t98/domi/study/dib/dib.html

[6] Microsoft MSDN ライブラリ
http://www.microsoft.com/japan/msdn/library/

[7] Hey! Java Programming!
http://www.mars.dti.ne.jp/~torao/program/

[8] Internet Programming
http://www.nakka.com/lib/inet/

[9] VC++でやる SDK
http://kerochan.no-ip.com/vcsdk/index.php

[10] ネットワークプログラミング
http://rananim.ie.u-ryukyu.ac.jp/~kono/lecture/2001/os/ex/tcp/network.html

[11] WindowsAPI Programming　第３章 ～ソケット通信～
http://yonex1.cis.ibaraki.ac.jp/~yonekura/2002kadai/lecture03.html

付録

【ソケット関数（WinSock32 API)】
ーバークレイソケット互換

socket 関数

SOCKET　socket(int af, int type, int protocol)

ソケット（Socket Descriptor）を作成する。

socket 関数で作成したソケットは必ず closesocket 関数（UNIX の場合は close 関数）で破棄する。

第一引数：アドレスファミリ（通常は AF_INET)、

第二引数：ソケットの種類（SOCK_STREAM：TCP で使用する、SOCK_DGRAM：UDP で使用する）

第三引数：プロトコル、を指定

戻り値：成功時は新しいソケット、失敗時は INVALID_SOCKET

bind 関数

int　bind(SOCKET sock, (const struct sockaddr) *addr, int len)

ローカルの IP アドレスとポート番号をソケットに結合する。

実際に構築されたソケットを使用するには、このバインドという作業を行わなければならない。

第一引数：socket 関数で作成されたソケット

第二引数：接続するソケットのアドレスへのポインタ

第三引数：ソケットのアドレスのサイズ

戻り値：成功時は０、失敗時は SOCKET_ERROR

connect 関数

int　connect(SOCKET sock, (const struct sockaddr) *addr, int len)

指定のソケットを使用してリモートホストに接続要求を出し、通信路の確立をする。（connect 関数は指定されたソケットがバインドされていなければ自動的にローカルホストのアドレスを取得してバインドを行う）

第一引数：socket 関数で作成されたソケット

第二引数：サーバのアドレスの構造体へのポインタ

第三引数：その構造体のサイズ

戻り値：成功時は０、失敗時は SOCKET_ERROR

listen 関数

int　listen(SOCKET sock, int queue)

指定したポートで接続を待つ。

第一引数：接続要求を受け付けるソケット

第二引数：接続要求キューが成長できる最大の長さ

戻り値：成功時は０、失敗時は SOCKET_ERROR

accept 関数

SOCKET　accept(SOCKET sock, (struct sockaddr) *addr, int *len)

サーバソケットにリモートホストからの接続要求を許可する。

（接続要求の待ち合わせと別ソケットの作成）

第一引数：socket 関数によって作成され、bind 関数でアドレスに結び付けられた
　　ソケット。ここで指定されたソケットへの接続要求を受け入れる。

第二引数：接続先のホストのアドレスが入れられる。

第三引数：最初に第二引数の指す領域の大きさを入れて呼び出し、
　　実際のアドレスの長さ（byte）が返される。

戻り値：成功時は新しいソケット、失敗時は SOCKET_ERROR

send 関数

int　send(SOCKET sock, char *buf, int len, int flag)

ソケットを使用してデータを送信する。（ファイルでの出力にあたる）

第一引数：socket 関数で作成されたソケット

第二引数：送信する文字列のバッファ

第三引数：文字列の長さ、第四引数：呼び出し方法の指定

戻り値：成功時は送信されたバイト数、失敗時は SOCKET_ERROR

recv 関数

int　recv(SOCKET sock, char *buf, int len, int flag)

ソケットを使用してデータを受信する。（ファイルでの入力にあたる）

第一引数：socket 関数で作成されたソケット

第二引数：受信するバッファ、第三引数：受信バッファのサイズ

第四引数：呼び出し方法の指定

戻り値：成功時は受信バイト数（接続が終了しているときは０）、
　　失敗時は SOCKET_ERROR

shutdown 関数

int　shutdown(SOCKET sock, int how)

指定のソケットの送信や受信を無効にする。

第一引数：socket 関数で作成されたソケット

第二引数：０は recv、１は send、２は send と recv をそれぞれ無効にする

戻り値：成功時は０、失敗時は SOCKET_ERROR

closesocket 関数

int　closesocket(SOCKET sock)

socket 関数で作成されたソケットを破棄する。

引数：socket 関数で作成されたソケット

戻り値：成功時は０、失敗時は SOCKET_ERROR

―バイトオーダー変換

htons 関数

unsigned short　htons(unsigned short hostnum)

２バイト整数を Windows 形式からネットワークのバイト並びに変換。

htonl 関数

unsigned long　htons(unsigned long hostnum)

４バイト整数を Windows 形式からネットワークのバイト並びに変換。

―アドレス変換

inet_addr 関数

unsigned long　inet_addr(const char *ipaddr)

ドットで区切った１０進数の IP アドレスを 32bit の整数で表した IP アドレスに変換する。

引数：ドットで区切った１０進数の IP アドレスの文字列へのポインタ

戻り値：成功時は３２bit の整数で表した IP アドレス、失敗時は INADDR_NONE

―その他

gethostbyname 関数

struct hostent　*gethostbyname(const char *hostname)

ホスト名からホスト情報を取得する。

引数：ホスト名の文字列へのポインタ

戻り値：成功時はホストの情報のポインタ、失敗時は NULL

ー拡張されたもの

WSAStartup 関数

int　WSAStartup(WORD wVersionRequired, LPWSADATA lpWSAData)

WinSock の初期化を行う。

第一引数：使用する WinSock のバージョン

第二引数：WinSock の情報を収める構造体へのポインタ

戻り値：成功時は０、失敗時は０以外

WSACleanup 関数

int　WSACleanup(void)

WinSock のリソースの解放を行う。

（すべての未処理データを送信し、ソケットを閉じる）

引数：なし

戻り値：成功時は０、失敗時は SOCKET_ERROR

WSAGetLastError 関数

int　WSAGetLastError(void)

WinSock のエラーコードを取得する。

引数：なし

戻り値：エラーコード

WSAAsyncSelect 関数

int　WSAAsyncSelect（SOKET sock, HWND hwnd, UINT wMsg, long lEvent）;

非同期処理を行う関数。

第一引数：非同期にするソケット

第二引数：ソケットイベントの通知をするウインドウのハンドル

第三引数：ウインドウに通知するメッセージ

第四引数：通知させるイベントのタイプ

戻り値：成功なら０、失敗なら SOCKET_ERROR