情報通信実験（課題⑤　Base64 による暗号化と復号を用いた通信）

実験レポート

出席番号：　16　名前：　　小形孔明　　実験日：2025 年 10　月 23　日

１．課題目的

5-1: Base64（エンコード／デコード）の仕組みを理解し、Java で自作クラスと静的メソッドを設計・実装する力を身につける。あわせて、文字列→配列や数値→文字列などの基本 API を横断的に使いこなし、入出力の照合を通じて実装の正しさを検証する。

5-2: Base64 処理を実際のクライアント／サーバ通信に組み込み、送信時エンコード・受信時デコードの位置づけを理解する。さらに、パケットキャプチャで線上を流れるデータが Base64 化されていることを観測する。

２．課題内容

5-1: 事前指定のクラス名 MyBase64 に、static String encode(String str1)（平文→Base64 文字列）と static String decode(String str1)（Base64 文字列→平文）を実装する。C 言語の Base64 サンプルを参考にしつつ、Java の toCharArray や String.valueOf、Integer.toBinaryString、substring 等の文字列変換 API を活用する。実験では少なくとも入力「a」「hello」「あ」「制御」「Σ」の変換結果を確認・記録し、文字種（ASCII／多バイト）の違いによる挙動も把握する。

5-2: 送信側（クライアント）は自作 MyBase64 で送信直前にエンコードし、受信側（サーバ）は java.util.Base64 で受信直後にデコードする。送信テキストは課題④-2 と同様に苗字（全て小文字英字）を用いる。サーバ側で Wireshark により暗号化（エンコード）済みデータを受信している TCP パケットを観測し、指定用紙に TCP ヘッダと TCP データのみ記録する。提出は、記録用 Excel を PDF 化し、他の観測結果 PDF と連結して 1 つの PDF にまとめる。加点対象。

３．作成プログラムの重要箇所の解説

5-1:MyBase64.java

static final String TABLE =

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";// 6bit→1 文字の写像“辞書”。以降のビット演算の拠り所。

sb.append(TABLE.charAt((b0 >>> 2) & 0x3F));

sb.append(TABLE.charAt(((b0 << 4) | (b1 >>> 4)) & 0x3F));

sb.append(hasB1 ? TABLE.charAt(((b1 << 2) | (b2 >>> 6)) & 0x3F) : '=');

sb.append(hasB2 ? TABLE.charAt(b2 & 0x3F) : '='); //エンコードの核心：ビットシフト＆マスクで 6bit 抽出＋=処理。

5-2:SimpleServer.java（受信直後デコード→応答を再エンコード）

String decoded = new String(java.util.Base64.getDecoder().decode(msg), StandardCharsets.UTF_8);

//受信直後に Base64→平文化（UTF-8）してアプリ処理できる形に。

String res = "ECHO: " + decoded;

out.println(java.util.Base64.getEncoder().encodeToString(res.getBytes(StandardCharsets.UTF_8)));// 応答は平文を組み立ててから再度 Base64 化して送信。

SimpleClient2.java（送信前 encode／受信後 decode を“差し込み”）

```
public void sendMessage(String s){
   out.println(MyBase64.encode(s)); //  自作エンコーダで送信直前に変換
}
String line = in.readLine();
String shown = MyBase64.decode(line); //  受信直後に平文化して表示
con.displayMessage(shown);
```

４．結果

5-1

エンコード前	エンコード後
a	YQ==
hello	aGVsbG8=
あ	44GC
制御	5Yi25b6h
Σ	zqM=

5-2:後述

５．感想・考察

今回の課題を通じて、Base64 は「暗号」ではなく文字集合の制約を越えてデータを可搬化するための符号化であることを、実装と計測の両面から改めて体感した。MyBase64 の手実装では、3 バイトを 4 文字へ写像する際のビットシフトとマスク、末端の=によるパディング、そして UTF-8 で文字列を一旦バイト列に落とす必然性が腑に落ちた。特に「あ」「制御」「Σ」のような非 ASCII 文字は UTF-8 で可変長になるため、Base64 のブロック長と整合させるにはまずバイトに還元する設計が不可欠である。一方で、readLine() が改行終端を消費する性質や、ストリーム全体の文字コードとペイロードの関係にも注意を向けられた。

通信への組み込みでは、送信直前エンコード／受信直後デコードという挿入位置を明確に定義することで、アプリ層のロジックが従来どおり平文を扱える設計の利点を実感した。サーバ側での終了判定やエコー処理は、デコード後に行う必要があり、順序を誤ると「bye」のような制御語が認識されない。Wireshark 観測では TCP データ部に Base64 文字列のみが載っていることが視覚的に確認でき、行単位プロトコルでは CRLF と 1 行=1 メッセージの整合が取りやすいことも理解できた。Base64 の採用によりおよそ 33% のサイズ増が生じる点、誤って「秘匿化」と混同すると過信を招く点も重要で、秘匿性が必要な場面では TLS や AES など真正の暗号を別途組み合わせるべきである。総じて、符号化アルゴリズムの構造を手で追い、ネットワーク計測で現象として確かめ、アプリ設計で位置づけを定義するという三層の学びが連結したというわけだ。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	3	e	7	f	4	c	3	0	a	7	e	f	7	b	e	e	4	6	7	9	f	3	9	5	0	1	8	0	0	f	f
c	5	d	b	0	0	0	0																								
5	2	5	5	4	e	4	9	5	4	7	a	6	f	6	7	6	2	3	2	6	4	6	8	6	4	4	7	4	5	3	d
0	d	0	a																												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
サーバ側ポート999				クライアント側ポート62656				シーケンス番号176093118								確認応答番号3831996217								DO=5(=20B), Flags=0x18				ウィンドウサイズ255			
TCPチェックサム				緊急ポインタ0																											
R		U		N		I		T		z		o		g		b		2		d		h		d		G		E		=	
CR		LF																													