目 次

第 1 章　序論 **1**

第 2 章　ワンタイムパスワード認証と暗号技術 **2**

2.1　ワンタイムパスワード認証方式 2

2.1.1　S/Key 認証 2

2.2　ハッシュ関数 3

2.2.1　ハッシュ関数の安全性 3

2.2.2　MD(Message Digest Algorythm) 4

2.2.3　SHA(Secure Hash Algorythm) 4

第 3 章　SAS-L 認証方式と実装 **5**

3.1　SAS-L(4) 5

3.1.1　定義と表記法 5

3.1.2　初回登録フェーズ 6

3.1.3　認証フェーズ 7

3.2　SAS-L(3) 9

3.2.1　定義と表記法 10

3.2.2　初回登録フェーズ 10

3.2.3　認証フェーズ 11

3.2.4　SAS-L 認証方式の演算負荷比較 13

3.3　安全性 14

第 4 章　実験方法および評価実験 **16**

4.1　実装方法 16

4.2　計算機環境 16

4.3　実験結果 17

第 5 章　結論 **19**

謝辞 **20**

参考文献 **21**

付 録 A その他 **22**

A.1 実行方法 22

付 録 B プログラムリスト **23**

B.1 SAS-L(4) 23

B.1.1 サーバ側の動作 23

B.1.2 ユーザ側の動作 27

B.2 SAS-L(3) 30

B.2.1 サーバ側の動作 30

B.2.2 ユーザ側の動作 33

第1章　序論

近年、インターネット技術の発展により、従来の PC やスマホ端末に加え、自動車、家電製品、建物など、世界中の様々なものがインターネットに接続されている。インターネットに接続された IoT 機器の増加にともない、サイバー攻撃への対応が大きな問題になっている。実際、情報通信研究機構（NICT）が運用するサイバー攻撃観測網（NICTER）が 2020 年（令和 2 年）に観測したサイバー攻撃の統計結果 [1] によると、約 4 割が IoT 機器を狙ったものであるという結果が示されている。IoT 機器は PC などに比べると処理能力が低いため、安全性を確保しつつ処理コストをおさえたセキュリティ対策が必要である。

本研究では、IoT のための認証手法として、SAS-L(Simple And Secure password authentication protocol,Light processing version) [2] について検討する。SAS-L とは、高知工科大学の清水明宏教授が開発したワンタイムパスワード認証方式である。特徴は、従来のワンタイムパスワード認証方式 SAS-2 に比較して、更に処理コストが低いであることである。特に、認証側あるいは被認証側において、処理負荷の大きい一方向性関数の適用を必要としない。センサや IC タグのような小型かつ低スペックの IoT 機器に搭載可能である。

SAS-L には、SAS-L(1)〜(4) という 4 種類のバージョンが提案されている。このうち被認証側で一方向性関数を必要としないのは SAS-L(3) と SAS-L(4) である。SAS-L(4) については実装実験が行われているが、SAS-L(3) については実装による評価はされていない。そこで、本研究では SAS-L(3) の実装および安全性に関する検討を行い、類似している SAS-L(4) と比較しながら SAS-L(3) の有効性について検証する。

第 1 章では、研究の背景および目的について述べる。第 2 章では、SAS で用いられる暗号技術について述べる。第 3 章では SAS-L(3) および SAS-L(4) のプロトコルについて述べる。第 4 章では、評価実験の方法および結果を示す。第 5 章では、本研究のまとめおよび今後の課題を述べる。

第2章　ワンタイムパスワード認証と暗号技術

2.1　ワンタイムパスワード認証方式

ワンタイムパスワード認証 [3] とは、ワンタイム (1 度だけ)、あるいは短期間だけ使用できるパスワードを利用した認証である。認証の度に認証情報が変化するため、不正ログイン、第 3 者によるなりすましの防止などのセキュリティ強化が期待できる。ワンタイムパスワードを生成するには、大きく 2 つ方法がある。カウンタベースのトークンでは、ベースシークレットを同期カウンタと組み合わせてワンタイムパスワードを生成する。クロックベースのトークンでは、同期クロックを使ってワンタイムパスワードを生成する。これらの手法はすべて、ワンタイムパスワードトークン内に格納されているランダムなベースシークレットを利用する。このベースシークレットを何らかの任意の値 (カウンタ、クロック、またはその両方と組み合わせることで、新しいパスワードを生成する。

2.1.1　S/Key 認証

S/Key [4] とは、Unix システム用のログオン技術として 1990 年代はじめに Bellcore 社が開発したワンタイムパスワードシステムである。技術的なコンセプトを最初に提案したのは Leslie Lamport で、これは 1981 年に論文 [5] として発表された。Lamport の方式では、ハッシュ関数を使ってサーバにパスワードを保存するため、パスワードをそのまま保存しない。また、認証の度に異なる情報が通信される特徴を持つ。Lamport の方式の手順を以下に示す。

初期化

・ユーザは乱数 x を選択

・ユーザはハッシュ関数 F を使って次の値を計算してサーバに安全な通信路で送信 (y_i は i 回目の認証情報)

$y_1 = F^{1000}, y_2 = F^{999}(x), \cdots, y_{1000} = F^1(x)$

認証手順

・ユーザは、$x_i = F^{1000-i}(x)$ をサーバに送信

・ サーバで $F(x_i)$ を計算し、y_i と比較

認証手順を繰り返すと、最終的には $x_{1000} = F^1(x)$ をユーザからサーバに送信するところで、再利用可能なパスワードが尽きてしまう。そのため、この段階でもう一度初期化処理を行う。

2.2　ハッシュ関数

ハッシュ関数 [6] とは、任意長のメッセージを入力すると、メッセージを代表する固定長の値を出力する関数である。ハッシュ関数の出力値をハッシュ値と呼ぶ。入力サイズが小さくても大きくても、出力サイズは一定になる。また、同じハッシュ関数に同じメッセージが入力されたとき、同じハッシュ値が出力されなければならない。さらに、入力値が大きくても高速でハッシュ値が計算できることが求められる。そのため、ハッシュ関数はリアルタイムな処理に使用できる。

2.2.1　ハッシュ関数の安全性

ハッシュ関数の標準的な安全性として、次の 3 つが挙げられる。

・ **一方向性 (原像計算困難性)**

一方向性とは、ハッシュ値が与えられたとき、元のメッセージを求めることが困難であることである。原像計算困難性とも呼ぶ。このように逆計算が困難な関数を、一般に一方向性関数という。

・ **第 2 原像計算困難性**

第 2 原像計算困難性とは、あるメッセージ (第 1 原像) とそのハッシュ値が与えられたとき、同一のハッシュ値になる別のメッセージ (第 2 原像) を計算することが困難であることである。

・ **衝突困難性**

衝突困難性とは、同じハッシュ値になるような 2 つの異なるメッセージを求めることが困難であることである。

これらの安全性を満たすハッシュ関数は、暗号的ハッシュ関数と呼ばれる [6]。

2.2.2　MD(Message Digest Algorythm)

MD4 [6] は 1990 年にリベスト (Rivest) によって提案されたハッシュ関数である。MD4、MD5 は 128 ビットのハッシュ値を出力する。特に MD4 は、後の専用ハッシュ関数 (例: MD5,SHA-1) の設計に大きな影響を与えた。

MD5 の衝突困難性は破られているが、現在のところ第 2 原像計算困難性は破られていない。しかし、衝突困難性が破られたことにより 、MD5 を利用しているアプリケーションでは様々な問題が発生している。 そのため、現在では MD5 の代わりに別の安全なハッシュ関数の使用 が推奨されている。

2.2.3　SHA(Secure Hash Algorythm)

SHA(Secure Hash Algorithm) [6] は米国国立標準技術研究所 (NIST) に よって米国標準として制定された SHS(Secure Hash Standard) のアルゴリズムの総称である。 1993 年に FIPS180 で SHA-0 が制定された。ところが、脆弱性が指摘されたため、 1995 年に SHA-1 に置き換わった。 SHA-1 は 264 ビット未満のメッセージを入力として、 160 ビットのハッシュ 値を出力します。内部で使用する圧縮関数の入力は 160+512 ビット (K は定 数なので入カサイズからは除外)、出力は 160 ビットであるため、ブロックサイズは 512 ビットである。 2002 年には SHA-1 のアルゴリズムに改良が加えられ、 160 ビットを超えるハッシュ値を生成できるようになった。 これらを SHA-256、SHA-384、 SHA-512 という (数字がハッシュ値のビッ ト長)。 2004 年には SHA-224 も発表された。これらの 4 つのアルゴリズムをまとめて SHA-2 と呼ぶ。

第3章　SAS-L 認証方式と実装

本章では、SAS-L [2] [7] について述べる。まず、SAS-L(4) [7] と SAS-L(3) [7] について説明する。

3.1　SAS-L(4)

まず、SAS の１つとして SAS-2 がある。この方式はワンタイムパスワード認証方式を用いてサーバとユーザ間で相互認証を実現し、S/Key などの従来方式と比較して、一方向性関数の適用回数が少なく、処理負荷の軽量さが特徴である。

SAS-L(4) は、従来方式の SAS-2 における被認証者側の演算負荷を改善する方式として 2018 年に清水明宏教授によって考案された。SAS-L(4) の認証情報の作成では、被認証者側における一方向性関数の適用回数は 0 回となり、排他的論理和、加算は数回行うのみで実現可能であることが大きな特徴である。SAS-L(4) は SAS-2 と同様に、初回認証の前に行われる登録フェーズと、認証フェーズに分かれているが、SAS-2 は一方向性変換関係の検証を認証の論拠にしているのに対して、SAS-L(4) は今回認証情報と次回認証情報の演算が同一になるかどうかで認証を行う。SAS-L(4) のアルゴリズムの詳細を次に示す。

3.1.1　定義と表記法

以下に、各フェーズを説明する際に使用する用語および記号の定義を記載する。

・認証情報　とは、サーバとユーザで共有された認証に用いる情報表す。

・今回認証情報　とは、現在のセッションで認証に用いる認証情報を表す。

・次回認証情報　とは、次のセッションで認証に用いる新しい認証情報を表す。

・i　とは、セッションの回数を表す 1 以上の整数値を表す。

・S とは、 User の識別子を表す。

・H(s) とは、s に対して一方向性関数を 1 度適用し、得た演算結果を表す。

・N_i とは、i 回目の認証時に生成される乱数を表す。

・A_1 とは、初回認証情報を表す。

・A_i とは、i 回目のセッションで用いる今回認証情報を表す。

・M_i とは、 i 回目の認証時に生成されるマスク値を表す。

・+ とは、ビット毎の算術加算を表す。

・$\oplus$ とは、 ビット毎の排他的論理和である。

3.1.2　初回登録フェーズ

初回登録フェーズは以下の処理を安全な通信で行う。図 4.1 に初回登録フェーズのフローチャートを示す。

サーバ

1. ユーザ識別子 S を設定・保持する。

2. 初回認証用の乱数 N_1 、マスク値 M_1 を生成し保存する。

3. 入力された S、 生成された N_1 を用いて 初回認証用の認証情報 $A_1 = H(S \oplus N_1)$ を演算し、保存する。

4. 安全なルートを経由し、A_1、M_1 をユーザへ送信する。

ユーザ

1. 受け取った A_1、M_1 を保存する。

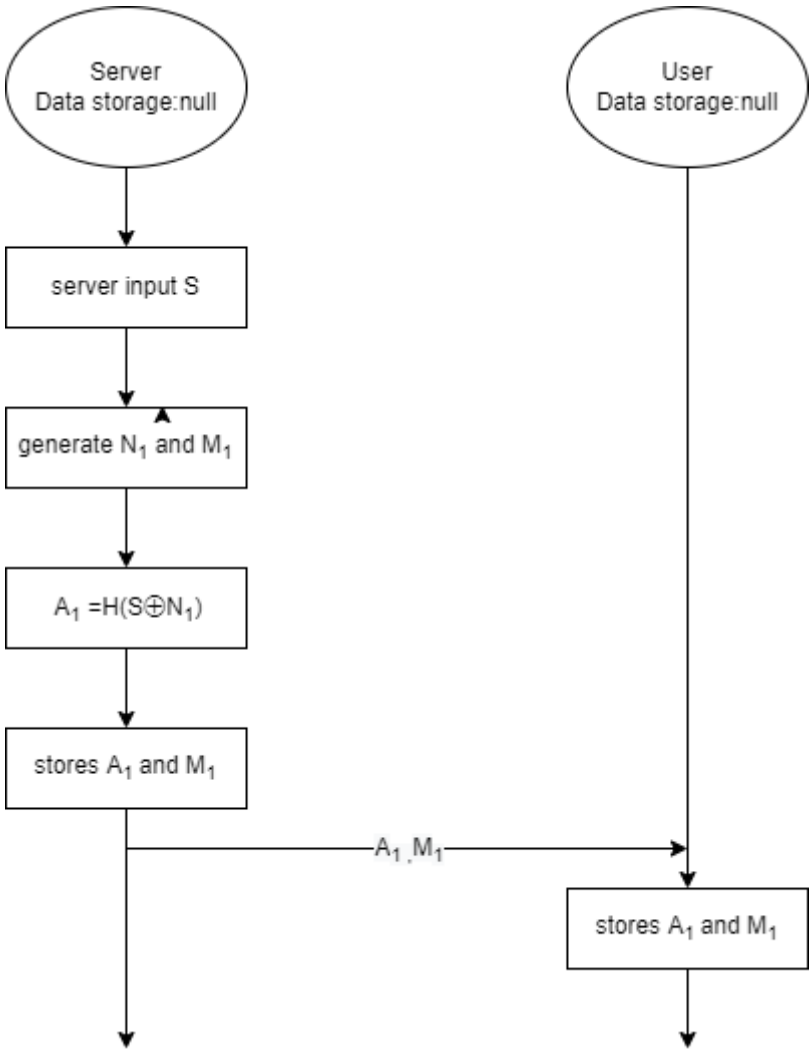

図 3.1　SAS-L(4) の登録フェーズ

3.1.3　認証フェーズ

図 3.2 に i 回目の認証フェーズのフローチャートを示す。

サーバ

1. ユーザ識別子　S 、乱数 N_i を入力する。
2. 次回認証用の乱数 N_{i+1} を生成する。
3. 乱数 N_{i+1} から次回認証用の認証情報 $A_{i+1} = H(S \oplus N_{i+1})$、$\alpha = A_i \oplus A_{i+1} \oplus M_i$ を演算する。
4. α をユーザへ送信する。以降、認証情報送信に使用するネットワークはインターネットなどの安全でないルートであっても問題はない。

5. 受信した β と 、 $A_i + A_{i+1}$ を比較し、 一致すれば認証が成功、 以下の処理が実行される。不一致ならば認証は不成立となり以下の処理は実行されない。

6. $M_{i+1} = A_i + M_i$ を演算する。M_{i+1} の演算を行う際に桁あふれを起こした場合はその値を切り捨てるものとする。

7. 保存されている N_i の代わりに N_{i+1} を新しい認証情報、M_i の代わりに $M_{i+1} = A_i + M_i$ を新しいマスク値として保存する。

ユーザ

1. 受信した α と保存された A_i を用いて $X = \alpha \oplus A_i \oplus M_i, \beta = X + A_i$ を演算する。

2. β をサーバへ送信する。

3. 保存されている A_i の代わりに $A_{i+1} = X$ を新しい認証情報、M_i の代わりに $M_{i+1} = A_i + M_i$ を保存する。

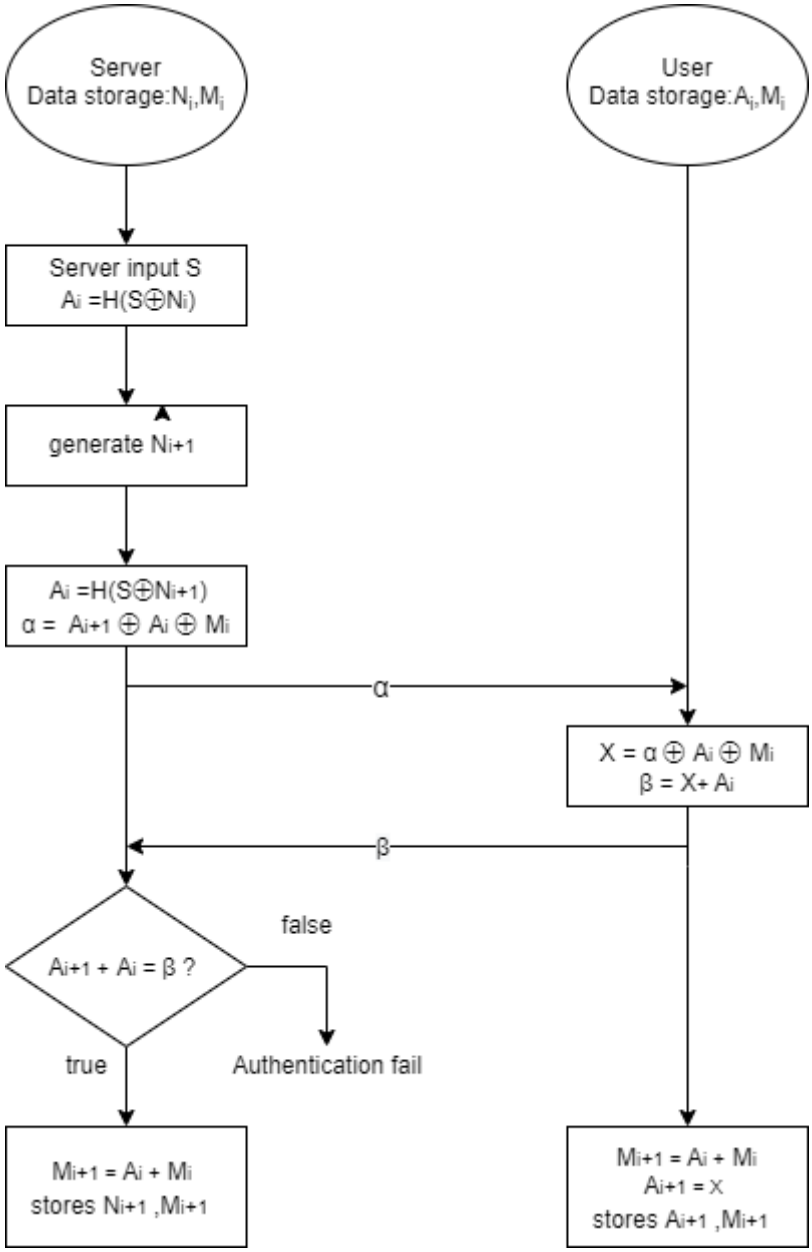

図 3.2　SAS-L(4) の認証フェーズ

3.2　SAS-L(3)

SAS-L(3) は、SAS-L(4) と同様に、従来方式の SAS-2 における被認証者側の演算負荷を改善する方式として清水明宏教授によって考案された。ただし、SAS-L(3) では、マスク値 (秘匿情報) を用いない。SAS-L(3) は SAS-L(4) と同様に、初回認証の前に行われる登録フェーズと、認証フェーズに分かれていて、今回認証情報と次回認証情報の演算が同一になるかどうかを検証することで認証を行う。SAS-L(3) のアルゴリズムの詳細を次に示す。

3.2.1　定義と表記法

以下に、各フェーズを説明する際に使用する用語および記号の定義を記載する。

・認証情報　とは、サーバとユーザで共有された認証に用いる情報表す。

・今回認証情報　とは、現在のセッションで認証に用いる認証情報を表す。

・次回認証情報　とは、次のセッションで認証に用いる新しい認証情報を表す。

・i　とは、セッションの回数を表す 1 以上の整数値を表す。

・S とは、 User の識別子を表す。

・H(s) とは、s に対して一方向性関数を 1 度適用し、得た演算結果を表す。

・N_i とは、i 回目の認証時に生成される乱数を表す。

・A_1 とは、初回認証情報を表す。

・A_i とは、i 回目のセッションで用いる今回認証情報を表す。

・+ とは、ビット毎の算術加算を表す。

・$\oplus$ とは、 ビット毎の排他的論理和である。

3.2.2　初回登録フェーズ

初回登録フェーズは以下の処理を安全な通信で行う。図 3.3 に初回登録フェーズのフローチャートを示す。

サーバ

1. ユーザ識別子 S を設定・保持する。
2. 初回認証用の乱数 N_1 を生成し保存する。
3. 入力された S、 生成された N_1 を用いて 初回認証用の認証情報 $A_1 = H(S \oplus N_1)$ を演算し、保存する。
4. 安全なルートを経由し、A_1 をユーザへ送信する。

ユーザ

1. 受け取った A_1 を保存する。

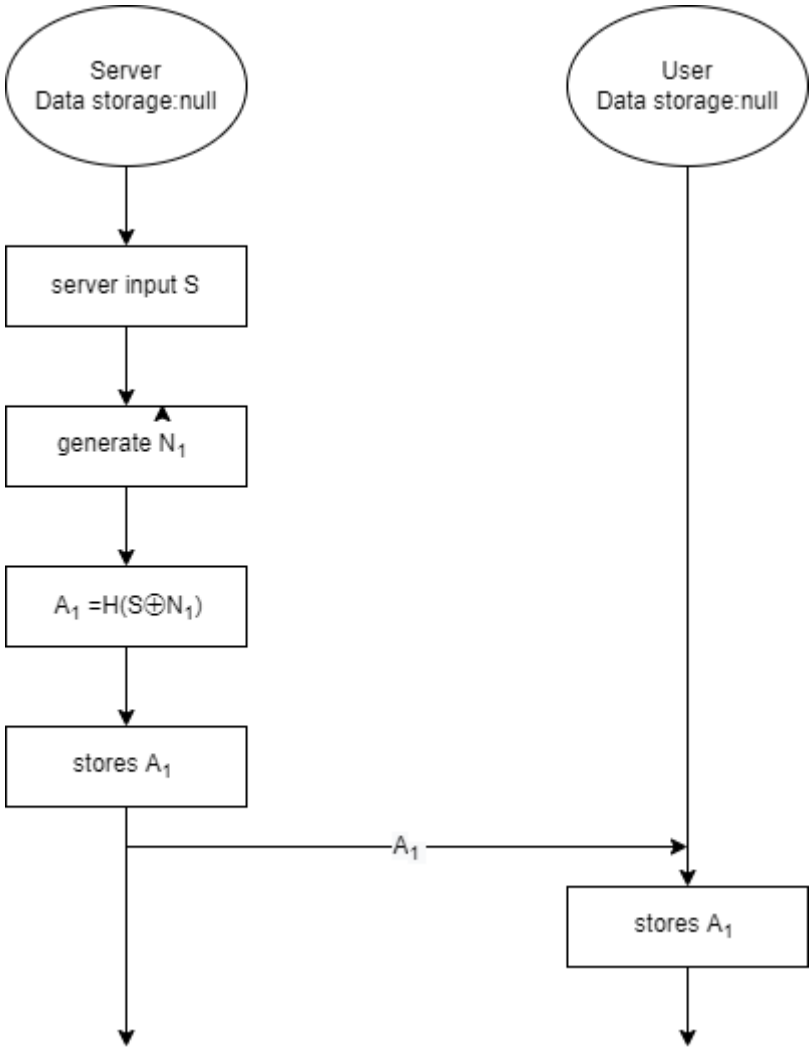

図 3.3　SAS-L(3) の登録フェーズ

3.2.3　認証フェーズ

図 3.4 に i 回目の認証フェーズのフローチャートを示す。

サーバ

1. ユーザ識別子 S 、乱数 N_i を入力する。

2. 次回認証用の乱数 N_{i+1} を生成する。

3. 乱数 N_{i+1} から次回認証用の認証情報 $A_{i+1} = H(S \oplus N_{i+1})$、$\alpha = A_i \oplus A_{i+1}$ を演算する。

4. α をユーザへ送信する。以降、認証情報送信に使用するネットワークはインターネットなどの安全でないルートであっても問題はない。

5. 受信した β と 、 $A_i + A_{i+1}$ を比較し、 一致すれば認証が成功、 以下の処理が実行される。不一致ならば認証は不成立となり以下の処理は実行されない。

6. 保存されている A_i の代わりに A_{i+1} を新しい認証情報として保存する。

ユーザ

1. 受信した α と保存された A_i を用いて $X = \alpha \oplus A_i, \beta = X + A_i$ を演算する。

2. β をサーバへ送信する。

3. 保存されている A_i の代わりに $A_{i+1} = X$ を新しい認証情報を保存する。

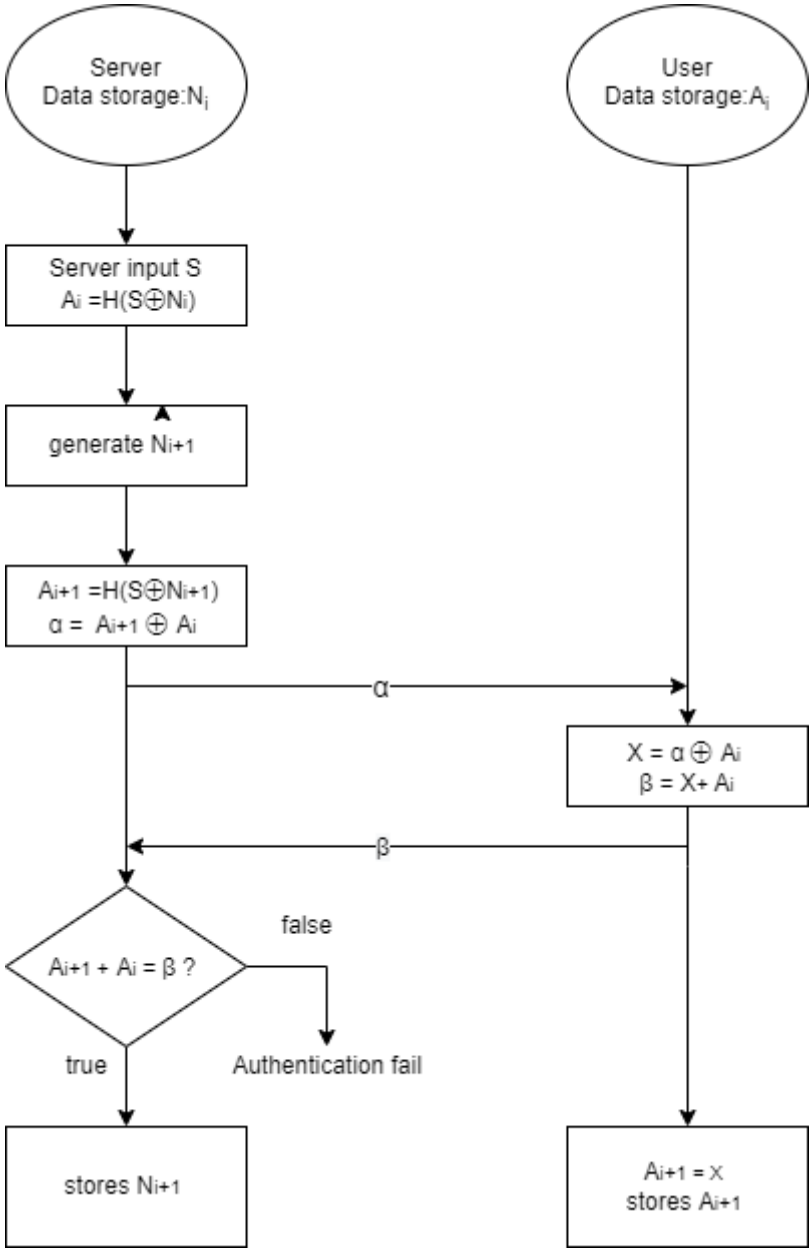

図 3.4　SAS-L(3) の認証フェーズ

3.2.4　SAS-L 認証方式の演算負荷比較

SAS-(4)、SAS-L(3) に要求される計算回数を比較した表 3.1 に示す。SAS-L(4) 認証方式は、ユーザ側において、一方向性変換の適用回数は 0 回、排他的論理和 (XOR) の適用回数が 2 回、加算の適用回数が 2 回である。サーバ側においては一方向性変換の適用回数は 2 回、排他的論理和の適用回数は 4 回、加算の適用回数は 2 回である。これに対して、SAS-L(3) はユーザ側において、一方向性変換の適用回数が 2 回、排他的論理和の適用回数が 1 回、加算

の適用回数が 1 回である。サーバ側においては一方向性変換の適用回数を 2 回、排他的論理和の適用回数は 3 回、加算の適用回数は 1 回である。SAS-L(3) は、マスク値 (秘匿情報) を用いないため、SAS-L(4) と比較して、排他的論理和の適用回数が 2 回、加算の適用回数が 2 回少ないことがわかる。よって理論上では、SAS-L(3) の方が演算負荷が軽量といえる。

表 3.1　演算負荷の比較表

	被認証者側			**認証者側**		
	一方向性関数	XOR	加算	一方向性関数	XOR	加算
SAS-L(4)	0	2	2	2	4	2
SAS-L(3)	0	1	1	2	3	1

3.3　安全性

SAS-(3) の安全性の検証方法について述べる。認証フェーズにおいてユーザとサーバ間でのやり取りするデータ α と β は、第 3 者は入手することができる。SAS-L(1) では、i 回目、i+1 回目、i+2 回目のデータ α と β を用いて i+3 回目のデータを作成可能であり、過去のデータを再利用できるリプレイアタックの脅威がある。リプレイアタックとは、ネットワークを盗聴し認証情報を取得し、それらを別の機会にシステムに接続させるために用いる攻撃である [?]。SAS-L(2)、SAS-L(4) では、マスク値 (秘匿情報) を付加することで、リプレイアタックの脅威を回避している [7]。 SAS-L(3) においては、安全性は未検証であるため、リプレイアタックの脅威を調べることで安全性の検証を行う。

まず SAS-L(3) のリプレイアタックの脅威において、SAS-L(1) 同様に過去 3 回分の α と β を利用されたときを検討する。例えば本実験例で、第 i 回目から第 i+2 回目の認証手順において、以下のデータが α と β として送信される。

i 回目

$$\alpha_i = A_i \oplus A_{i+1} \tag{3.1}$$

$$\beta_i = A_i + A_{i+1} \tag{3.2}$$

i+1 回目

$$\alpha_{i+1} = A_{i+1} \oplus A_{i+2} \tag{3.3}$$

$$\beta_{i+1} = A_{i+1} + A_{i+2} \tag{3.4}$$

i+2 回目

$$\alpha_{i+2} = A_{i+2} \oplus A_{i+3} \tag{3.5}$$

$$\beta_{i+2} = A_{i+2} + A_{i+3} \tag{3.6}$$

$\alpha_i \oplus \alpha_{i+1} \oplus \alpha_{i+2}$ により

$$A_i \oplus A_{i+3} \tag{3.7}$$

が生成できる。また、$\beta_i - \beta_{i+1} + \beta_{i+2}$ により

$$A_i + A_{i+3} \tag{3.8}$$

が得られる。SAS-L(1) の場合、式 (3.7)、(3.8) を次の i+3 回目の α_{i+3} と β_{i+3} として入力すれば、認証が成立する [7]。

SAS-L(3) について i+3 回目の認証を行うことを考える。サーバー側で作成した α_{i+3} が攻撃者により (3.7) 式に置き換えられたとする。このとき攻撃者は次回認証情報を A_i にしようとしている。ユーザー側は (3.7) 式を受け取り、ユーザー側で (3.8) 式が計算される。ユーザーから (3.8) 式がサーバ側に返されるが、サーバー側では正しい次回認証情報 A_{i+3} を保存しており正しい β_{i+3} つまり $A_{i+4} + A_{i+3}$ が計算できる。この計算結果は不正な (3.8) 式とは異なるため認証が失敗する。つまり、SAS-L(3) では SAS-L(1) で有効だったリプレイアタックは失敗する。

第4章　実験方法および評価実験

4.1　実装方法

SAS-L(4),SAS-L(3) のプロトコルをＣ言語で実装する。また、ユーザとサーバ間の通信はソケット通信（Linux の標準ライブラリの socket 関数）で実装した。

一方向性関数は、代表的なアルゴリズムとして MD5、SHA256、SHA512 などが存在する。MD5 は同じハッシュ値を持つ入力値のペアが、一般的な PC で探索できてしまい、安全性が低いとされる。SHA256 は、現在の安全性の基準である 128 ビット安全性を満足するため、本実装では一方向性関数として OpenSSL 内の SAH256 を用いる。乱数生成では、Ｃ言語で実装可能な OpenSSL 内の RAND_bytes() 関数を用いる。

本研究では、CPU 時間を計測し SAS-L(3) の計算時間の評価を行う。CPU 時間の計測方法として Linux の標準ライブラリの clock 関数を用いる。clock 関数は CPU でかかった時間を計測するため、処理内容について複数回実行し評価する。

実行方法を付録 A に、ソースコードを付録 B に示す。

4.2　計算機環境

計算機環境について以下の表 4.1 に示す。

表 4.1　計算機環境

CPU	Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz
メモリ	8GB
ディスク	20GB
OS	Linux7.9.2009

4.3　実験結果

SAS-L(3),SAS-L(4) において、認証フェーズを 1000 回繰り返す実験を 10 回行い、計測した CPU 計算時間の平均時間を表 4.2、表 4.3 に示す。演算 1 では、加算、排他的論理和、一方向性関数、演算 2 では、加算、排他的論理和が含まれる。また、表 4.2、表 4.3 を棒グラフにしたものを図 4.1、図 4.2 に示す。

表 4.2　サーバ側の各認証方式における認証部 1000 回分の CPU 時間 (秒)

サーバ	全体	乱数生成	通信	演算 1
SAS-L(4)	0.0259	0.0048	0.0141	0.0070
SAS-L(3)	0.0255	0.0048	0.0140	0.0067

表 4.3　ユーザ側の各認証方式における認証部 1000 回分の CPU 時間

ユーザ	全体	通信	演算 2
SAS-L(4)	0.0195	0.0160	0.0035
SAS-L(3)	0.0191	0.0159	0.0032

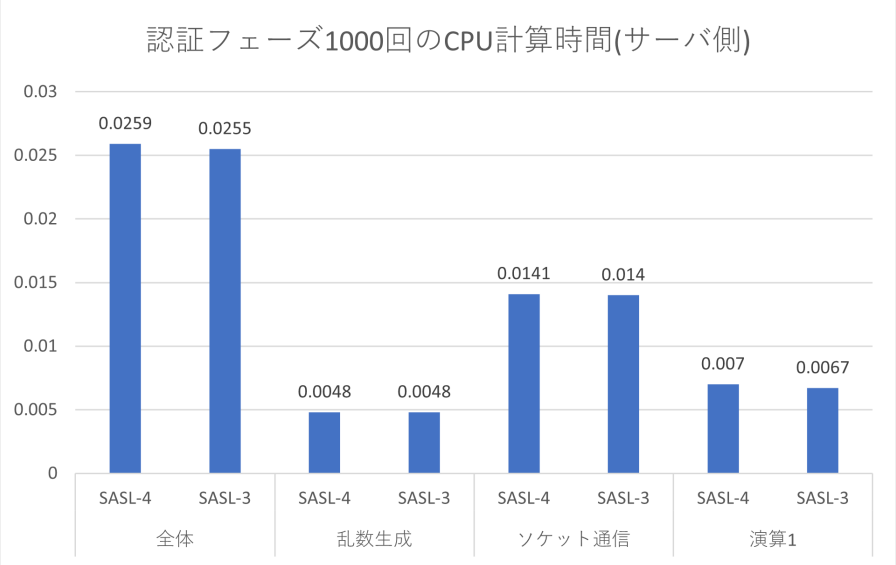

図 4.1　認証フェーズ 1000 回の CPU 計算時間 (サーバ側)

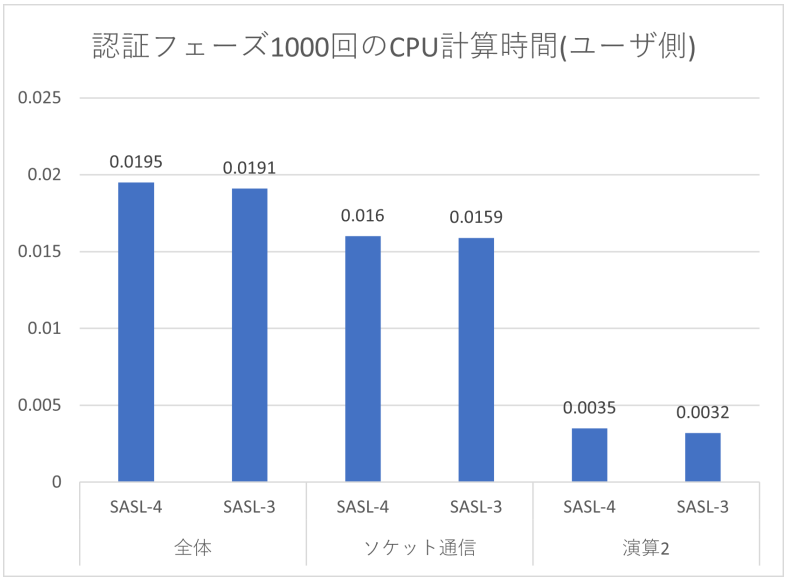

図 4.2　認証フェーズ 1000 回の CPU 計算時間 (ユーザ側)

サーバーおよびユーザーの認証フェーズについて、SAS-L(3) が SAS-L(4) に比べて排他的論理和が 1 回、加算が 1 回少ない。そのため、表 4.2 および表 4.3 に示すように、SAS-L(3) のほうが短い計算時間で認証が行えることが確認できた。従って、演算回数、計算時間、安全性の面で SAS-L(3) の有効性を確認することができた。　但し、ユーザーの認証フェーズの演算 2（排他的論理和と加算）については、表 3.1 に示すように、SAS-L(3) の計算コストは SAS-L(4) の計算コストの 1/2 であるはずである。しかし計算時間を計測すると、SAS-L(3) の演算 2 の計算時間は SAS-L(4) の演算 2 の計算時間より 0.0003 秒だけ早く計算できるという結果になった。この点についてはより精度の高い時間の評価方法を用いて検討する必要がある。

第5章　結論

本研究では、ワンタイムパスワード認証方式 SAS-L(3）の安全性を評価し、SAS-L(1) では存在したリプレイアタックの脅威が SAS-L(3) に存在しないことを確認した。さらに、SAS-L(3) を計算機上に実装し、SAS-L(4) と比較してサーバおよびユーザの認証フェーズの計算時間が短縮できることを示した。　本研究では SAS-L(3) を Linux を搭載した PC 上のみで検証したが、今後の課題として、

・ SAS-L(3) の IoT 機器への実装や計算時間の評価

があげられる。

謝辞

本研究を遂行するにあたり、常日頃より丁寧なご指導を頂きました、高橋寛教授、甲斐博准教授、王森レイ講師に深く御礼申し上げます。そして、本研究に際し、ご審査頂きました遠藤慶一准教授、宇戸寿幸准教授に深く御礼申し上げます。最後に、ご支援いただいた本学情報工学科の諸先生方、研究室の皆様に厚く御礼申し上げます。

参考文献

［1］ サイバーセキュリティタスクフォース事務局，“サイバー攻撃の最近の動向等について，令和 2 年 12 月 3 日”
https://www.soumu.go.jp/main_content/000722477.pdf(参照:2022-1-15)

［2］ 清水明宏，“SAS-L ワンタイムパスワード認証方式について，” 高知工科大学, preprint, 2020.

［3］ Richard E. Smith，“認証技術　パスワードから公開鍵まで，” オーム社，2003.

［4］ Bellcore，“The S/KEY One-Time Password System，” RFC1760 Feb 1995.

［5］ Leslie Lamport，“Password authentication with insecure communication，” Communications of the ACM, Volume 24,Issue 11,Nov.1981,pp770-772.

［6］ IPUSIRON，“暗号技術のすべて，” 翔泳社，2017.

［7］ 清水明宏，“SAS ワンタイムパスワード認証方式について，” 高知工科大学, スライド, 2021,10.

付 録A　その他

A.1　実行方法

認証情報が 256bit の場合のコンパイル方法と実行方法は次の通りである。

認証者側

コンパイル方法は次の通りである。ここでは、ソースファイル名を sasl-4_server.c、実行ファイル名を server とする。openSSL の sha.h をインクルードするため、リンクオプションとして-lcrypto を追加する。

```
% gcc sasl2server.c -o server -lcrypto
```

次に、実行方法は次の通りである。

```
% ./server
```

被認証者側

コンパイル方法は次の通りである。ここでは、ソースファイル名を sas-4_client.c、実行ファイル名を client とする。次に示す例ではリンクオプション-lcrypto が追加されているが、SAS-L(4) の被認証者側プログラムでは追加する必要はない。

```
% gcc sas2client.c -o client -lcrypto
```

次に、実行方法は次の通りである。

```
% ./client
```

SAS-L(3) においても、ソースファイル名が異なるだけで、同様の方法である。

付 録B　プログラムリスト

B.1　SAS-L(4)

B.1.1　サーバ側の動作

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <openssl/sha.h>
#include <openssl/rand.h>
#include <time.h>

#define SERVER_ADDR "127.0.0.1"
#define SERVER_PORT 8080
#define BUF_SIZE 32

void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len);
void add(unsigned char result[],unsigned char x[],unsigned char y[],int len);
void show(unsigned char s[],int len);
int transfer_first(int sock,unsigned char S[],unsigned char N1[],unsigned char M1[]);
int transfer_n(int sock,unsigned char S[],unsigned char N1[],unsigned char Mn[],clock_t *rand_clock,clock_t *sock_clock);

int main(void) {
  int w_addr, c_sock;
  struct sockaddr_in a_addr;
  unsigned char S[BUF_SIZE] = {0};//password
  unsigned char N1[BUF_SIZE] = {0};//rondom
  unsigned char M1[BUF_SIZE] = {0};//
  unsigned char Nn[BUF_SIZE] = {0};//rondom
  unsigned char Mn[BUF_SIZE] = {0};

  FILE *file;
  int i,cnt = 0;
  clock_t s_clock,e_clock,sum_clock = 0,rand_clock = 0,sock_clock = 0;

  /* ソケットを作成 */
  w_addr = socket(AF_INET, SOCK_STREAM, 0);
  if (w_addr == -1) {
    printf("socket error\n");
    return -1;
  }

  /* 構造体を全て 0 にセット */
  memset(&a_addr, 0, sizeof(struct sockaddr_in));

  /* サーバーの IP アドレスとポートの情報を設定 */
  a_addr.sin_family = AF_INET;
  a_addr.sin_port = htons((unsigned short)SERVER_PORT);
  a_addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

  /* ソケットに情報を設定 */
  if (bind(w_addr, (const struct sockaddr *)&a_addr, sizeof(a_addr)) == -1) {
    printf("bind error\n");
    close(w_addr);
    return -1;
  }

  /* ソケットを接続待ちに設定 */
  if (listen(w_addr, 3) == -1) {
    printf("listen error\n");
    close(w_addr);
    return -1;
  }

  //接続要求の受け付け（接続要求くるまで待ち）
  printf("Waiting connect...\n");
  c_sock = accept(w_addr, NULL, NULL);
  if (c_sock == -1) {
    printf("accept error\n");
    close(w_addr);
    return -1;
  }
```

```
  printf("Connected!!\n");

  printf("---初回登録---\n");
  //接続済のソケットでデータのやり取り
  transfer_first(c_sock,S,N1,M1);


  for(i = 0;i < BUF_SIZE;i++){
    Nn[i] = N1[i];
    Mn[i] = M1[i];
  }

  //------------------------------------------------
  while(cnt !=1000){
    //接続要求の受け付け（接続要求くるまで待ち）
    printf("Waiting connect...\n");
    c_sock = accept(w_addr, NULL, NULL);
    if (c_sock == -1) {
      printf("accept error\n");
      close(w_addr);
      return -1;
    }
    printf("Connected!!\n");

    printf("---N 回目認証---\n");

    printf("cnt:%d\n",cnt);

    s_clock  = clock();
    //接続済のソケットでデータのやり取り
    transfer_n(c_sock,S,Nn,Mn,&rand_clock,&sock_clock);

    e_clock  = clock();
    sum_clock = sum_clock + (e_clock - s_clock);

    cnt++;
  }

  //ソケット通信をクローズ
  close(c_sock);

  printf("cnt:%d, sum_clock  :%f\n",cnt,(double)(sum_clock)/CLOCKS_PER_SEC);
  printf("cnt:%d, rand_clock :%f\n",cnt,(double)(rand_clock)/CLOCKS_PER_SEC);
  printf("cnt:%d, sock_clock :%f\n",cnt,(double)(sock_clock)/CLOCKS_PER_SEC);
  printf("cnt:%d, sum-ran-soc:%f\n",cnt,(double)(sum_clock - rand_clock - sock_clock)/CLOCKS_PER_SEC);

  /* 接続待ちソケットをクローズ */
  close(w_addr);

  return 0;
}

void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] ^ y[i];
  }
}

void add(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] + y[i];
  }
}

void show(unsigned char s[],int len){
  int i;
  for(i=0;i< len;i++){
    printf("%02X",s[i]);
  }
  printf("\n");
}

int transfer_first(int sock,unsigned char S[],unsigned char N1[],unsigned char M1[]) {
  unsigned char send_buf[BUF_SIZE*2]={0};
  char recv_buf          ;
  int send_size, recv_size;
  int i;

  SHA256_CTX sha_ctx;//コンテキストを生成

  unsigned char xor_result[BUF_SIZE] = {0};//result_xor
  unsigned char A1[BUF_SIZE] ={0};//hash
  FILE *fp;


  //  printf("パスワードを入力してください please password\n");
  //scanf("%s",S);


  //password  data -> S
  RAND_bytes(S,BUF_SIZE);
  //ランダムデータ random data -> N1
  RAND_bytes(N1,BUF_SIZE);
  //make M1
  RAND_bytes(M1,BUF_SIZE);
```

```
  //xor
  xor(xor_result,S,N1,sizeof(xor_result));

  //hash
  SHA256_Init(&sha_ctx);
  SHA256_Update(&sha_ctx,xor_result,sizeof(xor_result));
  SHA256_Final(A1,&sha_ctx);

  //print
  //printf("S     = ");
  //show(S,sizeof(S));
  //printf("N1    = ");
  //show(N1,sizeof(N1));
  //printf("A1    = ");
  //show(A1,sizeof(A1));
  //printf("M1    = ");
  //show(M1,sizeof(M1));

  //---------connect A1 M1----------
  for(i=0;i< BUF_SIZE;i++){
    send_buf[i] = A1[i];
  }
  int k = 0;
  for(i=BUF_SIZE;i< BUF_SIZE*2;i++){
    send_buf[i] = M1[k];
    k++;
  }
  //----------------------------------------

  /* 文字列を送信 */
  send_size = send(sock, send_buf, sizeof(send_buf) + 1, 0);
  if (send_size == -1) {
    printf("send error\n");
    return 0;
  }
  /* サーバーからの応答を受信 */
  recv_size = recv(sock, &recv_buf, 1, 0);
  if (recv_size == -1) {
    printf("recv error\n");
    return 0;
  }
  if (recv_size == 0) {
    /* 受信サイズが 0 の場合は相手が接続閉じていると判断 */
    printf("connection ended\n");
    return 0;
  }
  /* 応答が 0 の場合はデータ送信終了 */
  if (recv_buf == 0) {
    printf("接続を終了しました\n");
    return 0;
  }

  return 0;
}


int transfer_n(int sock,unsigned char S[],unsigned char Nn[],unsigned char Mn[],clock_t *rand_clock,clock_t *sock_clock) {
  unsigned char send_buf[BUF_SIZE]={0};
  unsigned char recv_buf[BUF_SIZE]={0};
  int send_size, recv_size;
  int i;

  SHA256_CTX sha_ctx;//コンテキストを生成
  unsigned char Nn_next[BUF_SIZE] = {0};//N n+1
  unsigned char xor_result[BUF_SIZE] = {0};//result_xor
  unsigned char An[BUF_SIZE] ={0};
  unsigned char An_next[BUF_SIZE] ={0};
  unsigned char alpha[BUF_SIZE] ={0};
  unsigned char beta[BUF_SIZE] ={0};
  unsigned char Mn_next[BUF_SIZE] = {0};
  clock_t s_time,e_time;
  FILE *fp,*fp2;

  //printf("パスワードを入力してください please password\n");
  //scanf("%s",S);

  //printf("S=");
  //show(S,BUF_SIZE);
  //printf("Nn=");
  //show(Nn,BUF_SIZE);
  //printf("Mn=");
  //show(Mn,BUF_SIZE);


  /*----------make An----------*/
  //xor
  xor(xor_result,S,Nn,sizeof(xor_result));

  //hash
  SHA256_Init(&sha_ctx);
  SHA256_Update(&sha_ctx,xor_result,sizeof(xor_result));
  SHA256_Final(An,&sha_ctx);

  //----------generate Nn next----------
  s_time = clock();
```

```
RAND_bytes(Nn_next,sizeof(Nn_next));

e_time = clock();
*rand_clock += (e_time - s_time);

//----------make An_next----------
//xor
xor(xor_result,S,Nn_next,sizeof(xor_result));

//hash
SHA256_Init(&sha_ctx);
SHA256_Update(&sha_ctx,xor_result,sizeof(xor_result));
SHA256_Final(An_next,&sha_ctx);

//----------make alpha----------
xor(alpha,An_next,An,BUF_SIZE);
xor(alpha,alpha,Mn,BUF_SIZE);

//----------send_buf <- alpha----------
for(i=0;i< BUF_SIZE;i++){
  send_buf[i] = alpha[i];
}

/*
printf("An      = ");
show(An,BUF_SIZE);
printf("Mn      = ");
show(Mn,BUF_SIZE);
printf("Nn_next = ");
show(Nn_next,sizeof(Nn_next));
printf("An_next = ");
show(An_next,BUF_SIZE);
printf("alpha   = ");
show(alpha,BUF_SIZE);
printf("send    = ");
show(send_buf,BUF_SIZE+1);
//----------make beta----------
*/
add(beta,An_next,An,BUF_SIZE);
/*
printf("An+1 +An= ");
show(beta,BUF_SIZE);
*/

//----------send alpha  ----------
s_time = clock();

send_size = send(sock, send_buf , sizeof(send_buf) + 1, 0);
if (send_size == -1) {
  printf("send error\n");
  return 0;
}
// client からの応答を受信
recv_size = recv(sock, recv_buf, BUF_SIZE, 0);
if (recv_size == -1) {
  printf("recv error\n");
  return 0;
}

if (recv_size == 0) {
  //受信サイズが 0 の場合は相手が接続閉じていると判断
  printf("connection ended\n");
  return 0;
}
//応答が 0 の場合はデータ送信終了
if (recv_buf == 0) {
  printf("接続を終了しました\n");
  return 0;
}

e_time = clock();
*sock_clock += (e_time - s_time);

//------------------------------------------------

int flag = 0;
//xor
for(i=0;i< BUF_SIZE;i++){
  if((recv_buf[i] ^ beta[i])!=0){
    flag= 1;
    break;
  }
  else{
    flag = 0;
  }
}

if(flag == 0){
  printf("認証成功\n");

  //----------make Mn_next----------
  //add
  add(Mn_next,An,Mn,sizeof(Mn_next));

  for(i = 0;i < BUF_SIZE;i++){
    Nn[i] = Nn_next[i];
    Mn[i] = Mn_next[i];
```

```
    }

  }
  else{
    printf("認証失敗\n");
  }

  return 0;
}

%\end{lstlisting}
```

B.1.2 ユーザ側の動作

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <openssl/sha.h>
#include <openssl/rand.h>
#include <time.h>

#define SERVER_ADDR "127.0.0.1"
#define SERVER_PORT 8080
#define BUF_SIZE 32

void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len);
void add(unsigned char result[],unsigned char x[],unsigned char y[],int len);
void show(unsigned char s[],int len);
int transfer_first(int sock,unsigned char A1[],unsigned char M1[]);
int transfer_n(int sock,unsigned char An[],unsigned char Mn[],clock_t *sock_clock);


int main(void) {
  int sock;
  struct sockaddr_in addr;
  FILE *file;
  int i,cnt=0;
  unsigned char A1[BUF_SIZE] = {0};
  unsigned char M1[BUF_SIZE] = {0};
  unsigned char An[BUF_SIZE] = {0};
  unsigned char Mn[BUF_SIZE] = {0};
  clock_t s_clock,e_clock,sum_clock = 0,sock_clock = 0;

  //------------------------------------------------
  /* ソケットを作成*/
  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock == -1) {
    printf("socket error\n");
    return -1;
  }

  /* 構造体を全て 0 にセット */
  memset(&addr, 0, sizeof(struct sockaddr_in));

  /* サーバーの IP アドレスとポートの情報を設定*/
  addr.sin_family = AF_INET;
  addr.sin_port = htons((unsigned short)SERVER_PORT);
  addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

  /* サーバーに接続要求送信 */
  printf("接続を開始します...\n");
  if (connect(sock, (struct sockaddr*)&addr, sizeof(struct sockaddr_in)) == -1) {
    printf("connect error\n");
    close(sock);
    return -1;
  }
  printf("接続が完了しました!\n");
  //------------------------------------------------
  printf("---初回登録---\n");
  /* 初回登録時の接続済のソケットでデータのやり取り */

  transfer_first(sock,A1,M1);

  for(i = 0;i < BUF_SIZE;i++){
    An[i] = A1[i];
    Mn[i] = M1[i];
  }

  /* ソケット通信をクローズ */
  close(sock);

  printf("---N 回目認証---\n");

  while(cnt !=1000){

    //ソケットを作成
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
```

```
      printf("socket error\n");
      return -1;
    }

    //構造体を全て 0 にセット
    memset(&addr, 0, sizeof(struct sockaddr_in));

    //サーバーの IP アドレスとポートの情報を設定
    addr.sin_family = AF_INET;
    addr.sin_port = htons((unsigned short)SERVER_PORT);
    addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

    /* サーバーに接続要求送信 */
    printf("接続を開始します...\n");
    if (connect(sock, (struct sockaddr*)&addr, sizeof(struct sockaddr_in)) == -1) {
      printf("connect error\n");
      close(sock);
      return -1;
    }
    printf("接続が完了しました!\n");

    printf("cnt:%d\n",cnt);

    //--------------------------------------------
    s_clock = clock();

    /* N 回目認証時の接続済のソケットでデータのやり取り */
    transfer_n(sock,An,Mn,&sock_clock);

    e_clock = clock();

    sum_clock = sum_clock + (e_clock - s_clock);


    /* ソケット通信をクローズ */
    close(sock);
    cnt++;
  }

  printf("cnt:%d, sum_clock :%f\n",cnt,(double)(sum_clock)/CLOCKS_PER_SEC);
  printf("cnt:%d, sock_clock:%f\n",cnt,(double)(sock_clock)/CLOCKS_PER_SEC);
  printf("cnt:%d, sock-sum  :%f\n",cnt,(double)(sum_clock - sock_clock)/CLOCKS_PER_SEC);
  return 0;
}


void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] ^ y[i];
  }
}

void add(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] + y[i];
  }
}

void show(unsigned char s[],int len){
  int i;

  for(i=0;i< len;i++){
    printf("%02X",s[i]);
  }
  printf("\n");
}

int transfer_first(int sock,unsigned char A1[],unsigned char M1[]) {
  int recv_size, send_size,i;
  unsigned char recv_buf[BUF_SIZE*2];
  char send_buf;

  FILE *fp;

  /* クライアントから文字列を受信 */
  recv_size = recv(sock, recv_buf, BUF_SIZE*2, 0);
  if (recv_size == -1) {
    printf("recv error\n");
    return 0;
  }
  if (recv_size == 0) {
    /* 受信サイズが 0 の場合は相手が接続閉じていると判断 */
    printf("connection ended\n");
    return 0;
  }

  //A1
  for(i = 0;i < BUF_SIZE;i++){
    A1[i] = recv_buf[i];
  }

  //M1
  int k = 0;
  for(i = BUF_SIZE;i < BUF_SIZE*2;i++){
    M1[k] = recv_buf[i];
```

```
    k++;
  }

  //--------------------------------------------
  /* 接続終了を表す 0 を送信 */
  send_buf = 0;
  send_size = send(sock, &send_buf, 1, 0);
  if (send_size == -1) {
    printf("send error\n");
    return 0;
  }

  printf("接続を終了しました\n");
  return 0;

}

int transfer_n(int sock,unsigned char An[],unsigned char Mn[],clock_t *sock_clock) {
  int recv_size, send_size,i;
  unsigned char recv_buf[BUF_SIZE]={0};
  unsigned char send_buf[BUF_SIZE]={0};
  unsigned char An_next[BUF_SIZE] ={0};
  unsigned char Mn_next[BUF_SIZE] ={0};
  unsigned char alpha[BUF_SIZE] ={0};
  unsigned char beta[BUF_SIZE] ={0};
  clock_t s_time,e_time;
  FILE *fp;

  //----------receve alpha----------
  s_time = clock();

  // server から文字列を受信
  recv_size = recv(sock, recv_buf, BUF_SIZE, 0);
  if (recv_size == -1) {
    printf("recv error\n");
    return 0;
  }
  if (recv_size == 0) {
    // 受信サイズが 0 の場合は相手が接続閉じていると判断
    printf("connection ended\n");
    return 0;
  }

  e_time = clock();
  *sock_clock += (e_time - s_time);

  //alpha
  for(i = 0;i < BUF_SIZE;i++){
    alpha[i] = recv_buf[i];
  }

  //----------make beta----------
  xor(beta,alpha,An,BUF_SIZE);
  xor(An_next,beta,Mn,BUF_SIZE);//make An_next
  add(beta,An_next,An,BUF_SIZE);

  /*
  //print
  printf("An      = ");
  show(An,BUF_SIZE);
  printf("An+1    = ");
  show(An_next,BUF_SIZE);
  printf("alpha   = ");
  show(alpha,BUF_SIZE);
  printf("beta?   = ");
  show(beta,BUF_SIZE);
  */


  //----------send beta---------
  s_time = clock();

  for(i = 0;i < BUF_SIZE;i++){
    send_buf[i] = beta[i];
  }
  send_size = send(sock, send_buf, BUF_SIZE, 0);
  if (send_size == -1) {
    printf("send error\n");
    return -1;
  }

  e_time = clock();
  *sock_clock += (e_time - s_time);

  //----------make Mn_next----------
  add(Mn_next,An,Mn,sizeof(Mn_next));

  for(i = 0;i < BUF_SIZE;i++){
    An[i] = An_next[i];
    Mn[i] = Mn_next[i];
  }

  printf("接続を終了しました\n");
  return 0;

}
```

B.2 SAS-L(3)

B.2.1 サーバ側の動作

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <openssl/sha.h>
#include <openssl/rand.h>
#include <time.h>

#define SERVER_ADDR "127.0.0.1"
#define SERVER_PORT 8080
#define BUF_SIZE 32

void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len);
void add(unsigned char result[],unsigned char x[],unsigned char y[],int len);
void show(unsigned char s[],int len);
int transfer_first(int sock,unsigned char S[],unsigned char N1[]);
int transfer_n(int sock,unsigned char S[],unsigned char N1[],clock_t *rand_clock,clock_t *sock_clock);

int main(void) {
  int w_addr, c_sock;
  struct sockaddr_in a_addr;
  unsigned char S[BUF_SIZE] = {0};//password
  unsigned char N1[BUF_SIZE] = {0};//rondom
  unsigned char Nn[BUF_SIZE] = {0};//rondom

  FILE *file;
  int i,cnt = 0;
  clock_t s_clock,e_clock,sum_clock = 0,rand_clock = 0,sock_clock = 0;

    /* ソケットを作成 */
    w_addr = socket(AF_INET, SOCK_STREAM, 0);
    if (w_addr == -1) {
        printf("socket error\n");
        return -1;
    }

    /* 構造体を全て 0 にセット */
    memset(&a_addr, 0, sizeof(struct sockaddr_in));

    /* サーバーの IP アドレスとポートの情報を設定 */
    a_addr.sin_family = AF_INET;
    a_addr.sin_port = htons((unsigned short)SERVER_PORT);
    a_addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

    /* ソケットに情報を設定 */
    if (bind(w_addr, (const struct sockaddr *)&a_addr, sizeof(a_addr)) == -1) {
        printf("bind error\n");
        close(w_addr);
        return -1;
    }

    /* ソケットを接続待ちに設定 */
    if (listen(w_addr, 3) == -1) {
        printf("listen error\n");
        close(w_addr);
        return -1;
    }

    //接続要求の受け付け（接続要求くるまで待ち）
    printf("Waiting connect...\n");
    c_sock = accept(w_addr, NULL, NULL);
    if (c_sock == -1) {
      printf("accept error\n");
      close(w_addr);
      return -1;
    }
    printf("Connected!!\n");

    printf("---初回登録---\n");
    //接続済のソケットでデータのやり取り
    transfer_first(c_sock,S,N1);


    for(i = 0;i < BUF_SIZE;i++){
      Nn[i] = N1[i];
    }

    //------------------------------------------------
    while(cnt !=1000){
      //接続要求の受け付け（接続要求くるまで待ち）
      printf("Waiting connect...\n");
      c_sock = accept(w_addr, NULL, NULL);
      if (c_sock == -1) {
printf("accept error\n");
close(w_addr);
return -1;
```

```
      }
      printf("Connected!!\n");

      printf("---N 回目認証---\n");

      printf("cnt:%d\n",cnt);
      s_clock  = clock();

      //接続済のソケットでデータのやり取り
      transfer_n(c_sock,S,Nn,&rand_clock,&sock_clock);

      e_clock  = clock();
      sum_clock += (e_clock - s_clock);

      cnt++;
    }

    //ソケット通信をクローズ
    close(c_sock);

    printf("cnt:%d, sum_clock  :%f\n",cnt,(double)(sum_clock)/CLOCKS_PER_SEC);
    printf("cnt:%d, rand_clock :%f\n",cnt,(double)(rand_clock)/CLOCKS_PER_SEC);
    printf("cnt:%d, sock_clock :%f\n",cnt,(double)(sock_clock)/CLOCKS_PER_SEC);
    printf("cnt:%d, sum-ran-soc:%f\n",cnt,(double)(sum_clock - rand_clock - sock_clock)/CLOCKS_PER_SEC);
    /* 接続待ちソケットをクローズ */
    close(w_addr);

    return 0;
}

void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] ^ y[i];
  }
}

void add(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] + y[i];
  }
}

void show(unsigned char s[],int len){
  int i;
  for(i=0;i< len;i++){
    printf("%02X",s[i]);
  }
  printf("\n");
}

int transfer_first(int sock,unsigned char S[],unsigned char N1[]) {
  unsigned char send_buf[BUF_SIZE]={0};
  char recv_buf          ;
  int send_size, recv_size;
  int i;

  SHA256_CTX sha_ctx;//コンテキストを生成

  unsigned char xor_result[BUF_SIZE] = {0};//result_xor
  unsigned char A1[BUF_SIZE] ={0};//hash
  FILE *fp;


  //  printf("パスワードを入力してください please password\n");
  //scanf("%s",S);


  //password  data -> S
  RAND_bytes(S,BUF_SIZE);
  //ランダムデータ random data -> N1
  RAND_bytes(N1,BUF_SIZE);

  //xor
  xor(xor_result,S,N1,sizeof(xor_result));

  //hash
  SHA256_Init(&sha_ctx);
  SHA256_Update(&sha_ctx,xor_result,sizeof(xor_result));
  SHA256_Final(A1,&sha_ctx);

  //print
  //printf("S     = ");
  //show(S,sizeof(S));
  //printf("N1    = ");
  //show(N1,sizeof(N1));
  //printf("A1    = ");
  //show(A1,sizeof(A1));
  //printf("M1    = ");
  //show(M1,sizeof(M1));

  //---------connect A1 M1----------
  for(i=0;i< BUF_SIZE;i++){
    send_buf[i] = A1[i];
  }
```

```
  //---------------------------------------

  /* 文字列を送信 */
  send_size = send(sock, send_buf, sizeof(send_buf) + 1, 0);
  if (send_size == -1) {
    printf("send error\n");
    return 0;
  }
  /* サーバーからの応答を受信 */
  recv_size = recv(sock, &recv_buf, 1, 0);
  if (recv_size == -1) {
    printf("recv error\n");
    return 0;
  }
  if (recv_size == 0) {
    /* 受信サイズが 0 の場合は相手が接続閉じていると判断 */
    printf("connection ended\n");
    return 0;
  }
  /* 応答が 0 の場合はデータ送信終了 */
  if (recv_buf == 0) {
    printf("接続を終了しました\n");
    return 0;
  }

  return 0;
}


int transfer_n(int sock,unsigned char S[],unsigned char Nn[],clock_t *rand_clock,clock_t *sock_clock) {
  unsigned char send_buf[BUF_SIZE]={0};
  unsigned char recv_buf[BUF_SIZE]={0};
  int send_size, recv_size;
  int i;

  SHA256_CTX sha_ctx;//コンテキストを生成
  unsigned char Nn_next[BUF_SIZE] = {0};//N n+1
  unsigned char xor_result[BUF_SIZE] = {0};//result_xor
  unsigned char An[BUF_SIZE] ={0};
  unsigned char An_next[BUF_SIZE] ={0};
  unsigned char alpha[BUF_SIZE] ={0};
  unsigned char beta[BUF_SIZE] ={0};
  clock_t s_time,e_time;
  FILE *fp,*fp2;

  //printf("パスワードを入力してください please password\n");
  //scanf("%s",S);

  //printf("S=");
  //show(S,BUF_SIZE);
  //printf("Nn=");
  //show(Nn,BUF_SIZE);
  //printf("Mn=");
  //show(Mn,BUF_SIZE);


  /*----------make An----------*/
  //xor
  xor(xor_result,S,Nn,sizeof(xor_result));

  //hash
  SHA256_Init(&sha_ctx);
  SHA256_Update(&sha_ctx,xor_result,sizeof(xor_result));
  SHA256_Final(An,&sha_ctx);

  //----------generate Nn next----------
  s_time = clock();

  RAND_bytes(Nn_next,sizeof(Nn_next));

  e_time = clock();
  *rand_clock += (e_time - s_time);
  //----------make An_next----------
  //xor
  xor(xor_result,S,Nn_next,sizeof(xor_result));

  //hash
  SHA256_Init(&sha_ctx);
  SHA256_Update(&sha_ctx,xor_result,sizeof(xor_result));
  SHA256_Final(An_next,&sha_ctx);

  //----------make alpha----------
  xor(alpha,An_next,An,BUF_SIZE);

  //----------send_buf <- alpha----------
  for(i=0;i< BUF_SIZE;i++){
    send_buf[i] = alpha[i];
  }


  //printf("An      = ");
  //show(An,BUF_SIZE);
  //printf("Mn      = ");
  //show(Mn,BUF_SIZE);
  //printf("Nn_next = ");
  //show(Nn_next,sizeof(Nn_next));
  //printf("An_next = ");
```

```
  //show(An_next,BUF_SIZE);
  //printf("alpha   = ");
  //show(alpha,BUF_SIZE);

  //printf("send    = ");
  //show(send_buf,BUF_SIZE);
  //----------make beta----------

  add(beta,An_next,An,BUF_SIZE);

  //printf("An+1 +An= ");
  //show(beta,BUF_SIZE);


    //----------send alpha  ----------

  s_time = clock();
  send_size = send(sock, send_buf , sizeof(send_buf) + 1, 0);
  if (send_size == -1) {
    printf("send error\n");
    return 0;
  }
  // client からの応答を受信
  recv_size = recv(sock, recv_buf, BUF_SIZE, 0);
  if (recv_size == -1) {
    printf("recv error\n");
    return 0;
  }

  if (recv_size == 0) {
    //受信サイズが 0 の場合は相手が接続閉じていると判断
    printf("connection ended\n");
    return 0;
  }
  //応答が 0 の場合はデータ送信終了
  if (recv_buf == 0) {
    printf("接続を終了しました\n");
    return 0;
  }

  e_time = clock();
  *sock_clock += (e_time - s_time);
  //-------------------------------------------------

  int flag = 0;
  //xor
  for(i=0;i< BUF_SIZE;i++){
    if((recv_buf[i] ^ beta[i])!=0){
      flag= 1;
      break;
    }
    else{
      flag = 0;
    }
  }

  if(flag == 0){
    printf("認証成功\n");

    for(i = 0;i < BUF_SIZE;i++){
      Nn[i] = Nn_next[i];
    }


  }
  else{
    printf("認証失敗\n");
  }

  return 0;
}
```

B.2.2　ユーザ側の動作

```
%\end{lstlisting}
%\begin{lstlisting}[caption =  sasl-(3)\_client.c]
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <openssl/sha.h>
#include <openssl/rand.h>
#include <time.h>

#define SERVER_ADDR "127.0.0.1"
#define SERVER_PORT 8080
#define BUF_SIZE 32

void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len);
```

```
void add(unsigned char result[],unsigned char x[],unsigned char y[],int len);
void show(unsigned char s[],int len);
int transfer_first(int sock,unsigned char A1[]);
int transfer_n(int sock,unsigned char An[],clock_t *sock_clock);


int main(void) {
  int sock;
  struct sockaddr_in addr;
  FILE *file;
  int i,cnt=0;
  unsigned char A1[BUF_SIZE] = {0};
  unsigned char An[BUF_SIZE] = {0};
  clock_t s_clock,e_clock,sum_clock = 0,sock_clock = 0;

  //------------------------------------------------
  /* ソケットを作成*/
  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock == -1) {
    printf("socket error\n");
    return -1;
  }

  /* 構造体を全て 0 にセット */
  memset(&addr, 0, sizeof(struct sockaddr_in));

  /* サーバーの IP アドレスとポートの情報を設定*/
  addr.sin_family = AF_INET;
  addr.sin_port = htons((unsigned short)SERVER_PORT);
  addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

  /* サーバーに接続要求送信 */
  printf("接続を開始します...\n");
  if (connect(sock, (struct sockaddr*)&addr, sizeof(struct sockaddr_in)) == -1) {
    printf("connect error\n");
    close(sock);
    return -1;
  }
  printf("接続が完了しました!\n");
  //------------------------------------------------
  printf("---初回登録---\n");
  /* 初回登録時の接続済のソケットでデータのやり取り */

  transfer_first(sock,A1);

  for(i = 0;i < BUF_SIZE;i++){
    An[i] = A1[i];
  }

  /* ソケット通信をクローズ */
  close(sock);

  printf("---N 回目認証---\n");

  while(cnt !=1000){

    //ソケットを作成
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
      printf("socket error\n");
      return -1;
    }

    //構造体を全て 0 にセット
    memset(&addr, 0, sizeof(struct sockaddr_in));

    //サーバーの IP アドレスとポートの情報を設定
    addr.sin_family = AF_INET;
    addr.sin_port = htons((unsigned short)SERVER_PORT);
    addr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

    /* サーバーに接続要求送信 */
    printf("接続を開始します...\n");
    if (connect(sock, (struct sockaddr*)&addr, sizeof(struct sockaddr_in)) == -1) {
      printf("connect error\n");
      close(sock);
      return -1;
    }
    printf("接続が完了しました!\n");

    printf("cnt:%d\n",cnt);

    //------------------------------------------------
    s_clock = clock();

    /* N 回目認証時の接続済のソケットでデータのやり取り */
    transfer_n(sock,An,&sock_clock);

    e_clock = clock();

    sum_clock += (e_clock - s_clock);

    /* ソケット通信をクローズ */
    close(sock);
    cnt++;
  }
  printf("cnt:%d, sum_clock :%f\n",cnt,(double)(sum_clock)/CLOCKS_PER_SEC);
```

```
  printf("cnt:%d, sock_clock:%f\n",cnt,(double)(sock_clock)/CLOCKS_PER_SEC);
  printf("cnt:%d, sock-sum  :%f\n",cnt,(double)(sum_clock - sock_clock)/CLOCKS_PER_SEC);
  return 0;
}


void xor(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] ^ y[i];
  }
}

void add(unsigned char result[],unsigned char x[],unsigned char y[],int len){
  int i;
  for(i=0;i< len;i++){
    result[i] = x[i] + y[i];
  }
}

void show(unsigned char s[],int len){
  int i;

  for(i=0;i< len;i++){
    printf("%02X",s[i]);
  }
  printf("\n");
}

int transfer_first(int sock,unsigned char A1[]) {
  int recv_size, send_size,i;
  unsigned char recv_buf[BUF_SIZE];
  char send_buf;

  FILE *fp;

  /* クライアントから文字列を受信 */
  recv_size = recv(sock, recv_buf, BUF_SIZE, 0);
  if (recv_size == -1) {
    printf("recv error\n");
    return 0;
  }
  if (recv_size == 0) {
    /* 受信サイズが 0 の場合は相手が接続閉じていると判断 */
    printf("connection ended\n");
    return 0;
  }

  //A1
  for(i = 0;i < BUF_SIZE;i++){
    A1[i] = recv_buf[i];
  }

  //--------------------------------------------
  /* 接続終了を表す 0 を送信 */
  send_buf = 0;
  send_size = send(sock, &send_buf, 1, 0);
  if (send_size == -1) {
    printf("send error\n");
    return 0;
  }

  printf("接続を終了しました\n");
  return 0;

}

int transfer_n(int sock,unsigned char An[],clock_t *sock_clock) {
  int recv_size, send_size,i;
  unsigned char recv_buf[BUF_SIZE]={0};
  unsigned char send_buf[BUF_SIZE]={0};
  unsigned char An_next[BUF_SIZE] ={0};
  unsigned char alpha[BUF_SIZE] ={0};
  unsigned char beta[BUF_SIZE] ={0};
  FILE *fp;
  clock_t s_time,e_time;
  //----------receve alpha----------
  // server から文字列を受信
  s_time = clock();

  recv_size = recv(sock, recv_buf, BUF_SIZE, 0);
  if (recv_size == -1) {
    printf("recv error\n");
    return 0;
  }
  if (recv_size == 0) {
    // 受信サイズが 0 の場合は相手が接続閉じていると判断
    printf("connection ended\n");
    return 0;
  }

  e_time = clock();
  *sock_clock += (e_time - s_time);

  //alpha
  for(i = 0;i < BUF_SIZE;i++){
    alpha[i] = recv_buf[i];
```

```
  }

  //----------make beta----------
  xor(An_next,alpha,An,BUF_SIZE);
  add(beta,An_next,An,BUF_SIZE);


  //print
  /*
  printf("An      = ");
  show(An,BUF_SIZE);
  printf("An+1    = ");
  show(An_next,BUF_SIZE);
  printf("alpha   = ");
  show(alpha,BUF_SIZE);
  printf("beta?   = ");
  show(beta,BUF_SIZE);
  */


  //----------send beta---------
  s_time = clock();

  for(i = 0;i < BUF_SIZE;i++){
    send_buf[i] = beta[i];
  }
  send_size = send(sock, send_buf, BUF_SIZE, 0);
  if (send_size == -1) {
    printf("send error\n");
    return -1;
  }

  e_time = clock();
  *sock_clock += (e_time - s_time);

  for(i = 0;i < BUF_SIZE;i++){
    An[i] = An_next[i];
  }

  printf("接続を終了しました\n");
  return 0;

}
```