目 次

第1章　序論

本章では、研究の背景、研究の目的・目標、本論文の構成について述べる。

1.1　まえがき

私たちの身の回りには様々な IoT(Internet of Things) 機器がある。IoT 機器は、複数の機器をローカルネットワーク、またはインターネットで接続し、情報や制御のやり取りを行っている [1]。このような機能を活かし、遠隔地で IoT 機器を稼働させたり、人が行っていた作業を IoT 機器が行い業務の生産効率を向上させるなど、IoT 機器によって私たちの生活は豊かになっている。一方で、機器間のデータの盗聴や改ざん、特定の IoT 機器へのなりすましという悪意のある攻撃を受ける恐れがある。

情報処理推進機構 IPA2012 年のレポートにより、自動車のタイヤ空気圧監視システム (TPMS:(Tire Pressure Monitoring System)) への攻撃事例がアメリカで報告されている [2]。TPMS とは、空気圧が低いタイヤで高速走行をすることによるタイヤ破裂事故を防ぐためにタイヤの空気圧を常時監視するシステムであり、測定したタイヤの空気圧を無線通信により収集し、タイヤの空気圧に応じて空気圧警告メッセージを送信する。この TPMS に対して、通信メッセージが暗号化されていないため、盗聴・解析が容易である、空気圧報告メッセージになりすまし、警告灯を点灯することが可能である、という報告が発表されている。

このような攻撃への対策として IoT 機器に、機器間の認証機能、通信データの暗号化機能を搭載することが挙げられる。しかしながら、IoT 機器にはコスト要因で処理能力の低い機器が多数存在するため、処理負荷の大きな認証方式や暗号方式は搭載することが困難である。そこで本研究では、IoT システムのセキュリティ強化のために、処理能力の低い IoT 機器間の相互認証機能およびデータ通信を暗号化する機能を有するセキュアな組込みシステムを開発することを目的とする。具体的には、高知工科大学の清水明宏教授が処理能力の低い装置へのセキュリティ機能実現のために開発された SAS-L2 という認証方式を IoT 機器に実装することで、軽量かつ強固なワンタイムパスワード認証方式を実現する。さらに、SAS-L2 認証メカニズムに基づいて、IoT 機器間のデータ通信を暗号化する方法を提案およ

び実装した。

SAS-L2 を導入したセキュアな IoT システムは、UML を用いて設計を行い、V 字開発モデルに従って、浅野と内山田のチームで開発を行った。

1.2　論文の構成

本論文の構成について述べる。本論文では、第 1 章に研究背景を述べる。第 2 章に本研究で用いた要素技術、アルゴリズムについて述べる。第 3 章に SAS-L2 を用いた IoT 機器間のデータ通信の暗号化について述べる。第 4 章に本研究で開発するシステムについて述べる。第 5 章にシステムのサーバの開発方法について述べる。第 6 章にシステムの実装結果と評価を述べる。第 7 章に実装結果に対する考察を述べる。第 8 章に本論文のまとめを述べる。

第2章　準備

本章では、セキュアな IoT 組込みシステム開発に用いた要素技術および用語について述べる。

2.1　TCP 通信プロトコル

TCP(Transmission Control Protocol) とは、コネクション型で信頼性のある通信プロトコルである [3]。TCP におけるデータの送信手順を図 2.1 に示す。

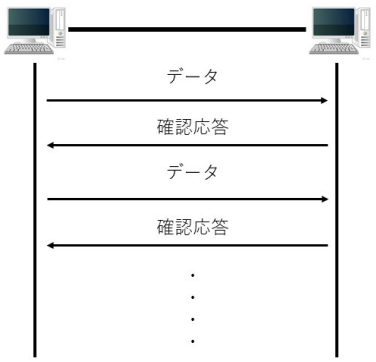


図 2.1　TCP におけるデータの送信手順

TCP では、送信したデータが受信側に届いたとき、受信側は送信側に確認応答を送信してデータが到達したことを通知する。送信側が一定時間内に確認応答を受信できなかった場合、もう一度同じデータを送信する再送機能がある。この確認応答により、データ到達の信頼性を実現している。

TCP の通信手順を図 2.2 に示す

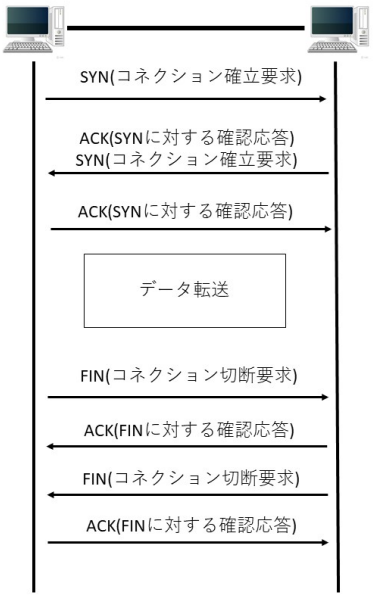


図 2.2　通信手順

TCP では、データ通信に先立ち、通信相手との間で通信を始める準備を行ってから通信を行う。まず、コネクション確立要求のパケットである SYN パケットを通信相手に送信して確認応答を待つ。通信相手から確認応答が送信された場合、データ通信は可能になるが、送信されなければデータ通信を行うことはできない。通信が終了すると、コネクション切断要求のパケットである FIN パケットを通信相手に送信し、コネクションの切断を行う。

2.2　暗号化方式

本節では、従来の暗号化方式として、AES、RSA、バーナム暗号の概要を述べる。

2.2.1　AES(Advanced Encryption Standard)

AES(Advanced Encryption Standard) は、無線 LAN などの通信データの暗号化に用いられる共通鍵暗号アルゴリズムである [4]。共通鍵暗号とは、データの送信者と受信者が同じ暗号鍵を用いて暗号化と復号を行う暗号方式である。また、AES はブロック暗号であり、ブロック長は 128bit(16byte) である。ブロック暗号とは、平文のデータを特定の長さに分割し、分割

した平文に対して暗号化を行うアルゴリズムである。[5]。AES では、128bit、192bit、256bit の中から鍵長を選択することができる。あらかじめ鍵長を選択して鍵を生成し、4 つの処理をおこない、暗号化を行う。AES の暗号化手順を図 2.3 に示す [6]。

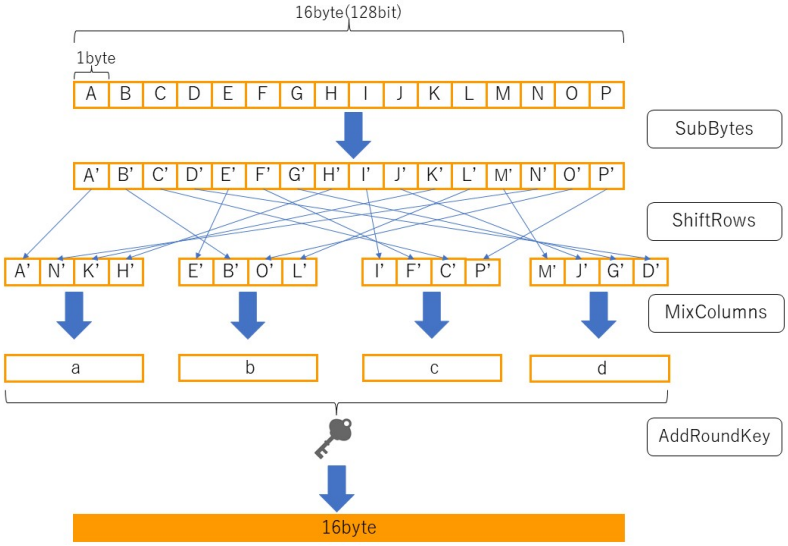


図 2.3　AES の暗号化手順

暗号化の 4 つの処理は以下の手順で行われている。

1. 16byte ごとに区切ったデータに対し、1byte 単位で置換する。(SubBytes)
2. 一定規則で 1byte 単位でデータの順序を入れ替える。(ShiftRows)
3. ビット演算による 4byte 単位の行列変換を行う。(MixColumns)
4. 鍵との XOR をとる。(AddRoundKey)

この一連の処理を 1 ラウンドとし、鍵長に応じた回数のラウンドを行う。鍵長が 128bit のときラウンド数は 10 回、192bit のときラウンド数は 12 回、256bit のときラウンド数は 14 回である。

復号処理は、上記の処理の逆変換を逆順で行う。

AES の特徴として、4 種類の変換を 1 ラウンドとして、その処理を複数回繰り返すことにより、暗号強度を高めている点が挙げられる。

2.2.2　RSA

RSA とは、桁数が大きい素数の素因数分解が困難であることを安全性の根拠としている公開鍵暗号方式である [7]。公開鍵暗号とは暗号鍵と復号鍵が別々の暗号方式のことで、暗号鍵は第三者に見られても問題ないということから公開鍵と呼ばれ、復号鍵は第三者に知られてはいけないことから秘密鍵という。RSA は、3 名の発明者 (R.L.Rivest、A.Shamir、L.Adleman) の名前に由来している。

RSA の暗号化手順の概要を図 2.4 に示す。

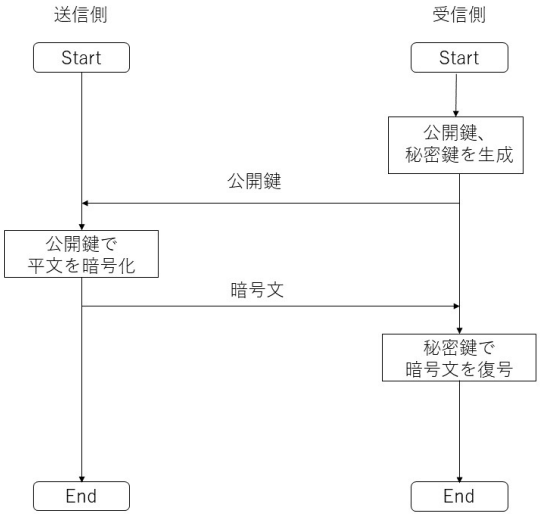


図 2.4　RSA の暗号化手順

RSA の暗号化の流れは以下のようになる。

1. 受信者が秘密鍵および公開鍵を生成し、公開鍵を送信者に送信する。
2. 送信者は、公開鍵を用いて平文を暗号化して暗号文を生成し、受信者に送信する。
3. 受信者は、暗号文を秘密鍵を用いて平文に復号する。

各処理について詳細を述べる。まず、鍵の生成手順を以下に示す。

1. 異なる任意の素数 p、q を任意にとる。
2. $n = pq$ とする。
3. $(p-1)(q-1)$ と互いに素な自然数 e を任意にとる。

4. ed を $(p-1)(q-1)$ で割った余りが 1 となる自然数 d を任意にとる。

5. n と e を公開鍵、p と q と d を秘密鍵とする。

次に、暗号化の手順を以下に示す。

1. 送信メッセージを x とする。ただし、$x<n$ とする。

2. x を e 乗し、これを n で割った余りを y とする。

3. y を暗号文とする。

最後に、復号の手順を以下に示す。

1. y を d 乗する。

2. 1 で得た値を n で割った余りが平文 x となる。

このように復号には d が必要であるが秘密鍵であるため、第三者が d を得るには p と q の値を特定しなければならない。p と q を特定するためには、公開鍵の n を素因数分解を行う必要があるが膨大な時間がかかり、解読が困難とされているため、安全性が確保されている。

2.2.3　バーナム暗号

バーナム暗号とは、バーナムによって考案され、シャノンによって理論的に解読不可能であることが証明された共通鍵暗号である [8]。バーナム暗号では、平文と、平文と同じサイズのランダムな秘密鍵の排他的論理和をとることで、平文に対して暗号化を行う。復号の際は、暗号文と秘密鍵の排他的論理和をとる。

バーナム暗号の長所として、演算処理の負荷が小さいことが挙げられる。データの送信者・受信者の両者が排他的論理和を一度行うだけで暗号化・復号化を行うことができる。さらに、安全性が高いことが長所として挙げられる。例え秘密鍵を総当たりして秘密鍵の候補を推測でき、暗号文との排他的論理和をとったとしても、それが正しい平文なのかを判断することができないという点から、バーナム暗号は解読不可能であるため、安全性が高い。

一方で、バーナム暗号の短所として、秘密鍵が使い捨てであるため、暗号化を行う度に使用可能な秘密鍵が減っていくということが挙げられる。また、暗号化の度に新しい鍵を共有する必要があるため、鍵配送のコストが大きくなるということ、秘密鍵を安全に配送する方法が困難であるという点が挙げられる。

2.3 SAS-L2(Simple And Secure password authentication protocol, Light processing version, type2)

SAS-L2(Simple And Secure password authentication protocol, Light processing version, type2) とは、高知工科大学の清水明宏教授が提案した、ワンタイムパスワード認証方式である。情報通信システムにおける、サーバ (認証側) とユーザ (被認証側) の認証において、ユーザの演算負荷を少なくすることで、ユーザの処理負荷が特に小さくなっていることが特徴である。

本節では、SAS-L2 のアルゴリズム、および SAS-L2 の実装に使用した要素技術について述べる。

2.3.1 SAS-L2 認証アルゴリズム

SAS-L2 は、あらかじめサーバ (認証者) が配下にあるユーザ (被認証者) を初期登録し、その後、初回の認証処理を行う。サーバはユーザ識別子を秘密に所持して、ユーザの情報を管理している。以下に本論文においての記法を示す。

- ユーザ識別子を S とする。
- n 回目の認証情報を生成する際に使用する乱数を N_n とする。
- s に対して一方向性関数を適用した値を $H(s)$ とする。
- n 回目の認証情報を A_n とする。
- n 回目の秘匿情報 (マスク値) を M_n とする。

一方向性関数とは、関数値は容易に計算できるが、逆関数の計算が非常に困難である関数のことである。

図 2.5 に初期登録のフローチャートを示す。

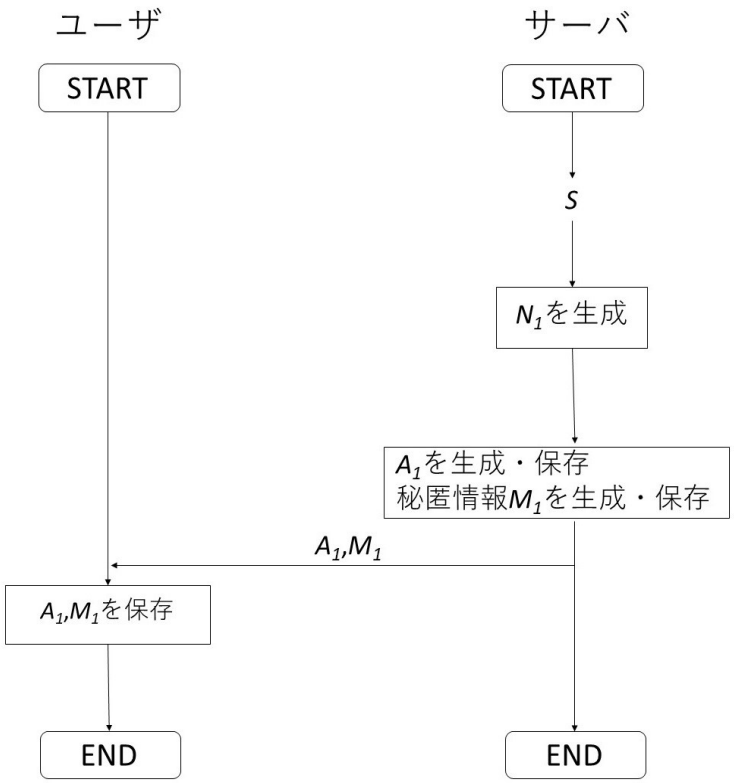


図 2.5　SAS-L2 初期登録のフローチャート

初期登録の流れは以下のようになる。

1. サーバは乱数 N_1 を生成する。
2. サーバは初回認証情報 $A_1 = H(S\,XOR N_1)$ を生成し、保存する。
3. サーバは初回秘匿情報 M_1 を生成し、保存する。
4. サーバは A_1 および M_1 を安全な手法でユーザに送信する。
5. ユーザは受信した A_1 および M_1 を保存する。

上記 4 における安全な手法とは、インターネットなどの安全ではないネットワークを経由しないルートでデータを受け渡すことを指す。

図 2.6 に認証のフローチャートを示す。

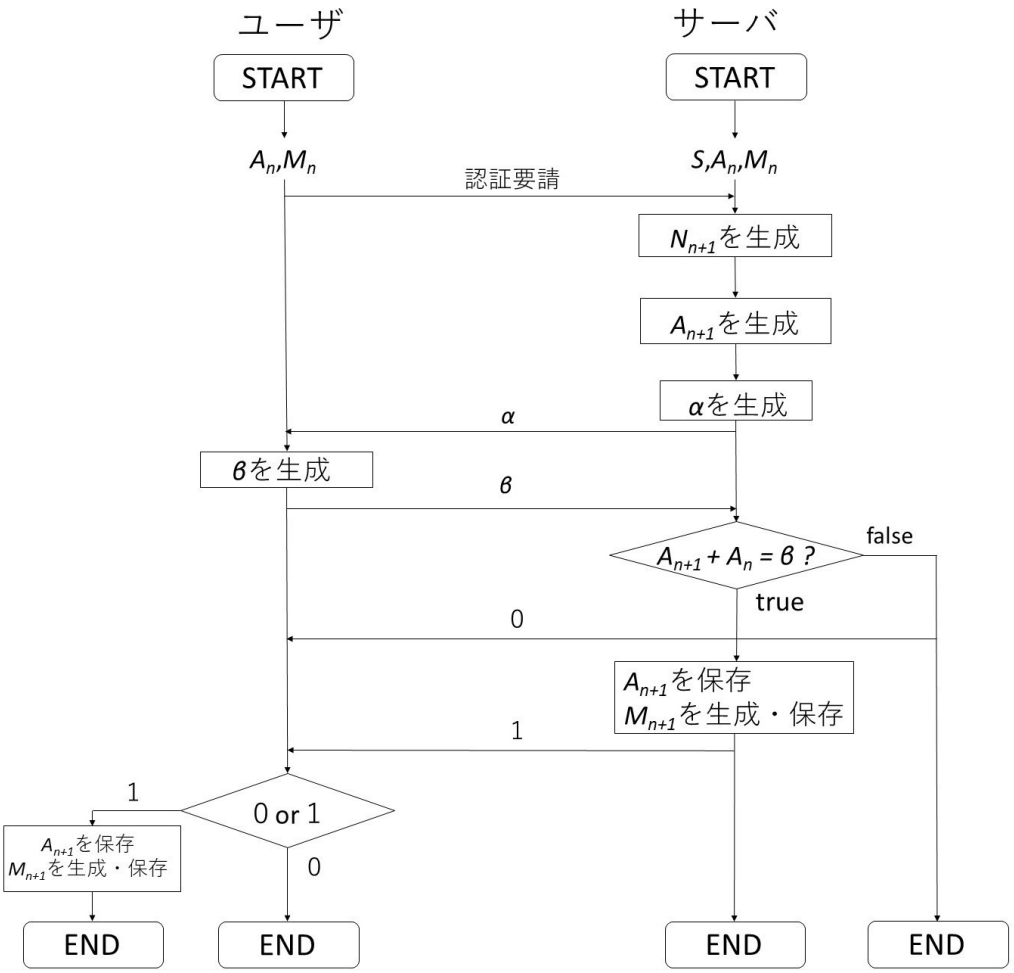


図 2.6　SAS-L2 認証のフローチャート

ユーザは認証情報 A_n を保持し、サーバはユーザ識別子 S と認証情報 A_n を保持しているものとする。認証の流れは以下のようになる。

1. サーバはユーザから認証要請等を受信すると、乱数 N_{n+1} を生成する。
2. サーバは次回認証情報 $A_{n+1} = H(S\ XOR\ N_{n+1})$ を生成する。
3. サーバは送信情報 $\alpha\ = A_{n+1}\ XOR\ A_n\ XOR\ M_n$ を生成し、ユーザに送信する。
4. ユーザは送信情報 $\beta\ = (\alpha\ XOR\ A_n\ XOR\ M_n) + A_n$ を生成し、サーバに送信する。
5. サーバは $A_{n+1} + A_n$ の演算を行い、$A_{n+1} + A_n$ と β の値が等しい場合は認証成立となり、6～8 の処理が行われる。等しくない場合は認証不成立となり、9 の処理が行われる。
6. サーバは次回認証情報 A_{n+1} を認証情報として保存する。

7. サーバは $M_{n+1} = A_n + M_n$ を生成して、秘匿情報として保存する。
8. サーバは認証フラグ (1) をユーザに送信し、10 の処理を行う。
9. サーバは認証フラグ (0) をユーザに送信する。
10. ユーザは認証フラグを受信し、認証フラグが 1 の場合は 11〜12 の処理を行い、0 の場合は認証処理を終了する。
11. ユーザは次回認証情報 A_{n+1} を認証情報として保存する。
12. サーバは $M_{n+1} = A_n + M_n$ を生成して、秘匿情報として保存する。

SAS-L2 では、サーバで生成した次回認証情報と、あらかじめユーザとサーバが所持していた今回認証情報と秘匿情報の排他的論理和演算の結果を配送することで、鍵をネットワークに直接流すことなく認証および鍵の更新を行っている。また、ユーザの演算が加算演算と排他的論理和演算のみとなっているため、ユーザの処理負荷が非常に小さいことが特徴である。

2.3.2　OpenSSL(Open Secure Sockets Layer)

OpenSSL(Open Secure Sockets Layer) とは、インターネット上で標準的に利用されている暗号通信プロトコルである SSL および TLS の機能を実装した、オープンソースのライブラリのことである [9]。ライブラリは基本的な暗号化関数と、様々なユーティリティ関数があり、本研究では、一方向性関数の実装のために MD5、乱数生成のために RAND_Bytes() を使用している。

2.3.3　MD5(Message Digest algorithm 5)

MD5(Message Digest algorithm 5) とは、1991 年にロナルド・リベストによって開発された、128bit のハッシュ値を生成する一方向性関数の一つである [10]。ハッシュ値へ変換されると、元に戻すことが非常に困難である。しかし MD5 には、ハッシュの衝突耐性に脆弱性があり、変換前のデータと同じハッシュ値を持つ非ユニークなデータが生成される、また、生成されたハッシュ値から論理的に変換前のデータを求められるという問題があるため、現在、暗号的一方向性関数を使用する際は、SHA-224 や SHA-256 が推奨されている。

2.4　システム設計に関する用語

本節では、システム設計に関する用語について述べる。

2.4.1　V 字開発モデル

V 字開発モデルとは、システム開発が開始してから終了するまでの流れにおける開発工程とテスト工程を表したモデルのことである [11]。図 2.7 に V 字開発モデルの概念図を示す。

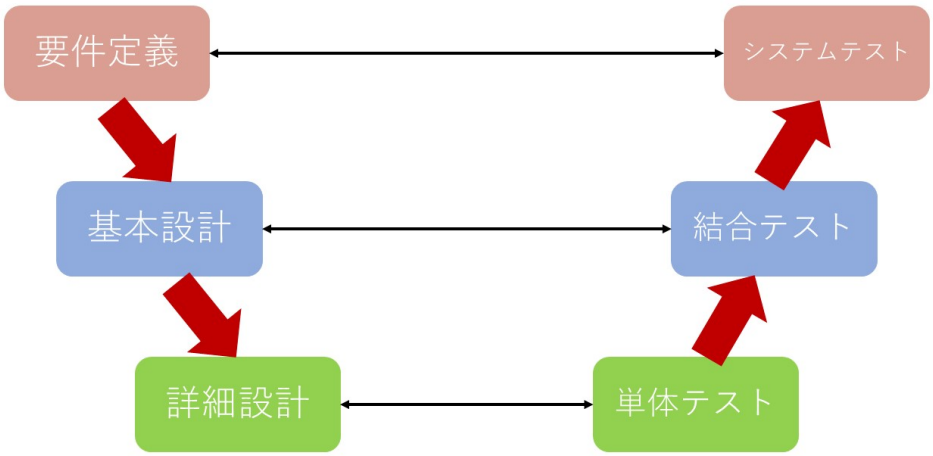


図 2.7　V 字開発モデル

V 字の左側が開発工程になり、右側がテスト工程になる。V 字の左上から左下へ開発工程が進み、終了すると、右下から右上へとテスト工程が進む。V 字開発では開発工程とテスト工程を詳細度合いに応じて対になっているため、成果物に対するテストのレベルや範囲、内容を適切に認識・設定することができる。

2.4.2　UML(Unified Modeling Language)

UML(Unified Modeling Language) とは、オブジェクト指向分析、設計においてシステムをモデル化する際の記法 (図形) を規定した言語である [11]。UML は、プログラムを書く前に図で自分の考えを整理することができる、チーム開発において図でコミュニケーションを取ることができる、ユーザや顧客との仕様の検討を円滑に行うことができる、という理由か

ら、ソフトウェア開発の際に用いられている。

2.4.3　ユースケース図

システムの仕様機能 (ユースケース) と外部環境 (アクター) との関係を表す [12]。図を読み取るための専門的な知識は必要なく、一目でシステムの機能や、システム内部と外部の境界を理解することができるため、ユーザや顧客がシステムの仕様を容易に理解することができる。

2.4.4　クラス図

モデルの静的な構造を表す図で、問題領域やシステムの構造を表現する [12]。パッケージ単位での表現、全体での表現、機能単位での表現というように、様々な視点で作成することができる。

2.4.5　シーケンス図

オブジェクトの相互作用を表す相互作用図の 1 つで、オブジェクト間のやり取りを時系列に沿って表現する [12]。

2.4.6　アクティビティ図

業務や処理の流れを表すために、関連する複数の業務手順や処理ステップを順序だてて配置したものである [12]。ワークフローからユースケースの処理フロー、メソッド内部のアルゴリズムなど、手順のあるものを表現することができる。

第3章　IoT機器間のデータ通信の暗号化

本章では、SAS-L2 認証メカニズムに基づいた通信データの暗号化の方法について述べる。

3.1　SAS-L2 を用いたデータ通信の暗号化

第 2 章で述べた従来の暗号方式の AES および RSA は演算による処理負荷が大きいため、処理能力が低い IoT 機器への搭載が難しい。バーナム暗号は演算の処理負荷は小さいが、共通鍵を直接ネットワークに流して共有しなくてはいけないという問題点がある。そこで本章では、SAS-L2 認証アルゴリズムを拡張して、処理能力の低い IoT 機器間のデータ通信を暗号化するアルゴリズムを提案する。なお、この暗号化アルゴリズムは、SAS-L2 による認証が成立した後に行うものとする。暗号化のフローチャートを図 3.1 に示す。

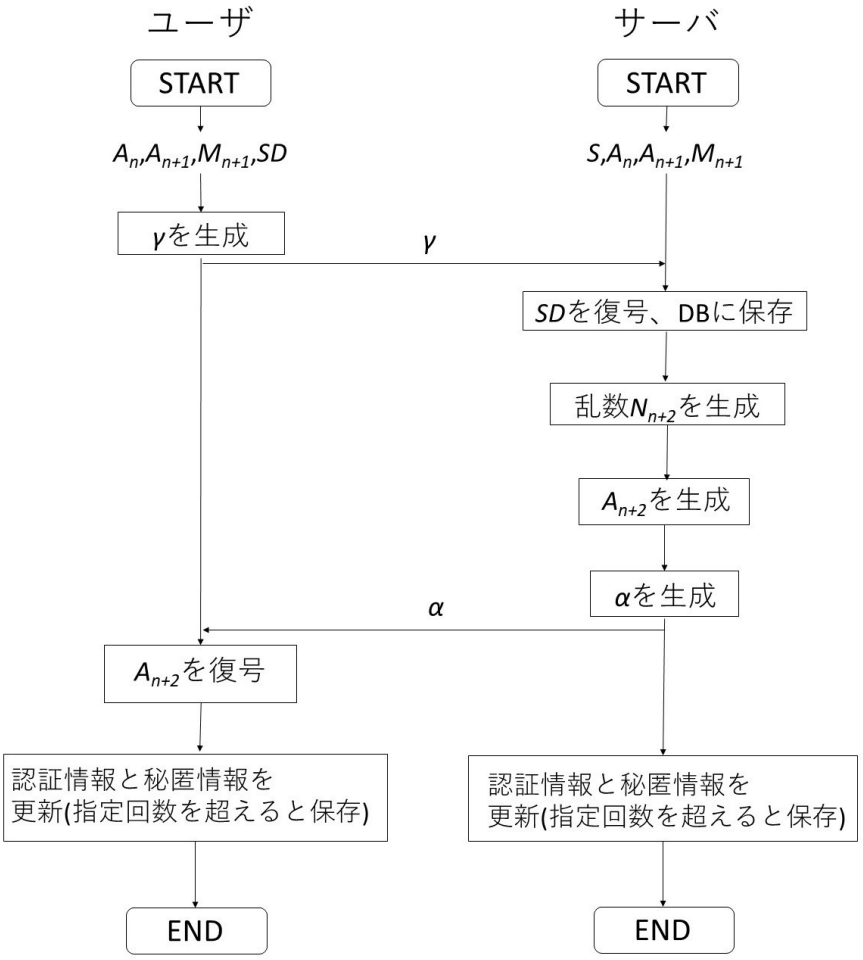


図 3.1　SAS-L2 に基づいたセンシングデータの暗号化アルゴリズムのフローチャート

本来、SAS-L2 の認証処理が成功すると、ユーザとサーバでは認証情報 A_n を次回認証情報 A_{n+1} に更新するが、暗号化を行うにあたって、今回認証情報 A_n および次回認証情報 A_{n+1} をそれぞれ保存する。さらに、ユーザとサーバは次回秘匿情報 M_{n+1} を保持している。また、サーバはユーザ識別子を保存しており、ユーザでは暗号化して送信したいデータ SD を取得しているものとする。以下に暗号化アルゴリズムの流れを示す。

1. ユーザは送信情報 $\gamma \leftarrow SD\ XOR\ A_{n+1}\ XOR\ A_n$ を生成し、サーバに送信する。
2. サーバは演算 $SD \leftarrow \gamma\ XOR\ A_{n+1}\ XOR\ A_n$ を行い、データを復号する。
3. サーバはデータをデータベースに保存する。
4. サーバは乱数 N_{n+2} を生成する。

5. サーバは次回認証情報 $A_{n+2} \leftarrow H(S\ XOR\ N_{n+2})$ を生成する。

6. サーバは送信情報 $\alpha \leftarrow A_{n+2}\ XOR\ A_{n+1}\ XOR\ M_{n+1}$ を生成し、ユーザに送信する。

7. ユーザは演算 $A_{n+2} \leftarrow \alpha\ XOR\ A_{n+1}\ XOR\ M_{n+1}$ を行い、A_{n+2} を復号する。

8. サーバおよびユーザは認証情報 $M_{n+2} \leftarrow A_{n+1} + M_{n+1}$ を生成する。

9. サーバおよびユーザは、今回認証情報の更新 $A_n \leftarrow A_{n+1}$、次回認証情報の更新 $A_{n+1} \leftarrow A_{n+2}$、秘匿情報の更新 $M_{n+1} \leftarrow M_{n+2}$ を行う。

10. 1 から 9 の処理を指定回数行う。

11. 認証情報を $A_n \leftarrow A_{n+1}$、秘匿情報を $M_n \leftarrow M_{n+1}$ として保存する。

11 のように、暗号化通信が終了して、認証情報および秘匿情報を保存することで、暗号化通信の際に何らかの問題が発生して通信を切断しても、認証情報に違いが生まれず、再度認証処理を行うことができる。

この暗号化アルゴリズムは、ユーザ側の演算が排他的論理和と加算のみとなっており、SAS-L2 と同様にユーザ側の演算処理の負荷が非常に小さいという特徴がある。また、SAS-L2 認証アルゴリズムと同様の方法で共通鍵を更新しているため、共通鍵を直接ネットワークに流さずに更新できるということも特徴として挙げられる。本章で提案した暗号化アルゴリズムはこれらの 2 つの特徴から、AES および RSA、バーナム暗号の問題点を解決し、処理能力の低い IoT 機器搭載可能な暗号化アルゴリズムを実現している。

第4章　SASを用いたセキュアなIoTシステム開発

本章では、開発方針、SAS を用いたセキュアな IoT システムの仕様、基本設計、システムテストおよび結合テストの項目について述べる。

4.1　開発方針

本研究は浅野と内山田の 2 人のグループで行う。V 字開発モデルに従い、SAS を用いたセキュアな組込システムを開発する。要件定義と基本設計を共同で行う。その後、浅野がエッジ側、内山田がサーバ側を担当し、それぞれ詳細設計、実装を行う。実装したものに対して、それぞれで単体テストを行った後、結合テストとシステムテストを行い、システムの評価を行う。

4.2　システムの要件定義

本研究で開発するシステムについて説明する。図 4.1 にシステムの構成図を示す。図 4.1 の赤枠で囲まれている部分がエッジ側で、青枠で囲まれている部分がサーバ側というように区別している。

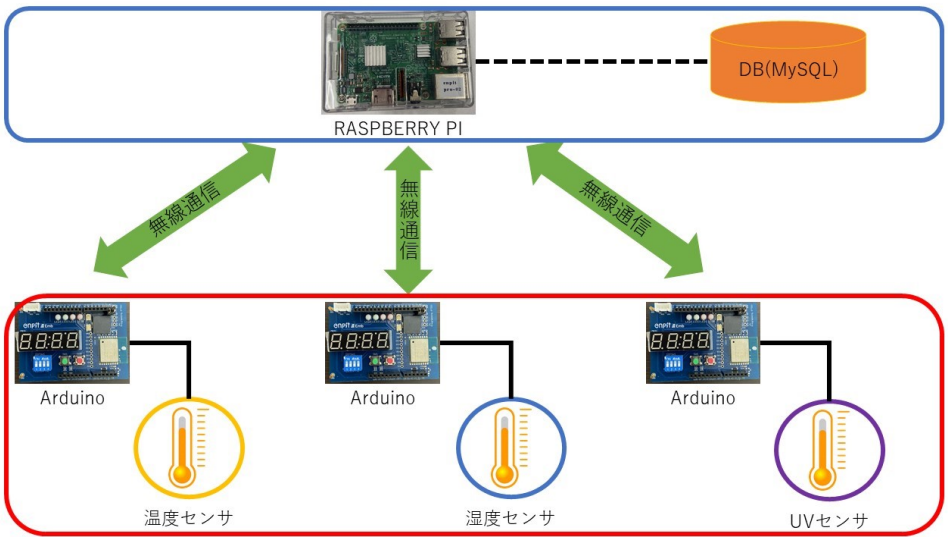


図 4.1　システムの構成図

本研究では、ユーザのデバイスに Arduino、サーバのデバイスに Raspberry Pi を使用する。Arduino を 3 台利用し、それぞれに温度センサ、湿度センサ、UV センサが接続されている。また、データベースは MySQL を利用している。

本システムにおけるセンシングデバイスの認証とセンシングデータの暗号化通信を行う処理の流れを以下に示す。

1. サーバを起動し、ユーザの認証要求待ち状態にする。
2. ユーザは決められた時刻になると、サーバとコネクションして認証要求を送信する。
3. サーバが認証要求を受信すると、認証処理を行う。
4. 認証が成立すると、ユーザはセンサからセンシングデータを取得する。
5. ユーザはセンシングデータとセンシングデータを取得した時間を暗号化し、サーバに送信する。
6. サーバは受信したデータを受信すると、データを復号し、センシングデータと取得時間をデータベースに保存する。
7. ユーザとサーバのコネクションを切断し、サーバは認証要求の受信待機状態になり、ユーザは決められた時刻になるまで待機する。

8. 2 に戻る。

本システムでは、処理 3 における認証方式に SAS-L2 を利用し、処理 5 におけるセンシングデータの暗号化に 3 章で提案したデータ通信の暗号化方式を利用している。また、ユーザとサーバ間の通信プロトコルは TCP を利用している。

以下に本システムの機能要件を述べる。なお、機能要件とは、開発するシステムにおいて、必ず実装するべき機能のことである。

- 設定した時刻にユーザからサーバにコネクションをして認証要求を送信する。
- 認証要求を受信後、認証処理を行う。
- 認証が成立した場合、ユーザはセンシングデータおよび取得時間を暗号化してサーバに送信する。
- 認証が失敗した場合、ユーザとサーバのコネクションを切断する。ユーザは設定した時刻まで待機し、サーバはユーザの認証要求の受信待ち状態になる。
- サーバは受信したデータを復号し、データベースに保存する。
- 暗号化通信が終了したら、ユーザとサーバのコネクションを切断する。ユーザは設定した時刻まで待機し、サーバはユーザの認証要求の受信待ち状態になる。

以下に本システムの非機能要件を述べる。なお、非機能要件とは、開発するシステムの機能のうち、機能要件以外の機能のことであり、システムの性能面やセキュリティ面を強化するための機能である。

- ユーザは、起動後に時刻同期を行う。
- 複数のユーザが同時刻に認証要求を送信した場合、1 つのユーザが通信を終了した後に、他のユーザは通信を開始する。
- 認証および暗号化通信の際、5 秒以上データを受信できなかった場合、ユーザとサーバのコネクションを切断する。ユーザは設定した時刻まで待機し、サーバはユーザの認証要求待ち状態になる。
- 認証結果をサーバからユーザに送信し、サーバとユーザそれぞれのモニターで認証結果を表示する。

- サーバは、センシングデータをデータベースに保存する際に、ユーザごとのテーブルに分けて保存する。
- ユーザは、取得したセンサ値および取得時間をモニタに表示する。
- 1 度の通信で、センシングデータと取得時間を 10 回送信する。
- ユーザは、サーバとの通信中にエラーが発生した場合、LED を点滅させる。
- 3 つのユーザとの通信を 10 秒以内に終わらせる。

センシングデバイスの識別方法を図 4.2 に示す。それぞれのユーザは 1〜3 の番号を持ち、その番号をサーバと共有している。認証要請を送信する際に、自身の番号を送信することで、サーバはどのユーザと通信を行っているかを識別する。

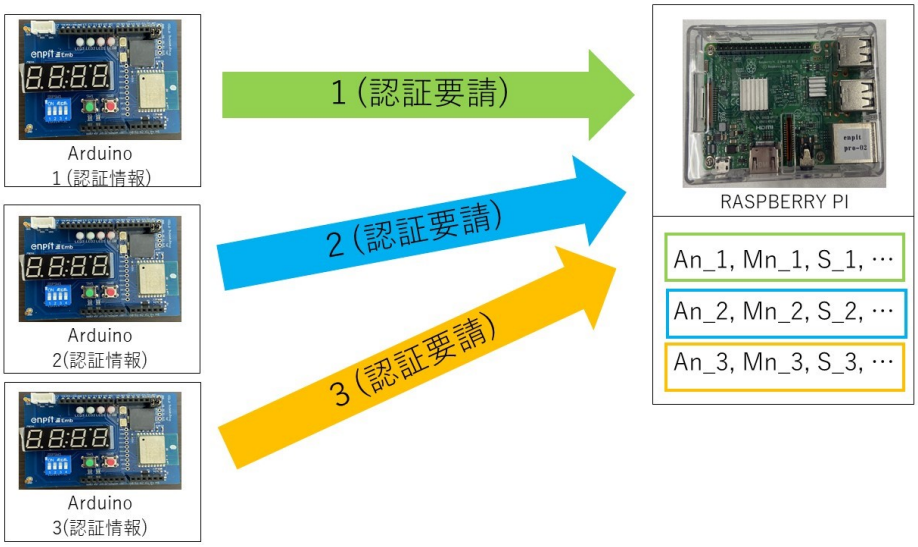


図 4.2　センシングデバイスの識別方法

センシングデータと取得情報を格納する際のデータフォーマットを図 4.3 に示す。センシングデータと取得時間は、unsigned char 型の 16byte の配列に格納される。先頭から 1〜4byte に float 型のセンシングデータが格納されおり、5〜11byte に int 型の日付がそれぞれ格納されている。12byte〜16byte は、オプションとして用意され、本システムでは乱数を格納している。

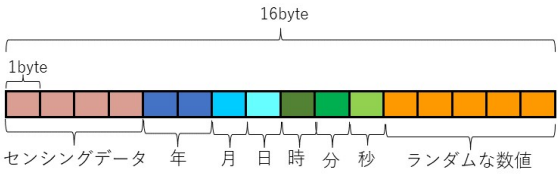


図 4.3　センシングデータと取得情報を格納する際のデータフォーマット

4.3　基本設計

前節で述べた機能要件および非機能要件を実装するため、UML を用いて基本設計を行った。ここでは、ユースケース図、クラス図、シーケンス図を用いる。図 4.4 にユースケース図を示す。

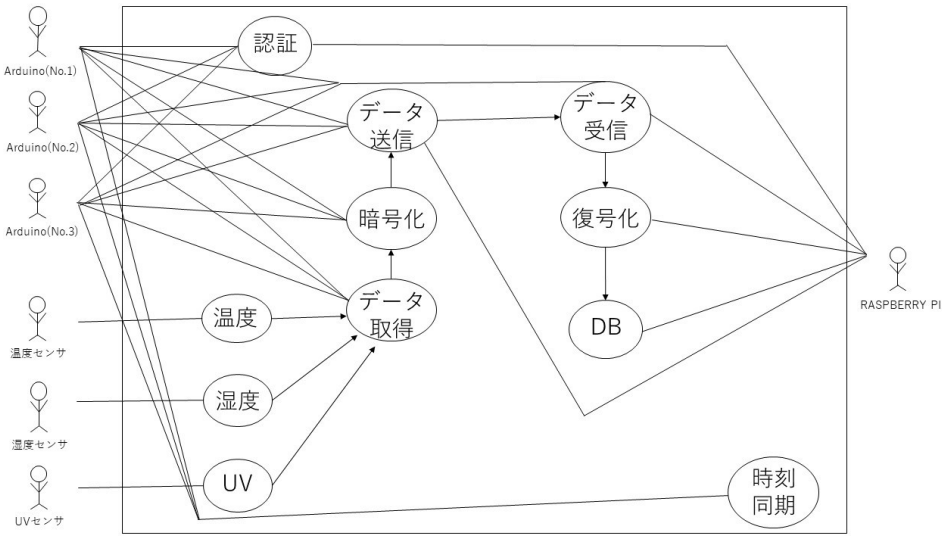


図 4.4　ユースケース図

外部環境 (アクター) として、Arduino、各センサ、Raspberry Pi があり、それぞれのアクターと使用機能がどのような関係にあるのかを示している。

図 4.5 にクラス図を示す。クラス図では、デバイスごとにクラスを分けており、各デバイスが保持しているデータと、各デバイスが行う処理を示している。

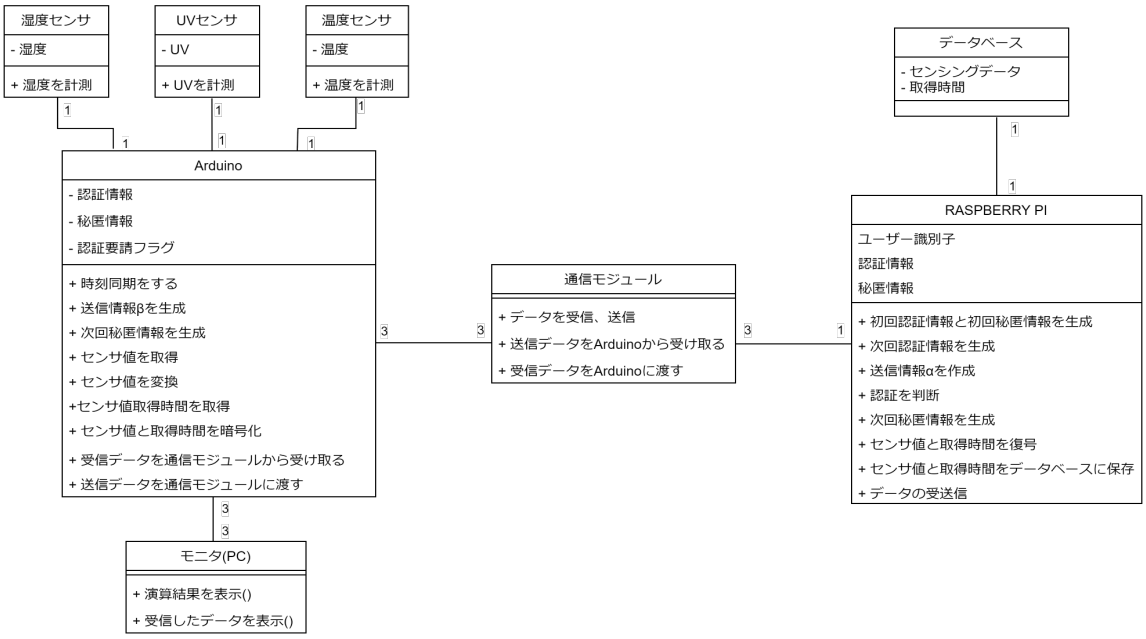


図 4.5　クラス図

図 4.6 にシーケンス図を示す。シーケンス図では、各デバイスを 1 つのオブジェクトとし、各オブジェクト間の相互作用を時系列に沿って示している。シーケンス図中の loop は、枠内の処理を指定回数繰り返すということを示している。

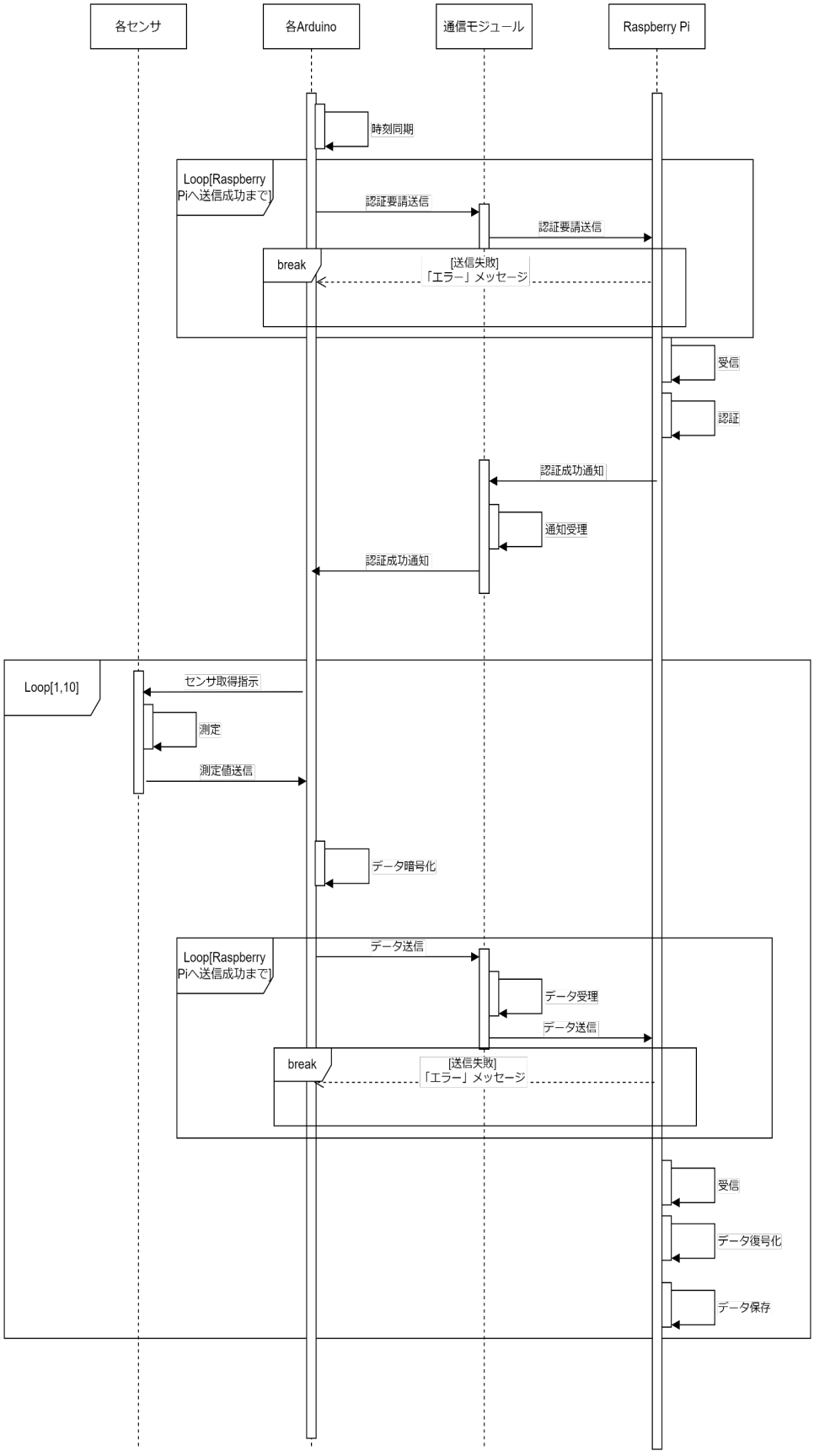

各センサ
各Arduino
通信モジュール
Raspberry Pi
時刻同期
Loop[Raspberry Piへ送信成功まで]
認証要請送信
認証要請送信
break
[送信失敗]
「エラー」メッセージ
受信
認証
認証成功通知
通知受理
認証成功通知
Loop[1,10]
センサ取得指示
測定
測定値送信
データ暗号化
Loop[Raspberry Piへ送信成功まで]
データ送信
データ受理
データ送信
break
[送信失敗]
「エラー」メッセージ
受信
データ復号化
データ保存

図 4.6　シーケンス図

第5章　SASを用いたセキュアなIoTシステムにおけるサーバの開発

本章では、SAS を用いたセキュアな IoT システムにおけるサーバの開発環境、詳細設計、実装方法、単体テストについて述べる。

5.1　開発環境

表 5.1 に本研究の開発環境を示す。

表 5.1　開発環境

使用機器	Raspberry Pi 3 Model B
OS	raspbian
使用言語	Python, C言語, SQL
周辺機器	キーボード, マウス, モニター

5.2　詳細設計

図 5.1 に本システムのサーバの機能をクラスで表わしたクラス図を示す。

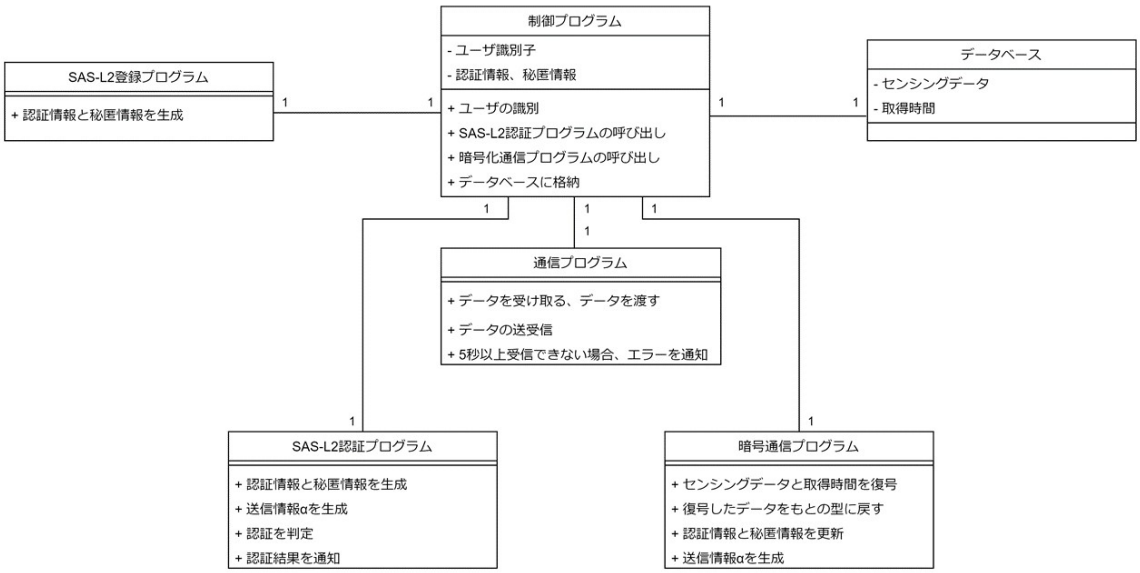


図 5.1　サーバのクラス図

サーバには、SAS-L2 認証処理を行うための初期登録機能、SAS-L2 認証処理を行う機能、3 章で提案したデータ通信暗号化方式で暗号化されたセンシングデータを復号する機能、ユーザと通信を行う機能がある。

機能ごとにプログラムを作成し、各プログラムを、制御プログラムで呼び出すことでサーバとして機能させる。ただし、SAS-L2 初期登録プログラムは、独立で実行し、生成した初回認証情報と初回秘匿情報をユーザとサーバであらかじめ共有しているものとする。

図 5.2 に認証のアクティビティ図、図 5.3 に暗号通信のアクティビティ図を示す。

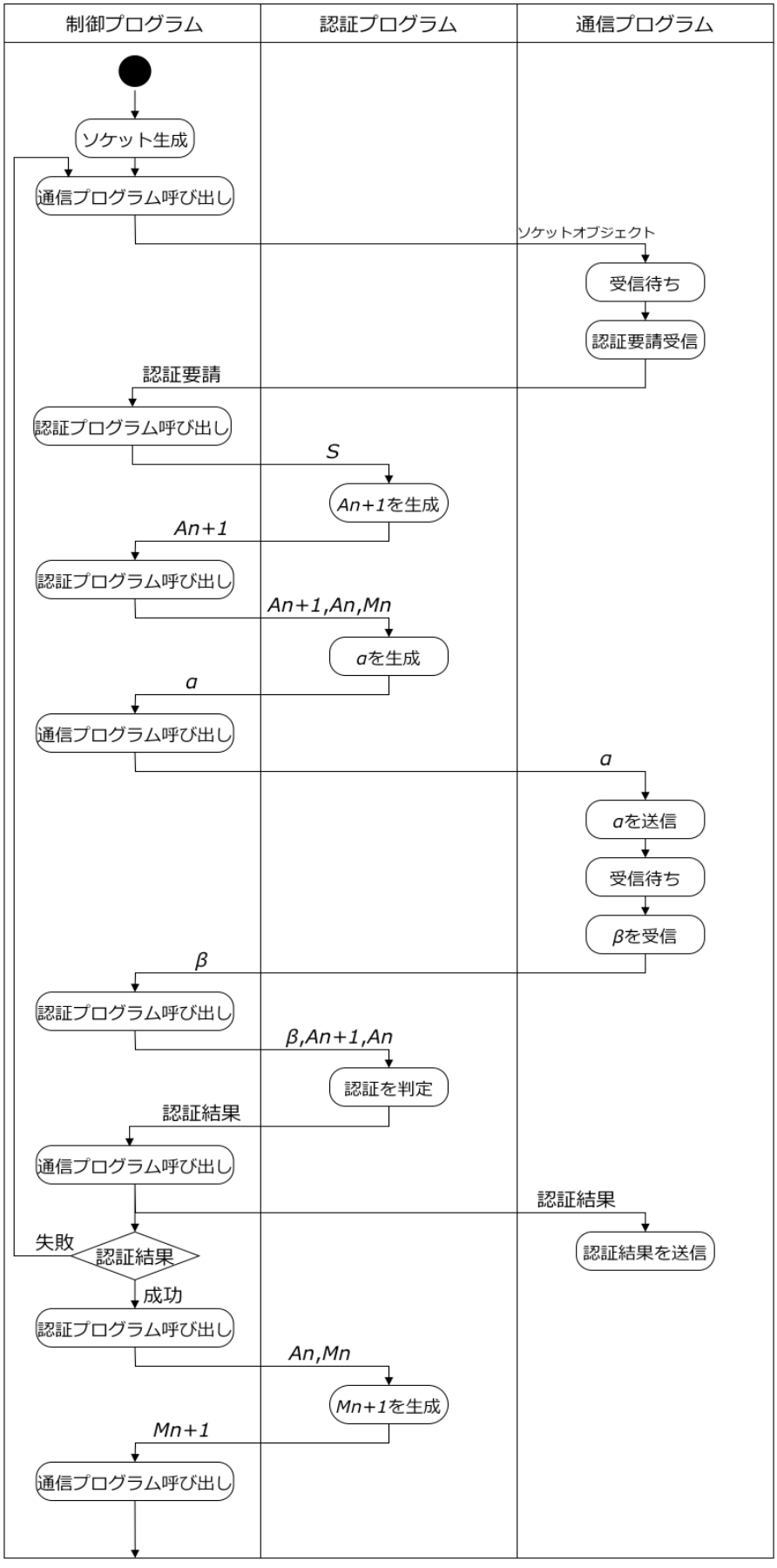


図 5.2　認証のアクティビティ図

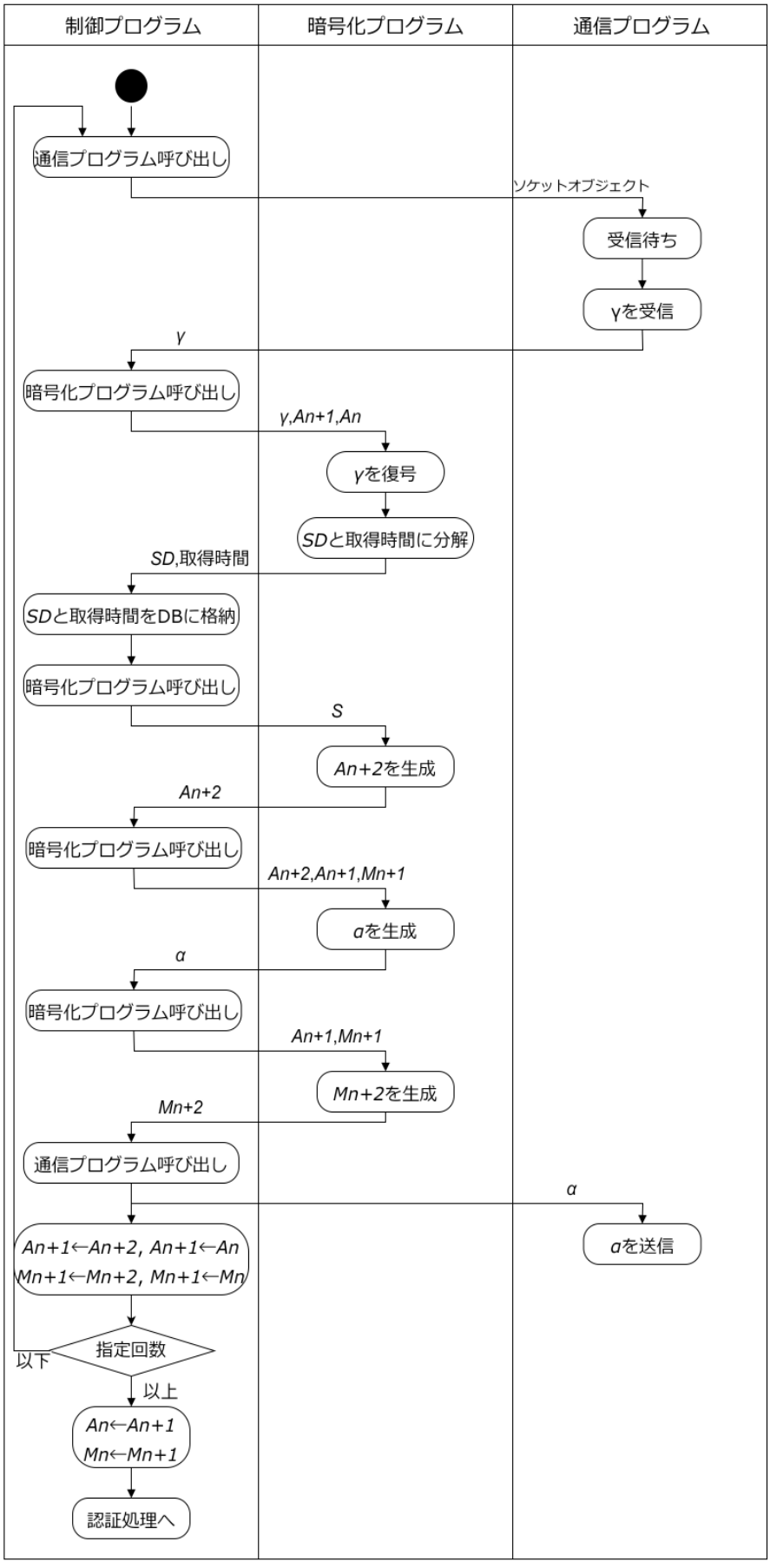


図 5.3　暗号通信のアクティビティ図

5.3　各機能の実装方法

前節で、機能ごとにプログラムを作成し、各プログラムを制御プログラムで呼び出すことでサーバとして機能させると述べた。本節では、各プログラムの概要を説明する。まず、制御プログラムについて述べる。制御プログラムは Python で記述している。表 5.2 に制御プログラムの機能および保存しているデータを示す。

表 5.2　制御プログラムの機能および保存データ

機能	ソケットを作成する。
	通信プログラムを呼び出す。
	認証プログラムを呼び出す。
	暗号通信プログラムを呼び出す。
	データベースにセンシングデータと取得時間を格納する。
保存データ	ユーザ識別子S
	認証情報A_n, 次回認証情報A_{n+1}
	秘匿情報M_n, 次回秘匿情報M_{n+1}
	IPアドレス, ポート番号

通信プログラムについて述べる。通信プログラムは Python で記述しており、5 つの関数から構成される。各関数の機能を表 5.3 に示す。

表 5.3　通信プログラムの機能

機能
接続を受け取る。ソケットオブジェクトを返す。
認証要請以外のデータを受信し、受信したデータを返す。 ただし、5秒以上受信できなかった場合、0を返す。
認証要請を受信し、受信した認証要請を返す。 認証要請を受信するまで待機する。
データを送信する。
コネクションを切断する。

SAS-L2 認証プログラムについて述べる。SAS-L2 認証プログラムは C 言語で記述しており、4 つの関数から構成される。加算の際に桁溢れした場合は、切り捨てている。また、次回認証情報を生成する際に使用する一方向性関数は MD5 を使用している。MD5 には第 2 章で述べた問題点があるが、本研究で使用するエッジデバイスの Arduino の処理負荷を考慮して、SHA-224 や SHA-256 よりもハッシュ長が短い MD5 を採用する。各関数の機能を表 5.4 に示す。

表 5.4　SAS-L2 認証プログラムの機能

機能
次回認証情報A_{n+1}を生成する。
送信情報 α を生成する。
認証を判定する。
次回秘匿情報M_{n+1}を生成する。

暗号プログラムについて述べる。暗号通信プログラムは C 言語で記述しており、4 つの関数から構成される。加算の桁溢れが生じた場合は切り捨て、次回認証情報を生成する際に使用する一方向性関数は MD5 を採用している。各関数の機能を表 5.5 に示す。

表 5.5　暗号通信プログラムの機能

機能
受信情報 γ を復号し、センシングデータと取得時間に分割する。
次回認証情報A_{n+2}を生成する。
送信情報 α を生成する。
次回秘匿情報M_{n+2}を生成する。

SAS-L2 初期登録プログラムについて述べる。SAS-L2 初期登録プログラムは C 言語で記述されている。制御プログラムで呼び出されるのではなく、独立で実行し、生成されたデータをユーザとサーバで共有している。SAS-L2 初期登録プログラムの機能を表 5.6 に示す。

表 5.6　SAS-L2 初期登録プログラムの機能

機能
初回認証情報A_1を生成する。
初回秘匿情報M_1を生成する。

以下に制御プログラムで SAS-L2 認証プログラムと暗号化通信プログラムを呼び出す際のソースコードの一部を示す。SAS-L2 認証プログラムの実行ファイル名は sasl2.so、暗号化通信プログラムの実行ファイル名は crypto.so とする。また、An1 と sd_date はそれぞれ実行ファイル内の関数である。

制御プログラムから SAS-L2 認証プログラムと暗号化通信プログラムを呼び出す際は ctypes を利用し、ポインタ渡しでデータの受け渡しを行っている。上に示したソースコードでは、実行ファイルの関数を呼び出す際の引数や返り値の型の指定を行っている。

```
import ctypes


sasl2_lib = ctypes.cdll.LoadLibrary('./sasl2.so')
sasl2_lib.An1.argtypes = (ctypes.POINTER(ctypes.c_ubyte),
  ctypes.POINTER(ctypes.c_ubyte))


crypto_lib = ctypes.cdll.LoadLibrary('./crypto.so')
crypto_lib.sd_date.argtypes = (ctypes.POINTER(ctypes.c_ubyte),
  ctypes.POINTER(ctypes.c_ubyte), ctypes.POINTER(ctypes.c_ubyte),
  ctypes.POINTER(ctypes.c_float), ctypes.POINTER(ctypes.c_int))
```

以下に通信プログラムの関数のソースコードを示す。

関数 connect は、引数としてソケット識別子を受け取り、接続を受け付け、ソケットオブジェクトを返すという動作をする。

関数 receive_request は、引数としてソケットオブジェクトを受け取り、認証要請を受信し、受信した認証要請を返すという動作をする。受信までの時間の制限はなく、認証要請を受信するまで待機する。

関数 receive は、引数としてソケットオブジェクトを受け取り、データを受信し、受信したデータを返すという動作をする。ただし、5 秒間待機してもデータを受信できない場合、0 を返す。

関数 send は、引数としてソケットオブジェクトと送信データを受け取り、送信データを送信するという動作をする。

関数 close は、引数としてソケット識別子とソケットオブジェクトを受け取り、ソケットを閉じるという動作をする。

```
def connect(sock):
    conn, addr = sock.accept()
    return conn


def receive_request(conn):
    data = conn.recv(1024)
    return data


def receive(conn):
    conn.settimeout(5.0)
    try:
        data = conn.recv(1024)
    except socket.timeout:
        data = 0
    return data
```

```
def send(conn, data):
    conn.send(data)


def close(sock, conn):
    conn.close()
```

以下に SAS-L2 認証プログラムの次回認証情報 A_{n+1} を生成する関数 An1 のソースコードを示す。

関数 An1 では、引数としてユーザ識別子と生成された A_{n+1} を格納する配列を受け取り、以下の手順で動作している。

1. RAND_bytes を用いて乱数を生成する。
2. ユーザ識別子と乱数の排他的論理和をとる。
3. 2 で生成した値を MD5 でハッシュ化する。

```
#define byte 16


void An1(unsigned char pass[byte], unsigned char An1[byte]){
  int n, i, md5;
  unsigned char rand[byte], xor[byte];
  MD5_CTX c;

  n = RAND_bytes(rand, byte);
  if(n != 1){
    printf("error: random number.\n");
  }

  for(i = 0; i < byte; i++){
    xor[i] = pass[i] ^ rand[i];
  }

  md5 = MD5_Init(&c);
  if(md5 != 1){
    perror("error: md5 context");
    exit(1);
  }
```

```
    md5 = MD5_Update(&c, xor, byte);
    if(md5 != 1){
      perror("error: hash");
      exit(1);
    }
    md5 = MD5_Final(An1, &c);
    if(md5 != 1){
      perror("error: output hash");
      exit(1);
    }

  }
```

以下に SAS-L2 認証プログラムの送信情報 α を生成する関数 alpha のソースコードを示す。

関数 alpha は、引数として次回認証情報 A_{n+1}、今回認証情報 A_n、今回秘匿情報 M_n、生成された送信情報 α を格納するための配列を受け取り、A_{n+1}、A_n、M_n の排他的論理和をとるという動作をする。

```
#define byte 16


void alpha(unsigned char An1[byte], unsigned char An[byte],
           unsigned char Mn[byte], unsigned char alpha[byte]){
  int i;


  for(i = 0; i < byte; i++){
    alpha[i] = An1[i] ^ An[i] ^ Mn[i];
  }


}
```

以下に SAS-L2 認証プログラムの認証を判定する関数 judge のソースコードを示す。

関数 judge は、引数として次回認証情報 A_{n+1}、今回認証情報 A_n、受信情報 β を受け取り、以下の手順で動作している。

1. 繰り上げの値を格納する配列 *carry* を初期化する。
2. A_{n+1}、A_n、*carry* を加算する。最上位のビットが桁溢れした場合は切り捨てる。
3. 2 で求めた値が β と等しい場合は 1 を返し、等しくない場合は 0 を返す。

```
#define byte 16

int judge(unsigned char An1[byte], unsigned char An[byte],
          unsigned char beta[byte]){
  int i, result;
  unsigned char carry[byte], judge[byte];

  for(i = 0; i < byte; i++){
    carry[i] = 0;
  }
  for(i = 15; i >= 0; i--){
    if((An1[i] + An[i] + carry[i]) >= 256){
      judge[i] = (An1[i] + An[i] + carry[i]) - 256;
      if(i != 0){
carry[(i - 1)] = 1;
      }
    }
    else{
      judge[i] = An1[i] + An[i] + carry[i];
    }
  }
```

```
  result = 1;
  for(i = 0; i < byte; i++){
    if(judge[i] != beta[i]){
      result = 0;
      break;
    }

  return result;

}
```

以下に SAS-L2 認証プログラムの次回秘匿情報 M_{n+1} を生成する関数 Mn1 のソースコードを示す。

関数 Mn1 は、引数として今回認証情報 A_n、今回秘匿情報 M_n、生成された次回秘匿情報 M_{n+1} を受け取り、以下の手順で動作している。

1. 繰り上げの値を格納する配列 *carry* を初期化する。

2. A_n、M_n、*carry* を加算する。最上位のビットが桁溢れした場合は切り捨てる。

```
#define byte 16


void Mn1(unsigned char An[byte], unsigned char Mn[byte],
         unsigned char Mn1[byte]){
  int i;
  unsigned char carry[byte];

  for(i = 0; i < byte; i++){
    carry[i] = 0;
  }
  for(i = 15; i >= 0; i--){
    if((An[i] + Mn[i] + carry[i]) >= 256){
      Mn1[i] = (An[i] + Mn[i] + carry[i]) - 256;
      if(i != 0){
carry[(i - 1)] = 1;
      }
    }
    else{
      Mn1[i] = An[i] + Mn[i] + carry[i];
    }
  }


}
```

暗号化通信プログラムのソースコードを示す。ただし、暗号通信プログラムでは、以下に示す共用体を用いるものとする。

```
union IntAndFloat{
  int ival;
  float fval;
};
```

以下に暗号化通信プログラムの暗号化されたセンシングデータと取得時間を復号して保存する関数 sd_date のソースコードを示す。

関数 sd_date は、引数として受信情報 γ、次回認証情報 A_{n+1}、今回認証情報 A_n、復号したセンシングデータを格納する配列 *sd*、復号した取得時間を格納する配列 *date* を受け取り、以下の手順で動作する。

1. γ、A_{n+1}、A_n の排他的論理和をとり、復号する。
2. 4 章で示したデータファーマットに従って、1 で生成した値をそれぞれ配列に格納する。
3. シフトと AND 演算を行い、センシングデータは float 型、取得時間は int 型として保存する。

```
#define byte 16


void sd_date(unsigned char gamma[byte], unsigned char An1[byte],
             unsigned char An[byte], float sd[1], int date[6]){
  int i, j, sift, t;
  float sd_f;
  int data[16] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      _sd[4] = {0, 0, 0, 0}, _year[4] = {0, 0}, _month[1] = {0},
      _day[1] = {0}, _hour[1] = {0}, _min[1] = {0}, _sec[1] = {0};
  union IntAndFloat target;
```

```
for(i = 0; i < byte; i++){
  data[i] = gamma[i] ^ An1[i] ^ An[i];
}

for(i = 0; i < 4; i++){
  _sd[i] = data[i];
}
for(i = 0; i < 2; i++){
  _year[i] = data[(i + 4)];
}
_month[0] = data[6];
_day[0] = data[7];
_hour[0] = data[8];
_min[0] = data[9];
_sec[0] = data[10];

target.fval = sd[0];
sift = 0;
for (i = 3; i >= 0; i--) {
  _sd[i] = _sd[i] << sift;
  target.ival = _sd[i] | target.ival;
  sift += 8;
}
sd[0] = target.fval;

sift = 0;
for (i = 1; i >= 0; i--) {
  _year[i] = _year[i] << sift;
  date[0] = _year[i] | date[0];
  sift += 8;
}
```

```
    date[1] = _month[0] | date[1];
    date[2] = _day[0] | date[2];
    date[3] = _hour[0] | date[3];
    date[4] = _min[0] | date[4];
    date[5] = _sec[0] | date[5];

}
```

暗号化通信プログラムには、次回認証情報 A_{n+2} を生成する関数 An2、送信情報 α を生成する関数 alpha、次回秘匿情報 M_{n+2} を生成する関数 Mn2 があるが、SAS-L2 認証プログラムの関数 An1、関数 alpha、関数 Mn1 と同様の動作をするため、記述は省略する。

以下に SAS-L2 初期登録プログラムのソースコードを示す。

SAS-L2 初期登録プログラムでは、SAS-L2 認証プログラムの関数 An1 と同様の処理を行い、初回認証情報 A_1 を生成し、RAND_bytes を用いた乱数を初回秘匿情報 M_1 として生成する。

```
#define byte 16


void A1(unsigned char pass[byte], unsigned char A1[byte]){
  int n, i, md5;
  unsigned char rand[byte], xor[byte];
  MD5_CTX c;

  n = RAND_bytes(rand, byte);
  if(n != 1){
    printf("error: random number.\n");
  }

  for(i = 0; i < byte; i++){
    xor[i] = pass[i] ^ rand[i];
  }

  md5 = MD5_Init(&c);
  if(md5 != 1){
    perror("error: md5 context");
    exit(1);
  }
```

```
  md5 = MD5_Update(&c, xor, byte);
  if(md5 != 1){
    perror("error: hash");
    exit(1);
  }
  md5 = MD5_Final(A1, &c);
  if(md5 != 1){
    perror("error: output hash");
    exit(1);
  }

}

void M1(unsigned char M1[16]){
  int n, i, md5;
  unsigned char rand[16], xor[16];
  MD5_CTX c;

  n = RAND_bytes(M1, byte);
  if(n != 1){
    printf("error: random number(M1).\n");
  }

}
```

第6章　検証・評価

本章では、実装したシステムの性能に対する検証方法および検証結果、評価方法および評価を述べる。

6.1　検証方法

本研究では、以下の手順で開発したシステムの動作を確認する。

1. 本システムにおけるサーバが行う演算結果を確認する。
2. エッジ側とサーバ側を結合させ、認証および暗号通信を行う。

ただし、認証および暗号通信は以下の条件で行う。

- 3 台の Arduino は、毎分 0 秒になると認証要請を Raspbery Pi に送信する。
- 認証不成立の場合の確認を行う際は、あらかじめサーバ側のプログラムを、Arduino から β を受信した後に、β の値を 0 に書き換えるように書く。
- 各 Arduino が 3 回認証および暗号通信を行うと、強制終了する。

6.2　評価方法

本研究は、第 4 章で述べた要件定義および基本設計、第 5 章で述べた詳細設計に基づき、システムテスト、結合テスト、単体テストを作成して、それぞれのテスト項目を満たしているかで評価する。表 6.1 にシステムテストの項目、表 6.2 に結合テストの項目、表 6.3 に単体テストの項目を示す。なお、単体テストでは、5.3 節で述べた各プログラムの演算が正しく行えているかを確認している。また、次回認証情報 A_{n+1} と A_{n+2}、次回秘匿情報 M_{n+1}、M_{n+2} は、それぞれ同じ演算方法であるため、まとめてテストを行う。

表 6.1　システムテストの項目

機能	デバイス	テスト項目
認証	Arduino	起動後に時刻同期を行う。
		設定した時刻にサーバにコネクションし、認証要求を送信している。
		複数台のArduinoが同時刻に認証要請を送信した場合、他のArduinoが通信を終了するまで待機する。
		SAS-L2認証処理を行う。
		認証結果をモニタに表示する。
		認証に成功した場合、暗号通信を行う。
		認証に失敗した場合、コネクションを切断し、次の設定時刻まで待機する。
	RASPBERRY PI	起動後に認証要請の待ち状態になる。
		認証要請を受信すると、ユーザを識別してSAS-L2認証処理を行う。
		認証結果を表示する。
		認証に成功した場合、暗号通信を行う。
		認証に失敗した場合、コネクションを切断し、認証要請の待ち状態になる。
暗号化	Arduino	センシングデータを取得する。
		センシングデータおよび取得時間を暗号化してRASPBERRY PIに送信する。
		これらの処理を10回行う。
		通信が終了すると、コネクションを切断し、次の設定時刻まで待機している。
	RASPBERRY PI	センシングデータと取得時間を復号する。
		センシングデータと取得時間をデータベースに格納する。
		センシングデータと取得時間をデータベースに格納する際、Arduinoごとにテーブルを分けている。
		これらの処理を10回行う。
		通信が終了すると、コネクションを切断し、認証要請待ち状態になる。
通信	Arduino	5秒以上データを受信できない場合、コネクションを切断し、次の設定時刻まで待機している。
		通信エラーが発生した場合、LEDを点滅させる。
	RASPBERRY PI	5秒以上データを受信できない場合、コネクションを切断し、認証要請待ち状態になる。
		3台のArduinoが同時刻に認証要請を送信した場合、3台のArduinoとの通信を10秒以内に完了する。

表 6.2　結合テストの項目結果

機能	テスト項目
認証	認証要請の送受信ができる。
	サーバからユーザに α を送信し、両者が保持している α の値が一致している。
	ユーザからサーバに β を送信し、両者が保持している β の値が一致している。
	認証結果の送受信ができる。
	次回認証情報 A_{n+1} と次回秘匿情報 M_{n+1} が一致している。
暗号化	ユーザからサーバに γ を送信し、両者が保持している γ の値が一致している。
	ユーザとサーバが保持しているセンシングデータと取得時間が一致している。
	サーバからユーザに α を送信し、両者が保持している α の値が一致している。
	通信終了後のユーザとサーバが保存している認証情報 A_n と秘匿情報 M_n が一致している。

表 6.3　単体テストの項目

テスト項目
次回認証情報 A_{n+1}、A_{n+2} を正しく生成している。
α を正しく生成している。
$A_{n+1} + A_n$ が正しく生成している。
次回秘匿情報 M_{n+1}、M_{n+2} を正しく生成している。

6.3　検証結果

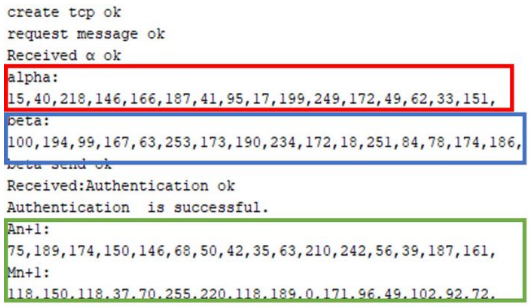
本システムのサーバが行う演算の結果を図 6.1 に示す。以下の 5 点を確認することができた。ただし、一方向性関数を使用する際は、入力と出力の値が異なることで、正しく計算できているものとする。

- 次回認証情報 A_{n+1} が正しく計算できている。
- α が正しく計算できている。
- $A_{n+1} + A_n$ が正しく計算できている。
- 次回秘匿情報 M_{n+1} が正しく計算できている。
- 加算演算で桁溢れが発生した場合、桁溢れした部分を切り捨てている。

図 6.1　演算の結果

認証処理の結果を図 6.2 に示す。以下の 4 点を確認することができた。

- 図 6.2(a) および図 6.2(b) の赤枠の箇所より、α が正常に送受信されている。
- 図 6.2(a) および図 6.2(b) の青枠の箇所より、β が正常に送受信されている。
- 図 6.2(a) および図 6.2(b) の緑枠の箇所より、認証後に保存している次回認証情報と次回秘匿情報の値が一致している。
- SAS-L2 認証処理手順に沿って、認証が成立している。



(a)Arduino の認証結果

(b)Raspberry Pi の認証結果

図 6.2　認証結果

認証不成立時の出力画面を図 6.3 に示す。以下の 2 点を確認することができた。

- SAS-L2 認証処理手順に沿って、認証が不成立になっている。
- 認証不成立の場合、通信していた Arduino とのコネクションを切断し、認証要請の待ち状態になっている。

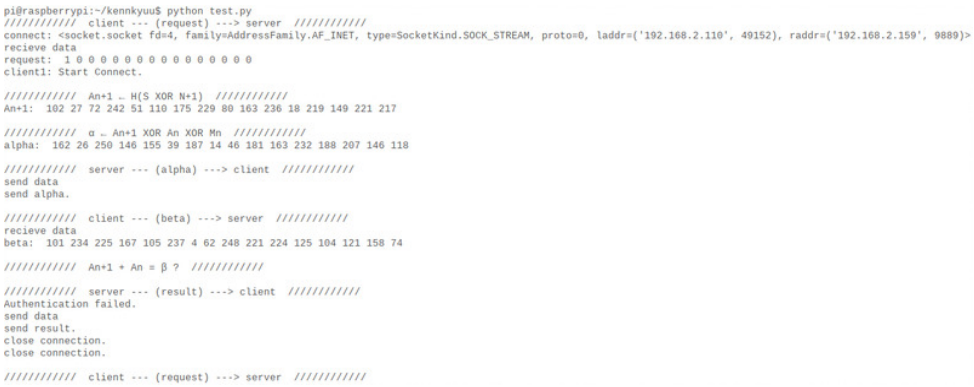
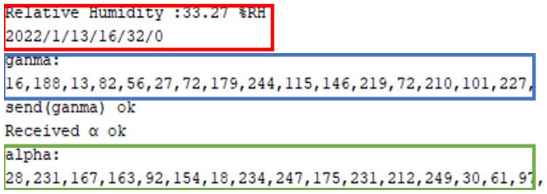


図 6.3　認証失敗時の出力画面

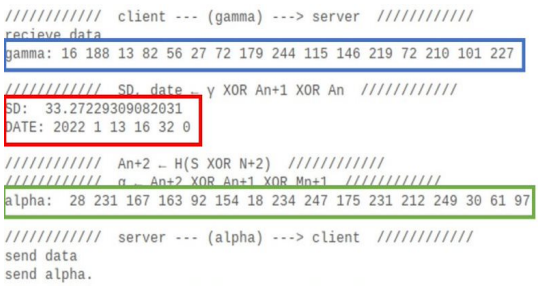
暗号通信の結果を図 6.4 に示す。以下の 4 点を確認することができた。ただし、Raspberry Pi で復号したセンシングデータの小数点第三位以下には、環境依存による誤差が表示されている。

- 図 6.4(a) および図 6.4(b) の赤枠の箇所より、センシングデータと取得時間を正しく復号し、Arduino と Raspberry Pi で同じ値を保持している。

- 図 6.4(a) および図 6.4(b) の青枠の箇所より、γ が正常に送受信されている。
- 図 6.4(a) および図 6.4(b) の緑枠の箇所より、α が正常に送受信されている。
- SAS-L2 を用いた暗号方式の処理手順に沿って、暗号通信を行っている。



(a)Arduino の暗号通信結果

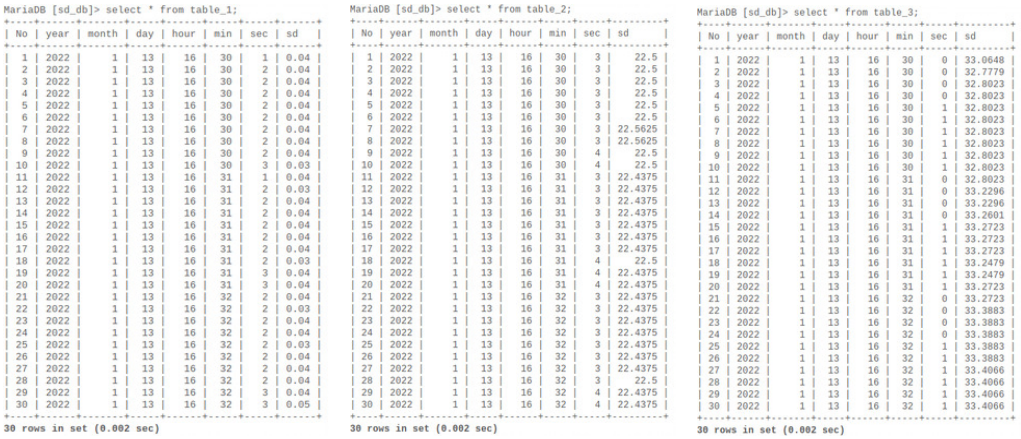


(b)Raspberry Pi の暗号通信結果

図 6.4　暗号通信結果

各 Arduino から受信したセンシングデータと取得時間を格納したデータベースのテーブルを図 6.5 に示す。以下の点を確認することができた。

- Arduino ごとにテーブルを分けてセンシングデータと取得時間を格納している。
- データベースに格納されたセンシングデータと取得時間と、Arduino が送信したセンシングデータと取得時間が一致している。
- 一度の通信で、センシングデータと取得時間を 10 回送受信している。
- 同時刻に認証要請を送信した 3 台の Arduino との通信を 3〜4 秒で完了している。
- 各 Arduino と 3 回通信が行われているということから、暗号通信後に保存している次回認証情報と次回秘匿情報の値が、Arduino と Raspberry Pi で一致している。



```
MariaDB [sd_db]> select * from table_1;
+----+------+-------+-----+------+-----+-----+------+
| No | year | month | day | hour | min | sec | sd   |
+----+------+-------+-----+------+-----+-----+------+
|  1 | 2022 |     1 |  13 |   16 |  30 |   1 | 0.04 |
|  2 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
|  3 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
|  4 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
|  5 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
|  6 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
|  7 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
|  8 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
|  9 | 2022 |     1 |  13 |   16 |  30 |   2 | 0.04 |
| 10 | 2022 |     1 |  13 |   16 |  30 |   3 | 0.03 |
| 11 | 2022 |     1 |  13 |   16 |  31 |   1 | 0.04 |
| 12 | 2022 |     1 |  13 |   16 |  31 |   2 | 0.03 |
| 13 | 2022 |     1 |  13 |   16 |  31 |   2 | 0.04 |
| 14 | 2022 |     1 |  13 |   16 |  31 |   2 | 0.04 |
| 15 | 2022 |     1 |  13 |   16 |  31 |   2 | 0.04 |
| 16 | 2022 |     1 |  13 |   16 |  31 |   2 | 0.04 |
| 17 | 2022 |     1 |  13 |   16 |  31 |   2 | 0.04 |
| 18 | 2022 |     1 |  13 |   16 |  31 |   2 | 0.03 |
| 19 | 2022 |     1 |  13 |   16 |  31 |   3 | 0.04 |
| 20 | 2022 |     1 |  13 |   16 |  31 |   3 | 0.04 |
| 21 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.04 |
| 22 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.03 |
| 23 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.04 |
| 24 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.04 |
| 25 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.03 |
| 26 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.04 |
| 27 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.04 |
| 28 | 2022 |     1 |  13 |   16 |  32 |   2 | 0.04 |
| 29 | 2022 |     1 |  13 |   16 |  32 |   3 | 0.04 |
| 30 | 2022 |     1 |  13 |   16 |  32 |   3 | 0.05 |
+----+------+-------+-----+------+-----+-----+------+
30 rows in set (0.002 sec)
```

```
MariaDB [sd_db]> select * from table_2;
+----+------+-------+-----+------+-----+-----+---------+
| No | year | month | day | hour | min | sec | sd      |
+----+------+-------+-----+------+-----+-----+---------+
|  1 | 2022 |     1 |  13 |   16 |  30 |   3 |    22.5 |
|  2 | 2022 |     1 |  13 |   16 |  30 |   3 |    22.5 |
|  3 | 2022 |     1 |  13 |   16 |  30 |   3 |    22.5 |
|  4 | 2022 |     1 |  13 |   16 |  30 |   3 |    22.5 |
|  5 | 2022 |     1 |  13 |   16 |  30 |   3 |    22.5 |
|  6 | 2022 |     1 |  13 |   16 |  30 |   3 |    22.5 |
|  7 | 2022 |     1 |  13 |   16 |  30 |   3 | 22.5625 |
|  8 | 2022 |     1 |  13 |   16 |  30 |   3 | 22.5625 |
|  9 | 2022 |     1 |  13 |   16 |  30 |   4 |    22.5 |
| 10 | 2022 |     1 |  13 |   16 |  30 |   4 |    22.5 |
| 11 | 2022 |     1 |  13 |   16 |  31 |   3 | 22.4375 |
| 12 | 2022 |     1 |  13 |   16 |  31 |   3 | 22.4375 |
| 13 | 2022 |     1 |  13 |   16 |  31 |   3 | 22.4375 |
| 14 | 2022 |     1 |  13 |   16 |  31 |   3 | 22.4375 |
| 15 | 2022 |     1 |  13 |   16 |  31 |   3 | 22.4375 |
| 16 | 2022 |     1 |  13 |   16 |  31 |   3 | 22.4375 |
| 17 | 2022 |     1 |  13 |   16 |  31 |   3 | 22.4375 |
| 18 | 2022 |     1 |  13 |   16 |  31 |   4 |    22.5 |
| 19 | 2022 |     1 |  13 |   16 |  31 |   4 | 22.4375 |
| 20 | 2022 |     1 |  13 |   16 |  31 |   4 | 22.4375 |
| 21 | 2022 |     1 |  13 |   16 |  32 |   3 | 22.4375 |
| 22 | 2022 |     1 |  13 |   16 |  32 |   3 | 22.4375 |
| 23 | 2022 |     1 |  13 |   16 |  32 |   3 | 22.4375 |
| 24 | 2022 |     1 |  13 |   16 |  32 |   3 | 22.4375 |
| 25 | 2022 |     1 |  13 |   16 |  32 |   3 | 22.4375 |
| 26 | 2022 |     1 |  13 |   16 |  32 |   3 | 22.4375 |
| 27 | 2022 |     1 |  13 |   16 |  32 |   3 | 22.4375 |
| 28 | 2022 |     1 |  13 |   16 |  32 |   3 |    22.5 |
| 29 | 2022 |     1 |  13 |   16 |  32 |   4 | 22.4375 |
| 30 | 2022 |     1 |  13 |   16 |  32 |   4 | 22.4375 |
+----+------+-------+-----+------+-----+-----+---------+
30 rows in set (0.002 sec)
```

```
MariaDB [sd_db]> select * from table_3;
+----+------+-------+-----+------+-----+-----+---------+
| No | year | month | day | hour | min | sec | sd      |
+----+------+-------+-----+------+-----+-----+---------+
|  1 | 2022 |     1 |  13 |   16 |  30 |   0 | 33.0648 |
|  2 | 2022 |     1 |  13 |   16 |  30 |   0 | 32.7779 |
|  3 | 2022 |     1 |  13 |   16 |  30 |   0 | 32.8023 |
|  4 | 2022 |     1 |  13 |   16 |  30 |   0 | 32.8023 |
|  5 | 2022 |     1 |  13 |   16 |  30 |   1 | 32.8023 |
|  6 | 2022 |     1 |  13 |   16 |  30 |   1 | 32.8023 |
|  7 | 2022 |     1 |  13 |   16 |  30 |   1 | 32.8023 |
|  8 | 2022 |     1 |  13 |   16 |  30 |   1 | 32.8023 |
|  9 | 2022 |     1 |  13 |   16 |  30 |   1 | 32.8023 |
| 10 | 2022 |     1 |  13 |   16 |  30 |   1 | 32.8023 |
| 11 | 2022 |     1 |  13 |   16 |  31 |   0 | 32.8023 |
| 12 | 2022 |     1 |  13 |   16 |  31 |   0 | 33.2296 |
| 13 | 2022 |     1 |  13 |   16 |  31 |   0 | 33.2296 |
| 14 | 2022 |     1 |  13 |   16 |  31 |   0 | 33.2601 |
| 15 | 2022 |     1 |  13 |   16 |  31 |   1 | 33.2723 |
| 16 | 2022 |     1 |  13 |   16 |  31 |   1 | 33.2723 |
| 17 | 2022 |     1 |  13 |   16 |  31 |   1 | 33.2723 |
| 18 | 2022 |     1 |  13 |   16 |  31 |   1 | 33.2479 |
| 19 | 2022 |     1 |  13 |   16 |  31 |   1 | 33.2479 |
| 20 | 2022 |     1 |  13 |   16 |  31 |   1 | 33.2723 |
| 21 | 2022 |     1 |  13 |   16 |  32 |   0 | 33.2723 |
| 22 | 2022 |     1 |  13 |   16 |  32 |   0 | 33.3883 |
| 23 | 2022 |     1 |  13 |   16 |  32 |   0 | 33.3883 |
| 24 | 2022 |     1 |  13 |   16 |  32 |   0 | 33.3883 |
| 25 | 2022 |     1 |  13 |   16 |  32 |   1 | 33.3883 |
| 26 | 2022 |     1 |  13 |   16 |  32 |   1 | 33.3883 |
| 27 | 2022 |     1 |  13 |   16 |  32 |   1 | 33.4066 |
| 28 | 2022 |     1 |  13 |   16 |  32 |   1 | 33.4066 |
| 29 | 2022 |     1 |  13 |   16 |  32 |   1 | 33.4066 |
| 30 | 2022 |     1 |  13 |   16 |  32 |   1 | 33.4066 |
+----+------+-------+-----+------+-----+-----+---------+
30 rows in set (0.002 sec)
```

(a)UV センサ　　(b) 温度センサ　　(c) 湿度センサ

図 6.5　データベースの各テーブル

6.4　評価

単体テストの結果を表 6.4、結合テストの結果を表 6.5、システムテストの結果を表 6.6 に示す。それぞれのテストにおいて期待どおりの動作が確認でき、第 4 章で述べた仕様および基本設計、第 5 章で述べた詳細設計で設定した機能を実現することができた。

表 6.4　単体テストの結果

テスト項目	テスト結果	テスト日	テスト担当者
次回認証情報A_{n+1}、A_{n+2}を正しく生成している。	○	1月17日	内山田
αを正しく生成している。	○	1月17日	内山田
$A_{n+1} + A_n$が正しく生成している。	○	1月17日	内山田
次回秘匿情報M_{n+1}、M_{n+2}を正しく生成している。	○	1月17日	内山田

表 6.5　結合テストの結果

機能	テスト項目	テスト結果	テスト日	テスト担当者
認証	認証要請の送受信ができる。	○	1月14日	浅野・内山田
	サーバからユーザにαを送信し、両者が保持しているαの値が一致している。	○	1月14日	浅野・内山田
	ユーザからサーバにβを送信し、両者が保持しているβの値が一致している。	○	1月14日	浅野・内山田
	認証結果の送受信ができる。	○	1月14日	浅野・内山田
	次回認証情報A_{n+1}と次回秘匿情報M_{n+1}が一致している。	○	1月14日	浅野・内山田
暗号化	ユーザからサーバにγを送信し、両者が保持しているγの値が一致している。	○	1月14日	浅野・内山田
	ユーザとサーバが保持しているセンシングデータと取得時間が一致している。	○	1月14日	浅野・内山田
	サーバからユーザにαを送信し、両者が保持しているαの値が一致している。	○	1月14日	浅野・内山田
	通信終了後のユーザとサーバが保存している認証情報A_nと秘匿情報M_nが一致している。	○	1月14日	浅野・内山田

表 6.6　システムテストの結果

機能	デバイス	テスト項目	テスト結果	テスト日	テスト担当者
認証	Arduino	起動後に時刻同期を行う。	○	1月14日	浅野・内山田
		設定した時刻にサーバにコネクションし、認証要求を送信している。	○	1月14日	浅野・内山田
		複数台のArduinoが同時刻に認証要請を送信した場合、他のArduinoが通信を終了するまで待機する。	○	1月14日	浅野・内山田
		SAS-L2認証処理を行う。	○	1月14日	浅野・内山田
		認証結果をモニタに表示する。	○	1月14日	浅野・内山田
		認証に成功した場合、暗号通信を行う。	○	1月14日	浅野・内山田
		認証に失敗した場合、コネクションを切断し、次の設定時刻まで待機する。	○	1月14日	浅野・内山田
	RASPBERRY PI	起動後に認証要請の待ち状態になる。	○	1月14日	浅野・内山田
		認証要請を受信すると、ユーザを識別してSAS-L2認証処理を行う。	○	1月14日	浅野・内山田
		認証結果を表示する。	○	1月14日	浅野・内山田
		認証に成功した場合、暗号通信を行う。	○	1月14日	浅野・内山田
		認証に失敗した場合、コネクションを切断し、認証要請の待ち状態になる。	○	1月14日	浅野・内山田
暗号化	Arduino	センシングデータを取得する。	○	1月14日	浅野・内山田
		センシングデータおよび取得時間を暗号化してRASPBERRY PIに送信する。	○	1月14日	浅野・内山田
		これらの処理を10回行う。	○	1月14日	浅野・内山田
		通信が終了すると、コネクションを切断し、次の設定時刻まで待機している。	○	1月14日	浅野・内山田
	RASPBERRY PI	センシングデータと取得時間を復号する。	○	1月14日	浅野・内山田
		センシングデータと取得時間をデータベースに格納する。	○	1月14日	浅野・内山田
		センシングデータと取得時間をデータベースに格納する際、Arduinoごとにテーブルを分けている。	○	1月14日	浅野・内山田
		これらの処理を10回行う。	○	1月14日	浅野・内山田
		通信が終了すると、コネクションを切断し、認証要請待ち状態になる。	○	1月14日	浅野・内山田
通信	Arduino	5秒以上データを受信できない場合、コネクションを切断し、次の設定時刻まで待機している。	○	1月14日	浅野・内山田
		通信エラーが発生した場合、LEDを点滅させる。	○	1月14日	浅野・内山田
	RASPBERRY PI	5秒以上データを受信できない場合、コネクションを切断し、認証要請待ち状態になる。	○	1月14日	浅野・内山田
		3台のArduinoが同時刻に認証要請を送信した場合、3台のArduinoとの通信を10秒以内に完了する。	○	1月14日	浅野・内山田

第7章　考察

本章では、SAS を用いたセキュアな IoT システムの性能に対する考察と、本システムの発展に対する考察を述べる。

まず、SAS を用いたセキュアな IoT システムの性能の考察を述べる。本研究では、認証方式に被認証側の処理負荷が小さい SAS-L2 認証方式を採用することで、処理能力が低いエッジデバイスに対して、認証機能を搭載することに成功した。さらに、SAS-L2 を利用した暗号方式を搭載することで、被認証側の処理負荷が小さく、バーナム暗号よりも安全性が高い暗号通信を、同様のエッジデバイスに搭載することに成功した。これらの結果から、処理能力が低いエッジデバイスに対して、認証機能および暗号通信機能を搭載することで、セキュリティ機能を搭載することができたと言える。一方、本システムは、エッジデバイスの処理負荷を考慮し、SAS-L2 および SAS-L2 を用いた暗号方式を実装する際の一方向性関数に、SHA-224 や SHA-256 と比較するとハッシュ値が短い MD5 を使用しているため、脆弱性がある。暗号方式の軽量化を行い、エッジデバイスの処理負荷をさらに小さくし、一方向性関数に SHA-224 や SHA-256 を使用することで、この問題を解決することができると考える。

次に、SAS を用いたセキュアな IoT システムの発展に対する考察を述べる。本システムでは、サーバが 1 つのスレッドで複数のユーザと通信を行っているため、ユーザが同時刻に認証要請をサーバに送信して通信を開始すると、1 つのユーザごとに約 1 秒間待機する必要がある。本研究では、3 台のエッジデバイスとサーバが通信を行っているため、すべてのユーザが同時に認証要請を送信した場合でも、約 4 秒ですべてのユーザとの通信が完了していた。しかしながら、ユーザの数が増えると、すべてのユーザとの通信が完了するまでの時間が多くなってしまう。そこで、サーバがマルチスレッドでユーザと通信を行なえば、同時刻に複数のユーザが認証要請を送信した場合でも、すべてのユーザとの通信を約 1 秒で完了することができ、本システムの性能を発展することができると考える。

第8章　あとがき

本研究では、処理能力の低い IoT 機器にセキュリティ機能を搭載することができるシステムの開発を目的とした。そのなかで、片方の処理負荷が低い認証方式である SAS-L2 と、SAS-L2 を利用した暗号方式を搭載した、SAS を用いたセキュアな IoT システムの開発を目標とした。

本研究は、2 名のチームで行い、V 字開発モデルに従って開発を進めた。始めに、チーム全体でシステムの要件定義を行い、UML を用いて基本設計を行った。次に、エッジ側とサーバ側に担当を分け、詳細設計、実装、単体テストを行った。最後に、エッジ側とサーバ側を結合させ、結合テストとシステムテストを行い、開発したシステムに対して評価を行った。

本研究の結果、SAS-L2 認証方式および SAS-L2 認証メカニズムに基づいたデータ通信を暗号化する機能を実装することができ、処理能力の低い IoT 機器間の相互認証機能およびデータ通信を暗号化する機能を有するセキュアな組込みシステムを開発することに成功した。

しかし、SAS-L2 認証機能を実装する際に、よりセキュリティ性の高い一方向性関数を採用し、セキュリティを強化する必要がある。また、サーバがマルチスレッドでユーザと通信を行うことで、複数のユーザと同時に通信する際の通信時間が短縮され、システムの性能が高くなる。このような課題の解決、ユーザとサーバの通信方法の改良を行うことで、本システムはさらにセキュリティ、性能が向上したシステムに発展することが期待できる。

謝辞

本研究を進めるにあたり、懇篤な御指導、御鞭撻を賜わりました本学高橋寛教授に深く御礼申し上げます。

本論文の作成に関し、詳細なるご検討、貴重な御教示を頂きました本学甲斐博准教授ならびに王森レイ講師に深く御礼申し上げます。

本研究に際し、ご審査頂きました本学高橋寛教授、樋上喜信教授、井門俊講師に深く御礼申し上げます。

最後に、多大なご協力と貴重な御助言を頂いた本学計算機・ソフトウェアシステム研究室の諸氏に厚く御礼申し上げます。

参考文献

［1］ ”【活用事例】IoT のメリット・デメリットを紹介”. NTTPC COMMUNICATIONS. 2019-12-20.

https://www.nttpc.co.jp/column/iot_mobile/foundation_iot.html,(参照 2021-12-29)

［2］ ”IPA テクニカルウォッチ「自動車の情報セキュリティ」に関するレポート ネットワーク化・オープン化が進む自動車の安全 ”. 独立行政法人 情報処理推進機構 技術本部 セキュリティセンター. 2012-5-31.

［3］ 竹下隆史、村山公保、荒井透、苅田幸雄. マスタリング TCP/IP 入門編 第 5 版. 東京. オーム社. 2012-2-25. 360p.

［4］ ” 暗号化の AES 方式とは？ほかの種類との違い・実施方法を解説！”. IT トレンド. 2020-11-5.

https://it-trend.jp/encryption/article/64-0070,(参照 2021-12-30)

http://www.picfun.com/mathlib02.html. (参照 2021-1-8).

［5］ ” ブロック暗号とは?特徴やストリーム暗号との違いを解説!”. IT トレンド. 2021-5-13.

https://it-trend.jp/encryption/article/64-0066, (参照 2022-2-18).

［6］ 齊藤貴之. ”3 分で分かる AES”. 日経 XTECH. 2020-3-11.

https://xtech.nikkei.com/atcl/nxt/keyword/18/00002/030800119/, (参照 2021-12-30).

［7］ ”RSA 暗号とは？仕組みや応用事例を初心者にもわかりやすく解説！”. IT トレンド. 2021-1-7.

https://it-trend.jp/encryption/article/64-0056, (参照 2021-12-30).

［8］ ” バーナム暗号”. kisaragi のブログ. 2019-3-25.

https://kisqragi.hatenablog.com/entry/2019/03/25/225922, (参照 2021-12-31).

［9］ yoshida. ”OpenSSL の仕組みとは？初歩から解説！”. CodeCampus. 2017-3-2.

https://blog.codecamp.jp/ssl-mechanism, (参照 2022-1-2).

［10］ ”MD5(Message Digest 5) とは”. 2020-8-24.

https://medium-company.com/md5/, (参照 2022-1-2).

［11］ ” システム開発における V 字モデルとは？W 字についても解説”. 発注ラウンジ. 2019-5-16.

https://hnavi.co.jp/knowledge/blog/system_v-model/. (閲覧日:2022-1-7).

［12］ ”UML 超入門”. オブラブ. 2012.

http://objectclub.jp/technicaldoc/uml/umlintro2. (閲覧日:2022-1-7).