目 次

第 1 章　序論 **1**

1.1　研究背景および目的・目標 1

1.2　本論文の構成 2

第 2 章　準備 **3**

2.1　暗号化手法 3

2.1.1　AES(Advanced Encryption Standard) 3

2.1.2　バーナム暗号 3

2.2　通信プロトコル 4

2.2.1　TCP(Transmission Control Protocol) 4

2.2.2　UDP(User Datagram Protocol) 4

2.3　システム開発プロセス 4

2.3.1　V 字開発モデル 4

2.3.2　UML(Unified Modeling Language) 5

2.3.3　ユースケース図 5

2.3.4　シーケンス図 5

2.3.5　クラス図 6

第 3 章　SAS-L2 認証方法 **7**

3.1　SAS-L2(SimpleAnd Secure password authentication protocol,Light processing version, type 2) の概要 7

3.1.1　初期登録処理 7

3.1.2　認証処理 8

第 4 章　センシングデータ通信の暗号化 **11**

4.1　SAS-L2 に基づいたデータ通信の暗号化 11

4.1.1　送信側のセンシングデータ暗号化 11

4.1.2 受信側のセンシングデータの復号化 12
4.1.3 提案手法の利点 13

第 5 章 SAS-L2 を用いたセキュアな組込みシステムの開発 15

5.1 SAS-L2 を用いたセキュアな組込みシステムの概要 15
5.2 要件定義 16

第 6 章 SAS-L2 を用いたセキュアな組込みシステムの設計 18

6.1 基本設計 18
6.2 役割分担 22
6.3 スケジュール 22
6.4 詳細設計 24
6.4.1 データフォーマット 28
6.5 テスト項目の作成 29
6.5.1 単体テスト項目 29
6.5.2 結合テスト項目 30
6.5.3 総合テスト項目 31

第 7 章 検証 33

7.1 使用端末 33
7.2 使用センサ 33
7.3 開発環境 33
7.4 テスト 34
7.4.1 単体テスト 34
7.4.2 結合テスト 38
7.4.3 総合テスト 41
7.5 考察 47

第 8 章 あとがき 49

謝辞 50

参考文献 51

第1章　序論

1.1　研究背景および目的・目標

近年では,IoT(Internet of Things) の普及が進んでおり, 様々なものがインターネットに繋がる時代である. インターネトに繋がるものを IoT 機器 (IoT デバイス) といい, 例として家電や自動車・産業用ロボットなどが挙げられる. IoT を実現する上で, 周囲の情報を検知するセンシングデバイスやネットワークは必要不可欠である. そして, センシングデバイスがインターネットに繋がることで, 外部の悪意のある第三者からの攻撃により, データを盗聴・改ざんする恐れが懸念される.

第三者からの攻撃事例に, タイヤ空気圧監視システムへの攻撃事例がある [1]. タイヤ空気圧監視システム (TPMS:(Tire Pressure Monitoring System)) とは, タイヤの空気圧を常時監視するシステムであり, 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バースト事故を防ぐ効果が期待される. しかし, この TPMS の無線通信には脆弱性があることが問題となっている. その問題の一つとして,TPMS では通信メッセージが暗号化されていないため, 盗聴解析が容易になるという点がある. また, TPMS の空気圧報告メッセージになりすますことができるという問題点もある. この事例から, センシングデバイスにおいて, ネットワーク上の脆弱性があることで外部からの攻撃によるデータの盗聴・改ざんが行われてしまうリスクがあることから, セキュリティ対策が不可欠となることが分かる.

センシングデバイスのセキュリティ対策の一つに認証・暗号化がある. しかしながら,AES など従来の暗号化方式は処理負荷が高いことから処理性能の低いセンシングデバイスには導入困難であるため, 現在市場に出回っているセンシングデバイスには認証・暗号化といったセキュリティ対策が不十分で, 脆弱性があると考えられる. そこで, センシングデバイスなど, 処理性能の低い IoT 機器において, 極めて小さい処理負荷で認証と暗号化通信のできる軽量なセキュリティ対策が求められる.

本研究では, 処理能力の低いセンシングデバイスで構成される IoT システムにおいて, デバイスとエッジサーバー間の安全なデータ通信を行う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, 高知工学科大学の清水明宏教授が処理能力の低い装置へのセ

キュリティ機能実装のために開発されたワンタイムパスワード認証方式 SAS-L2 を IoT センシングデバイスに実装することで, デバイス間の相互認証およびセンシングデータの暗号化通信方法を実現する. なお, 本研究は,2 人チーム (浅野美咲, 内山田隆太) で V 字開発モデルに従って処理能力の低いセンシングデバイスで構成される SAS-L2 認証を導入したセキュアな IoT システムの開発を行う. システムの設計には UML を利用する. チームメンバーの 2 人で分担し, 内山田がエッジサーバーの実装, 浅野がエッジデバイスの実装を行う.

1.2　本論文の構成

本論文の構成は以下の通りである. 第 1 章では, 研究の背景および目的・目標について述べる. 第 2 章では, 本論文に必要な暗号化手法と通信プロトコル, システム開発プロセスに関する用語について述べる. 第 3 章では,SAS-L2 認証について述べる. 第 4 章では, センシングデータ通信の暗号化について述べる. 第 5 章では, 開発したシステムの概要について述べる. 第 6 章では, システムの設計・テスト項目について述べる. 第 7 章では, 実装したシステムの検証結果と考察を述べる. 第 8 章では, 本論文のまとめを述べる.

第2章　準備

本章では, 本論文で必要な暗号化手法と通信プロトコル, システム開発プロセスに関する用語について説明する.

2.1　暗号化手法

本節では, 従来の暗号化方式として AES とバーナム暗号について説明する.

2.1.1　AES(Advanced Encryption Standard)

AES とは, 無線 LAN などの通信データの暗号化に用いられる暗号化アルゴリズムの一つである. AES は, 共通鍵暗号方式であり, 同じ暗号鍵を利用してデータの送信者が暗号化, データの受信者が復号化を行う. また,128・192・256bit のいずれかの鍵長を使用することが可能である. AES アルゴリズムは, 以下の 4 種類の変換を順番に用いる [2].

SubBytes 16 バイトごとに区切ったデータに対し, テーブルを用いて 1 バイト単位で置換する.

ShiftRows 1 バイト単位でデータの順序を入れ替える.

MixColumns 4 バイトごとに行列演算を行う.

AddRoundKey 128・192・256bit のいずれかの暗号鍵を基に生成した鍵で変換する.

この一連処理を複数回繰り返し暗号化を行う. 復号化では, 以上の 4 種類の変換と逆の変換を行う.

2.1.2　バーナム暗号

バーナム暗号とは,1917 年にギルバート・バーナムが発明した暗号化方式である [3]. 1949 年にはシャノンによって, 理論的に解読不可能であることが証明されている. バーナム暗号は暗号化と復号化の際に同じ秘密鍵を使用する共通鍵暗号方式であり, 秘密鍵は一度しか使

用できないという特徴がある. 秘密鍵が一度しか使用できなことから, 暗号化と復号化を行う度に, 秘密鍵を共有し直さなければならない.

2.2　通信プロトコル

本節では, 本研究で用いた通信プロトコルである TCP と UDP について説明する.

2.2.1　TCP(Transmission Control Protocol)

TCP とは, データの送信を開始する前に, 送信ホストと受信ホストの間で回線の接続をするコネクション型で, 信頼性のあるトランスポート層のプロトコルである [4]. 両端のホスト間でデータの到達性を保障する. ネットワークの帯域幅を有効に利用する仕組みや, ネットワークの混雑を和らげる仕組みなど, 様々な機能が組み込まれており信頼性の向上が図られる.

2.2.2　UDP(User Datagram Protocol)

UDP とは,TCP と異なりコネクションの確立や切断処理がないコネクションレス型で, 信頼性のないトランスポート層プロトコルである [4]. また,UDP には送信されるデータの誤りや順序の違いなどを検出する機能がない. パケット数が少ない通信や, ビデオ音声などのマルチメディア通信に向いている.

2.3　システム開発プロセス

本節では, 本研究でのシステム開発のプロセスに関する用語について説明する.

2.3.1　V 字開発モデル

V 字開発モデルとは, テストを重視し, 図 2.1 の様に左側の要件定義や設計のプロセスと対応させる. 例えば, 図 2.1 では詳細設計には単体テストが配置されている. これは, 詳細設計の正しさを単体テストによって確認し, 単体テストで不具合が見つかった場合は, 左側の対応するプロセスである詳細設計に戻って修正する [6].

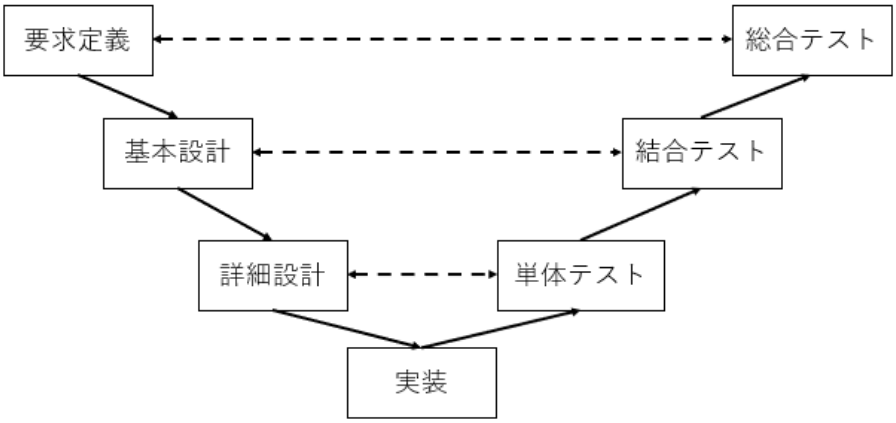

図 2.1　V 字開発モデル

2.3.2　UML(Unified Modeling Language)

UML 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を規定した言語である. UML の用途は, プログラムを書く前に図で考えを整理すことや, チーム開発において図でコミュニケーションをとること, ユーザーや顧客と仕様を検討するなどの場合である [7].

2.3.3　ユースケース図

ユースケース図とは, システムの仕様機能 (ユースケース) と, 外部環境 (アクター) との関連を表す [7]. 一目でシステムの機能やシステム外部と内部の境界を理解することができるため, ユーザーとクライアントとの意識統一を図ることが容易になる. ユースケース図において, ユースケースはシステムの使用機能を楕円の中にユースケース名として記述する. アクターは, 機能を利用するユーザーや, システムが使用するハードウェア, 外部システムを表現する.

2.3.4　シーケンス図

シーケンス図とは, オブジェクトの相互作用を表す相互作用図の一つで, オブジェクト間のメッセージのやりとりを, 時系列に沿って表現する [7].

2.3.5　クラス図

クラス図とは, モデルの静的な構造を表し, システムの構造を表現できる [7]. パッケージ単位の表現や, 全体での表現, または, 機能単位での表現など, 様々な視点で作成することができる.

第3章　SAS-L2認証方法

本章では,SAS-L2 での認証方法について説明する.

3.1　SAS-L2(SimpleAnd Secure password authentication protocol,Light processing version, type 2) の概要

SAS-L2 とは, 高知工科大学の清水明宏教授が提案した処理負荷が特に小さいワンタイムパスワード認証方式 SAS-L の一つである [5]. 表 3.1 から分かるように, 従来の SAS と比較し, 特に被認証者側の処理負荷が小さい認証方式である.

表 3.1　SAS-2 と SAS-L2 の演算回数の比較

	ユーザー			サーバー		
	ハッシュ関数	XOR	加算	ハッシュ関数	XOR	加算
SAS-2	2	3	0	1	2	0
SAS-L2	0	2	2	1	3	2

処理性能の低い IoT 機器において, 極めて小さい処理負荷で暗号鍵の配送が実現できる. バーナム暗号などと組み合わせて暗号系を組むことで, 処理能力の低い IoT 機器へ暗号化通信機能を付加できる [5]. SAS-L2 は, 認証者と被認証者にあらかじめ初回認証情報と初回秘匿情報の登録を行う初期登録処理と, 以降の認証処理から構成される. 認証者と被認証者は, 以降でそれぞれサーバー, ユーザーと定義する.

3.1.1　初期登録処理

SAS-L2 の初期登録処理のフローチャートを図 3.1 に示す.

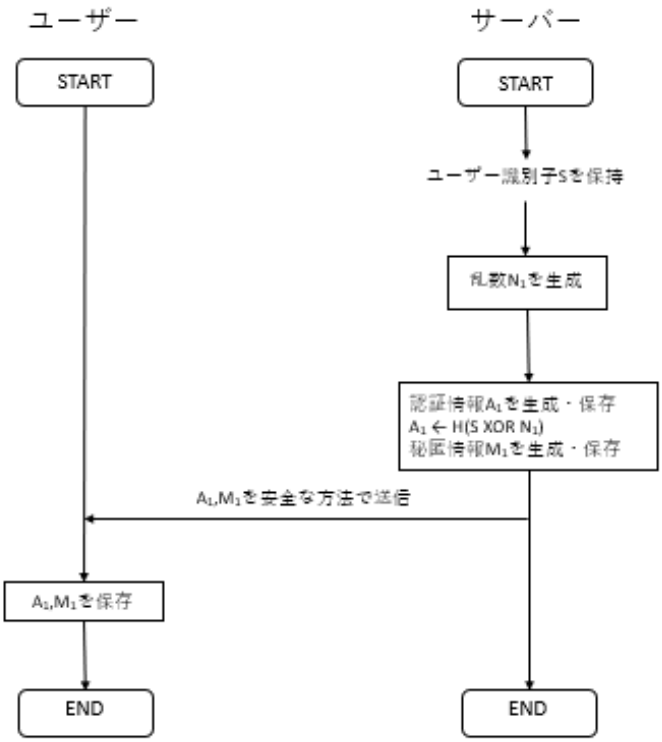

図 3.1　初期登録処理の手順

初期登録は, 以下の手順に沿って処理が行われる.

1. サーバーはユーザー識別子 S を保持する.
2. サーバーは初期認証用の乱数 N_1 を生成する.
3. サーバーは, ユーザー識別子 S と乱数 N_1 の排他的論理和にハッシュ関数を適用し, 初期認証用の認証情報 A_1 を生成・保存する.
4. サーバーは初期認証用の秘匿情報 M_1 を生成・保存する.
5. サーバーは認証情報 A_1 と秘匿情報 M_1 を安全な方法でユーザーに送信する.
6. ユーザーはサーバーから受信した認証情報 A_1 と秘匿情報 M_1 を保存する.

初期登録の終了後, 以降は認証処理が行われる.

3.1.2　認証処理

SAS-L2 の n 回認証時のフローチャートを図 3.2 に示す. N_n,A_n,M_n はそれぞれ n 回目認証用の乱数, 認証情報, 秘匿情報である.

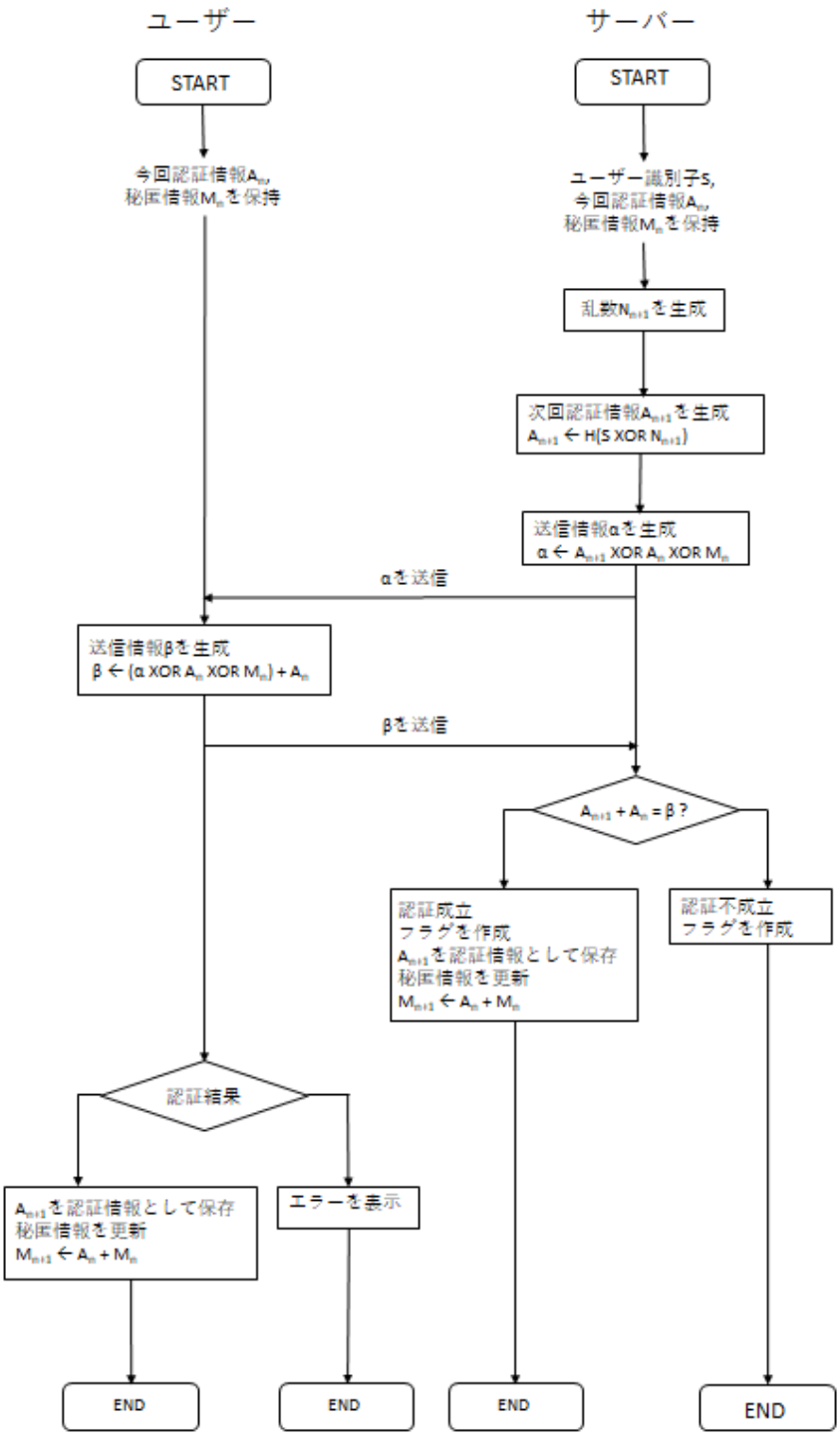

図 3.2　認証手順

認証は, 以下の手順に沿って処理が行われる.

1. サーバーはユーザー識別子 S と認証情報 A_n と秘匿情報 M_n を保持している. ユーザー

は認証情報 A_n と秘匿情報 M_n を保持している.

2. サーバーは次回認証用の乱数 N_{n+1} を生成する.

3. サーバーは, ユーザー識別子 S と乱数 N_{n+1} の排他的論理和にハッシュ関数を適用し, 次回認証用の認証情報 A_{n+1} を生成する.

4. サーバーは認証情報 A_{n+1},A_n と秘匿情報 M_n の排他的論理和を演算し,α を生成する.

5. サーバーは α をユーザーに送信する.

6. ユーザーはサーバーから受信した α と認証情報 A_n, 秘匿情報 M_n の排他的論理和 ($\alpha \oplus A_n \oplus M_n (= A_{n+1})$) と認証情報 A_n の算術加算により,β を生成する.

7. ユーザーは β をサーバーに送信する.

8. サーバーはユーザーから受信した認証情報 A_{n+1} と保持していた A_n の加算を β と比較し, 一致すれば認証成功し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.

9. ユーザーは α と認証情報 A_n, 秘匿情報 M_n の排他的論理和を演算し次回認証用の認証情報 A_{n+1} を生成し保存する. 認証情報 A_n と秘匿情報 M_n の加算を行い, 次回認証用の秘匿情報 M_{n+1} を M_n の代わりに保存する.

以上の処理で 1 回の認証が終了し, 次回以降も認証を行う度に以上の処理を繰り返す.

第4章　センシングデータ通信の暗号化

本章では,SAS-L2 に基づいたデータ通信の暗号化について説明する.

4.1　SAS-L2 に基づいたデータ通信の暗号化

本研究では,SAS-L2 ワンタイムパスワード認証方式に基づいたセンシングデータ通信の暗号化方法を提案し, IoT システムにおいてセンシングデータの暗号化通信を実現させる. 暗号化通信における送信側のセンシングデータの暗号化と受信側のセンシングデータの復号化のアルゴリズムを説明する.

4.1.1　送信側のセンシングデータ暗号化

アルゴリズム 1 は, センシングデータを送信するエッジデバイス側の暗号化アルゴリズムである. n 回目とは, その時点までに行った提案手法による暗号化通信の回数とする. 以降, エッジデバイスはユーザーと定義する.

ユーザーは初めに, アルゴリズム 1 の処理 1 のようにセンシングデータ (SD) と認証情報 A_{n+1},A_n の排他的論理和を演算し γ を生成する. 処理 2 では,γ をサーバーに送信する. 処理 3 では,α をサーバーから受信する. 処理 4 では,α と認証情報 A_{n+1} と秘匿情報 M_{n+1} の排他的論理和を演算し,A_{n+2} の復号化を行う. 処理 5 では, 認証情報 A_{n+1} と秘匿情報 M_{n+1} の算術加算により, 秘匿情報 M_{n+2} を生成する. 処理 6 では, 認証情報 A_n,A_{n+1} と秘匿情報 M_{n+1} を更新する. 処理 1 から処理 6 までを 1 回のエッジデバイス側でのデータ暗号化とし,10 回の繰り返しを終えたら, 処理 7 で認証情報 $A_n \leftarrow A_{n+1}$, 秘匿情報 $M_n \leftarrow M_{n+1}$ として保存する.

Algorithm 1 n 回目のエッジデバイス側でのデータ暗号化

Input: α,SD,n 回目認証情報 A_n,$n+1$ 回目認証情報 A_{n+1},$n+1$ 回目秘匿情報 M_{n+1}

Output: γ

1: $\gamma \leftarrow SD \oplus A_{n+1} \oplus A_n$

2: γ をサーバーに送信

3: α をサーバーから受信

4: $A_{n+2} \leftarrow \alpha \oplus A_{n+1} \oplus M_{n+1}$

5: $M_{n+2} \leftarrow A_{n+1} + M_{n+1}$

6: 認証情報 A_n,A_{n+1} と秘匿情報 M_{n+1} を更新.

$A_n \leftarrow A_{n+1}$

$A_{n+1} \leftarrow A_{n+2}$

$M_{n+1} \leftarrow M_{n+2}$

7: 処理 1 から 6 まで 10 回繰り返した後, 認証情報 $A_n \leftarrow A_{n+1}$, 秘匿情報 $M_n \leftarrow M_{n+1}$ として保存.

4.1.2　受信側のセンシングデータの復号化

アルゴリズム 2 は, センシングデータを受信するサーバー側の復号のアルゴリズムである. n 回目とは, その時点までに行った提案手法による暗号化通信の回数とする. 以降, エッジデバイスはユーザーと定義する.

サーバーは初めに, アルゴリズム 2 の処理 1 のように送信者であるユーザーから, センシングデータ (SD) を暗号化した γ を受信する. 処理 2 では,γ と認証情報 A_{n+1},A_n の排他的論理和を演算し,SD を復号化する. 処理 3 では, 乱数 N_{n+2} を生成し, 処理 4 で乱数 N_{n+2} とユーザー識別子 S の排他的論理和にハッシュ関数を適用することで, 次回認証情報 A_{n+2} を生成する. 処理 5 では, 認証情報 A_{n+2},A_{n+1} と秘匿情報 M_{n+1} の排他的論理和を演算し,α を生成する. 処理 6 では,α をユーザーに送信する. 処理 7 では, 認証情報 A_{n+1} と秘匿情報 M_{n+1} の算術加算により, 秘匿情報 M_{n+2} を生成する. 処理 8 では, 認証情報 A_n,A_{n+1} と秘匿情報 M_{n+1} を更新する. 処理 1 から処理 8 までを 1 回のサーバーの暗号化データの復号とし,10 回の繰り返しを終えたら, 処理 9 で認証情報 $A_n \leftarrow A_{n+1}$, 秘匿情報 $M_n \leftarrow M_{n+1}$ として保存する.

Algorithm 2 n 回目のサーバーの暗号化データの復号

Input: γ, ユーザー識別子 S,n 回目認証情報 A_n,$n+1$ 認証情報 A_{n+1},$n+1$ 秘匿情報 M_{n+1}

Output: α,SD

1: γ をユーザーから受信

2: $SD \leftarrow \gamma \oplus A_{n+1} \oplus A_n$

3: 乱数 N_{n+2} を生成

4: $A_{n+2} \leftarrow H(S \oplus N_{n+2})$

5: $\alpha \leftarrow A_{n+2} \oplus A_{n+1} \oplus M_{n+1}$

6: α をユーザーに送信

7: $M_{n+2} \leftarrow A_{n+1} + M_{n+1}$

8: 認証情報 A_n,A_{n+1} と秘匿情報 M_{n+1} を更新.

$A_n \leftarrow A_{n+1}$

$A_{n+1} \leftarrow A_{n+2}$

$M_{n+1} \leftarrow M_{n+2}$

9: 処理 1 から 8 まで 10 回繰り返した後, 認証情報 $A_n \leftarrow A_{n+1}$, 秘匿情報 $M_n \leftarrow M_{n+1}$ として保存.

4.1.3　提案手法の利点

第 2 章で述べたように, 従来暗号方式としてバーナム暗号がある. バーナム暗号では, 鍵は一度しか使用することができず, 暗号化を行う度に鍵を共有する必要があり, 鍵を共有する毎に鍵が直接ネットワークに流れるという欠点がある. これに対して, 提案手法の SAS-L2 に基いたデータ通信の暗号化方法では, 鍵を共有する際に, 鍵が直接ネットワークに流れないという利点がある. 例えば,1 回目の暗号化通信では, 鍵として認証情報 A_2 と認証情報 A_1 が必要となる. 認証情報 A_1 は,IoT エッジデバイスに初期認証情報として秘匿情報 M_1 と共に書き込まれているとすれば, 認証者が被認証者に認証情報を送信する必要がなくなる. 認証情報 A_2 は, 認証者が被認証者に送信することで共有するが, 認証情報 A_1 と秘匿情報 M_1 を持っていなければ認証情報 A_2 を復号することができない. このように, 提案手法では鍵と認証情報と秘匿情報の排他的論理和を演算して送信することから, 鍵がそのままネットワークに流れることなく鍵を配送できる.

また, 第 2 章で述べた従来の暗号化方式である AES は, 暗号化したいデータを, ブロックに分け, ブロック毎に 4 種類の変換を複数回繰り返すことで暗号化が行われる. このように,AES

などの従来の暗号化方式は暗号化での処理負荷が大きい. これに対して, 提案手法では暗号化を行いたいデータと認証情報 A_n と A_{n+1} との排他的論理和を演算し, 第 3 章で説明した SAS-L2 の認証アルゴリズムと同様に鍵を更新することで暗号化通信ができる. このように, 従来の暗号化方式と比較して提案手法は処理負荷が小さく, 処理性能の低い IoT センシングデバイスへの実装が実現できる.

第5章　SAS-L2を用いたセキュアな組込みシステムの開発

本章では,SAS-L2 を用いたセキュアな組込みシステムについての仕様を述べる.

5.1　SAS-L2 を用いたセキュアな組込みシステムの概要

はじめに,SAS-L2 を利用したセキュアな組込みシステムの概要図を図 5.1 示す.

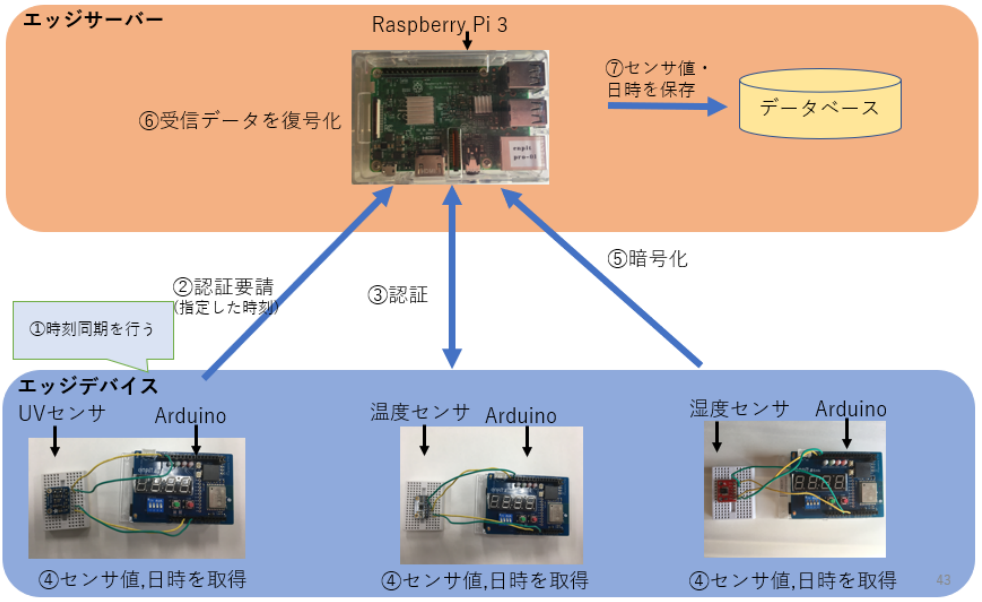

図 5.1　SAS-L2 を利用したセキュアな組込みシステムの概要図

Raspberry Pi をエッジサーバーとし,3 台の Arduino をエッジデバイスとして利用する. 図 5.1 のように, エッジデバイスにはそれぞれ,UV センサ, 温度センサ, 湿度センサを接続している. データベースには, エッジデバイスから収集したデータを保存する. 以降, エッジサーバーをサーバー, エッジデバイスをユーザーと定義する. 図 5.1 に沿って,SAS-L2 を利用したセキュアな組込みシステムの時刻同期, センシングデータ取得, 認証, および暗号化通信の処理の流れを説明する.

1. ユーザーを起動し, 時刻同期を行う.
2. ユーザーはサーバーに対して, 指定した時刻に認証要請を行う.
3. サーバーは認証要請を受信し, 認証要請を送信したユーザーとの SAS-L2 認証を行う.
4. ユーザーは認証完了後, 接続されたセンサからセンシングデータ (センサ値) と, センシングデータを取得した日時を取得する.
5. ユーザーは処理 4 で取得したデータを暗号化しサーバーへ送信する.
6. サーバーはユーザーから受信したデータを復号し, センシングデータと日時を取得する.
7. サーバーは取得した, センシングデータと日時をデータベースに保存する.
8. 処理 4 から処理 7 を 10 回繰り返す.
9. 処理 2 から処理 8 を繰り返す.

以上のように, 指定した時刻になると認証 1 回, 暗号化通信 10 回を行うシステムとなる.

5.2　要件定義

SAS-L2 を利用したセキュアな組込みシステムの要件定義を述べる. 要件定義には, 機能要件と非機能要件がある. 機能要件は, システムで実現すべき機能であり, クライアントから求められる機能のことである. 非機能要件は, 機能要件以外の要件であり, 主に性能やセキュリティ, 環境, 制約を指す.

機能要件

1. 指定した時刻にユーザ―からサーバーにコネクションして認証要請を送信する.
2. サーバーとユーザー間で SAS-L2 による認証を行う.
3. 認証が成功した場合, サーバーとユーザー間で SAS-L2 に基づいた暗号化通信を行う.
4. 認証が失敗した場合, コネクションを切断してシステム概要で述べた処理 1 からやり直す.
5. サーバーは受信データを復号してセンシングデータと日時をデータベースへ保存する.

6. 暗号化通信終了後, サーバーはコネクションを切断し, ユーザーからの認証要請待ち状態となる.

非機能要件

1. 複数のユーザーは同時刻に認証要請を送信するので, 一台の認証が終了するまで, その他のユーザーは待機状態になる.
2. サーバーは認証結果をユーザーに送信する.
3. サーバーとユーザーは認証結果を表示する.
4. ユーザーは, シリアルモニタに取得したセンシングデータとセンシングデータの取得日時を表示する.
5. ユーザーは起動後,1 度時刻同期を行う.
6. サーバーは暗号化通信の際,5 秒以上データを受信できなければ, コネクションを切断し, 認証要請受信の待機状態となる.
7. 指定した時刻毎に 3 つのユーザーとの通信を 10 秒以内に終わらせる.
8. サーバーはセンシングデータと取得日時を保存する際に, ユーザーごとのテーブルに分けて保存する.
9. 1 度の認証につき,10 回の暗号化通信を行う.
10. ユーザーで, 何らかのエラーが発生した場合, 赤 LED を点滅させる.

第6章　SAS-L2を用いたセキュアな組込みシステムの設計

本章では,SAS-L2 を利用したセキュアな組込みシステムについての設計を述べる. システム開発にあたり, 基本設計の作成と役割分担, スケジュールの決定, 詳細設計とテスト項目を作成した. 基本設計と詳細設計は UML を用いて行った.

6.1　基本設計

基本設計では, ユースケース図とクラス図とシーケンス図を作成した.

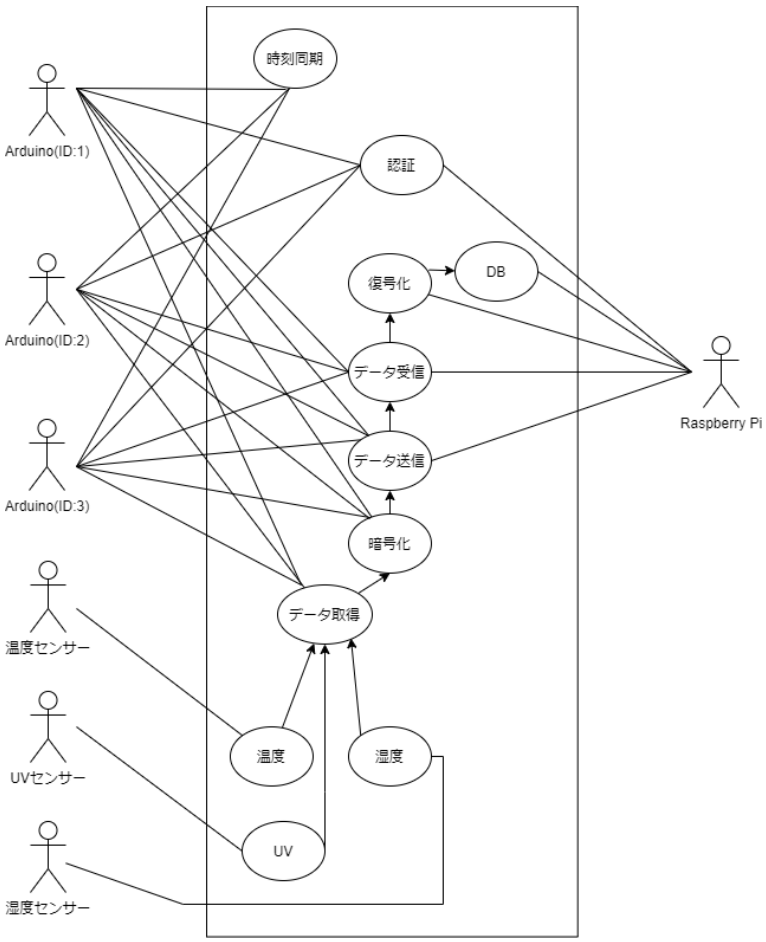

図 6.1　ユースケース図

図 6.1 のユースケース図では, システム内の基本機能を視覚的に図示している. ユースケース図の作成により, システムの機能の洗い出しや役割分担を決定した.

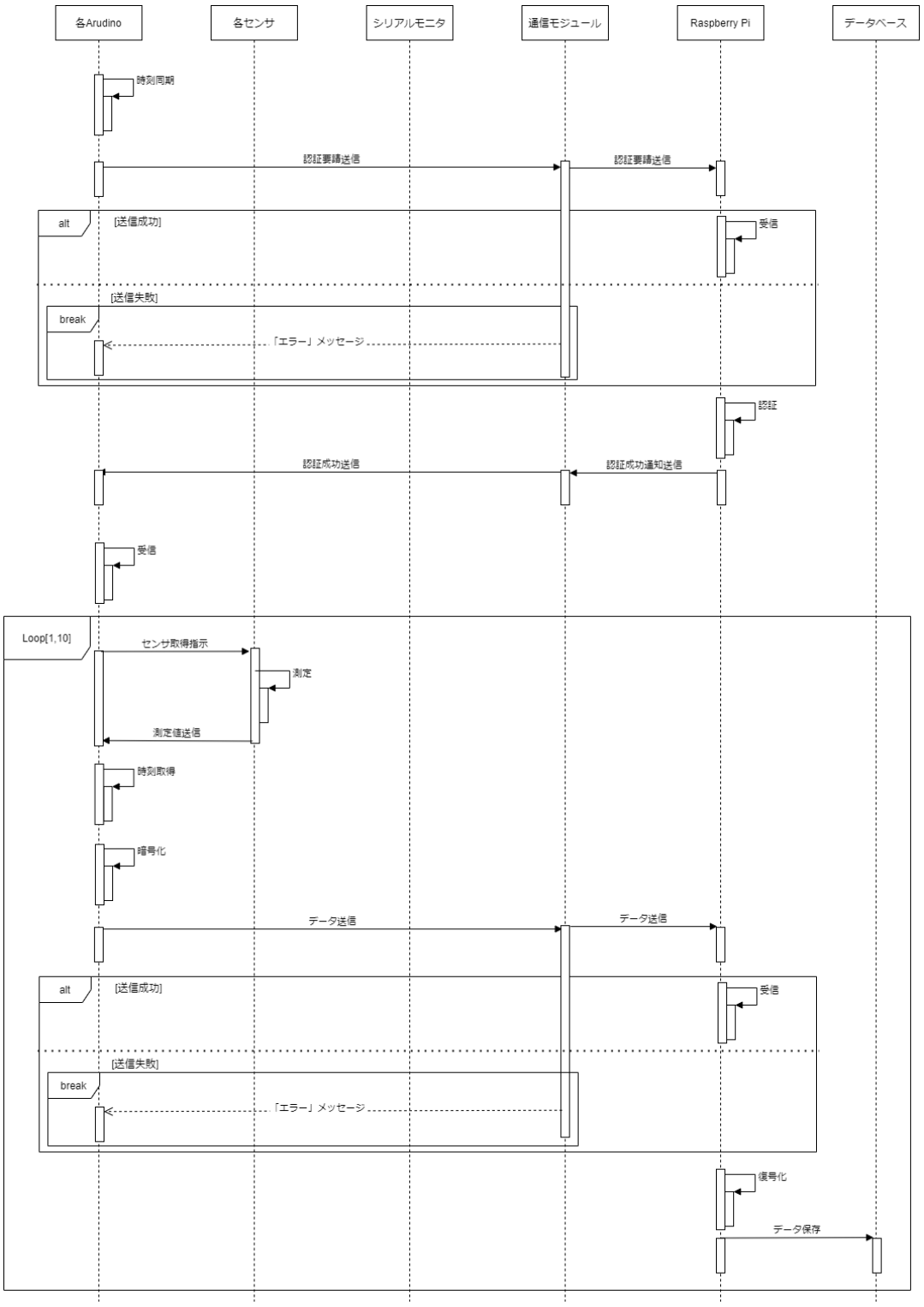

図 6.2　シーケンス図

図 6.2 のシーケンス図では, クラス間で行われる処理を時系列で表している. 初めに,Arduino が時刻同期を行う. 次に,Arduino から通信モジュールを経由して Raspbery Pi へ認証要請を

行う. Arduino が送信成功すれば Raspbery Pi は認証要請を受信し, 送信失敗した場合は通信モジュールからエラーメッセージが送信され, 処理が中断される. その後,Raspbery Pi により認証を行い, 認証成功通知を Arduino へ送信する. 認証処理終了後,Arduino は接続されているセンサからセンシングデータと, センシングデータを取得した日時を取得し, 暗号化を行う. Arduino は暗号化したデータを Raspbery Pi へ送信し, 送信成功すれば Raspbery Pi はデータを受信し, 送信失敗した場合は通信モジュールからエラーメッセージが送信され, 処理が中断される. 最後に Raspbery Pi が受信したデータを復号し, 得られたセンシングデータとセンシングデータの取得日時をデータベースへ保存する. 図 6.2 に示されている Loop[] はループと言い,[] 内で示された範囲での繰り返し処理が行われる. ここでは Loop[1,10] であることから,10 回の繰り返し処理が行われる.

図 6.3　クラス図

図 6.3 のクラス図では, システムの静的な構造と関係性を視覚的に表している. 温度センサ, 湿度センサ,UV センサ,Arduino, モニタ, 通信モジュール,Raspbery Pi, データベースをクラスとしている. 関連のあるクラス間での多重度も示している.

6.2　役割分担

図 6.1 のユースケース図から, 役割分担を行った. Arduino の時刻同期とセンサからのセンシングデータ取得, データを暗号化し Raspbery Pi へ送信する機能を浅野が担当した. Raspbery Pi の Arduino からのデータ受信とデータの復号化, データベースへの保存を内山田が担当した. 認証については, 被認証側を浅野, 認証側を内山田が担当した.

6.3　スケジュール

スケジュール管理は, ガントチャートで行った. 図 6.4 のようにガントチャートを EXCEL で作成し, チーム内で共有しながら進捗を管理した.

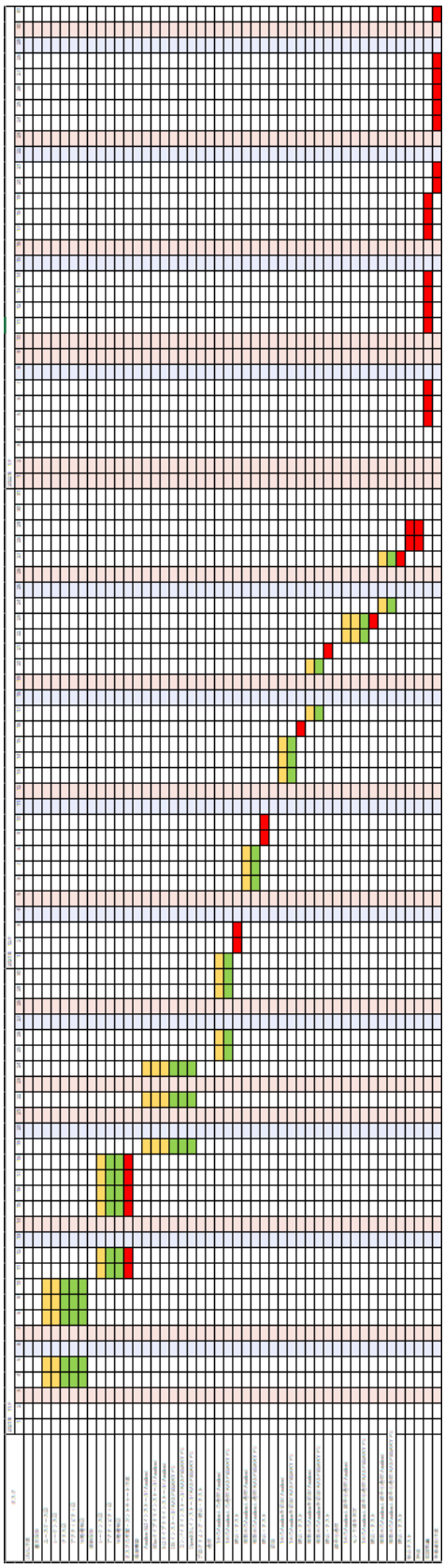

図 6.4　ガントチャート

6.4　詳細設計

担当する機能についての詳細設計を UML を用いて行い, 時間同期, 認証における被認証者の処理, センシングデータの取得, 暗号化通信についてのシーケンス図を作成した. また, 暗号化通信における通信フォーマットの定義も行った.

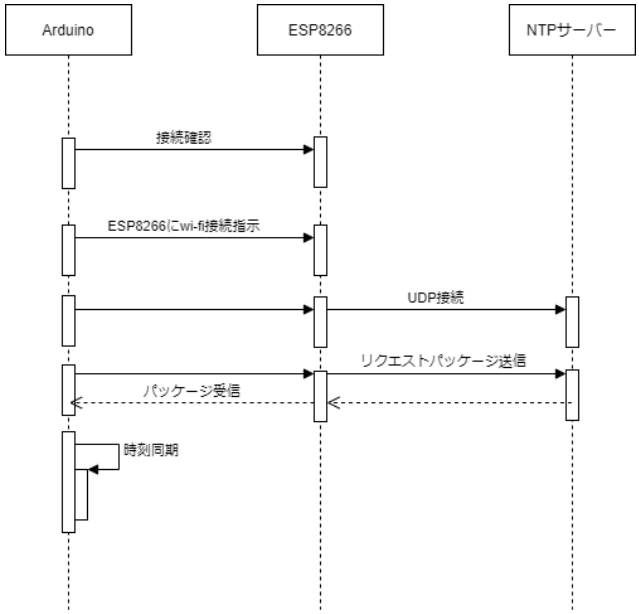

図 6.5　時間同期におけるシーケンス図

図 6.5 のシーケンス図では,Arduino における時刻同期に関するクラス間で行われる処理を時系列で表している. 初めに,Arduino が通信モジュールである ESP8266 との接続を行う. 次に,Arduino は通信モジュールを介して Wi-Fi との接続を行う. 日時を取得するために,Arduino から NTP サーバーに UDP 接続し, リクエストパッケージを送信する. そして,NTP サーバーから受信したパッケージより, 日時を取得して時刻同期を行う. 時刻同期の完了後,SAS-L2 による認証・暗号化通信が行われる.

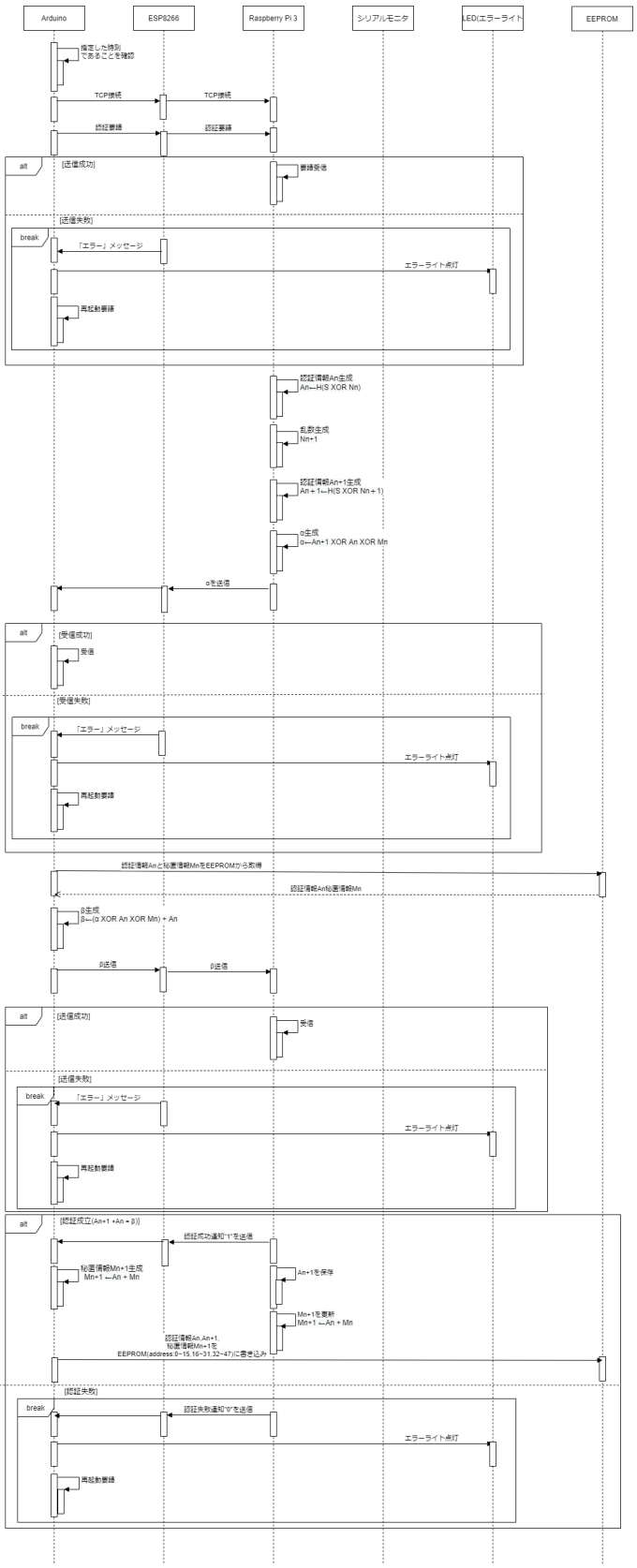

図 6.6　認証におけるシーケンス図

図 6.6 のシーケンス図では,Arduino における認証に関するクラス間で行われる処理を時系列で表している. 初めに,Arduino が指定した時刻になると,Raspberry Pi に対して TCP 接続を行い, 認証要請を送信する. 認証要請が送信成功すれば以降の処理に進み, 送信失敗であれば通信モジュールである ESP8266 からエラーメッセージが Arduino に対して送信され, Arduino の赤 LED が点滅し, 利用者に対して再起動するように求める. 次に,Raspberry Pi から α を受信する. この時, 受信が失敗すれば, Arduino の赤 LED が点滅し, 利用者に対して再起動するように求める. そして,Arduino の EEPROM に書き込まれている認証情報と秘匿情報を読み出して, β を生成し,Raspberry Pi へ送信する. この時, 送信が失敗すれば, Arduino の赤 LED が点滅し, 利用者に対して再起動するように求める. 次に, 認証結果を Raspberry Pi から受信する. 認証成功なら”1”を受信して秘匿情報を生成し, 更新した認証情報と秘匿情報を EEPROM に書き込む. 認証失敗なら”0”を受信して,Arduino の赤 LED が点滅し, 利用者に対して再起動するように求める.

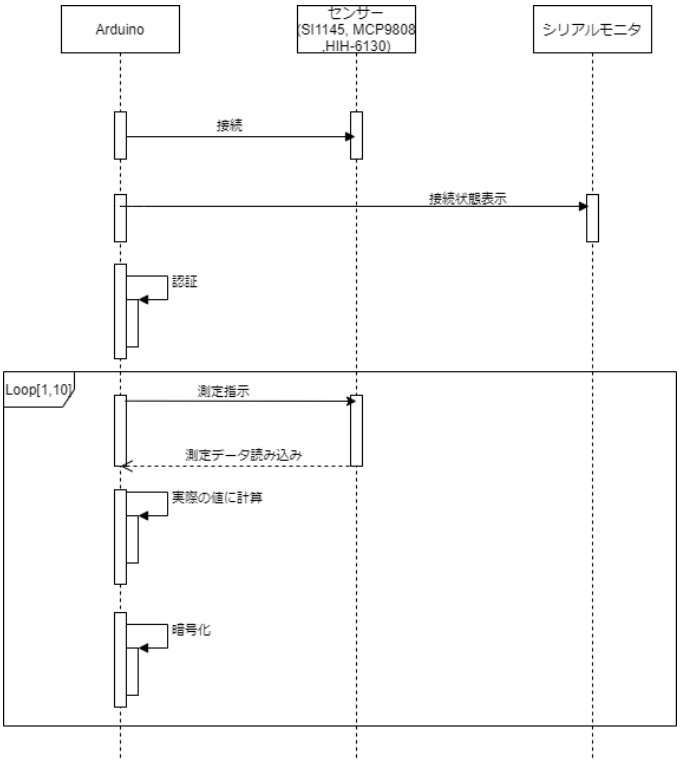

図 6.7　センシングデータ取得におけるシーケンス図

図 6.7 のシーケンス図では, Arduino におけるセンシングデータ取得に関するクラス間で行われる処理を時系列で表している. 初めに,Arduino は接続されているセンサと接続を行い, 確認のためにシリアルモニタへセンサとの接続状態を表示する. 次に, 認証完了後,Arduino は

センサへ測定指示を出して測定値を取得する. 取得した測定値を実際の値へと変換するために計算を行う. 計算後の値をセンシングデータとし, そのセンシングデータを利用して暗号化通信を行う. 図 6.7 に書かれている Loop[1,10] の部分では,10 回の繰り返し処理が行われる.

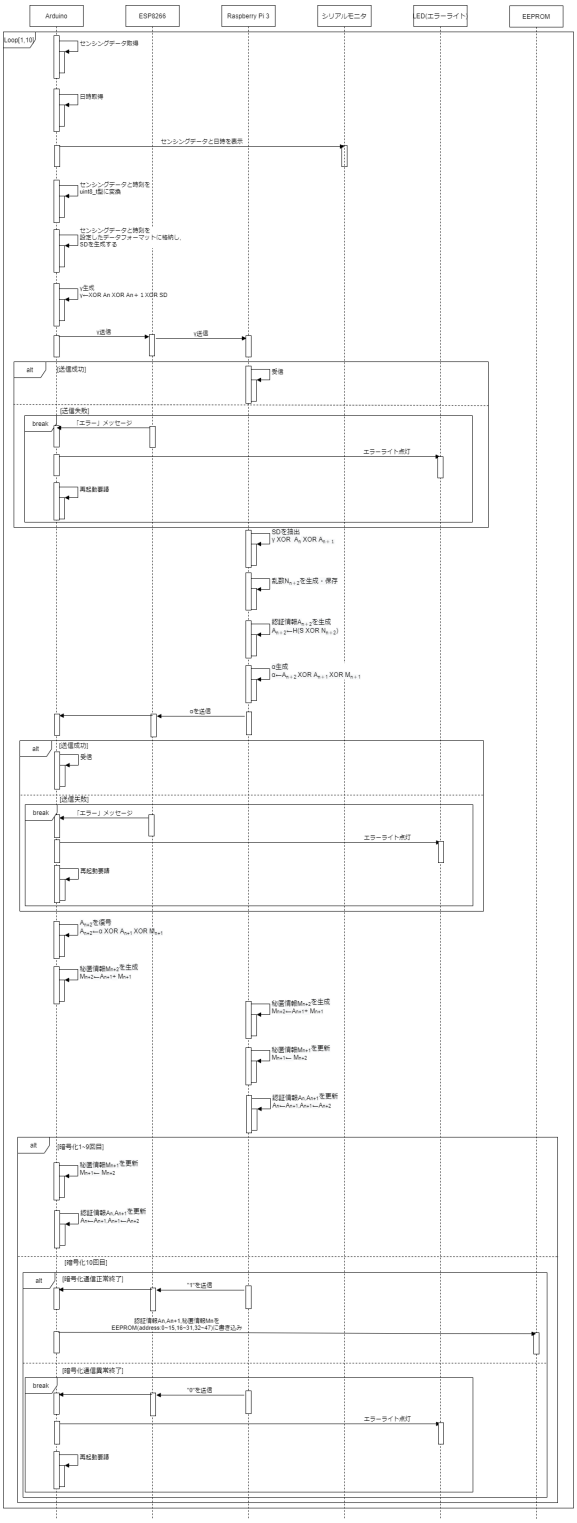

図 6.8　暗号化通信におけるシーケンス図

図 6.8 のシーケンス図では, Arduino における暗号化通信に関するクラス間で行われる処理を時系列で表している. 初めに,Arduino はセンシングデータとセンシングデータを取得した日時を取得する. そして, 取得したセンシングデータと日時をシリアルモニタに表示する. 次に, 取得したセンシングデータと日時を unsigned char 型へ変換して, データフォーマットに格納し SD を生成する. そして, 生成した SD を利用して γ を生成し,Raspberry Pi へ送信する. この時, 送信が失敗すれば, Arduino の赤 LED が点滅し, 利用者に対して再起動するように求める. 次に,Raspberry Pi から α を受信する. この時, 受信が失敗すれば, Arduino の赤 LED が点滅し, 利用者に対して再起動するように求める. 受信した α から次回認証情報を復号し, その後, 次回秘匿情報を生成する. 最後に, 認証情報と秘匿情報の更新を行う. 暗号化通信が 1 から 9 回目の場合は, 認証情報と秘匿情報を格納している配列を更新する. 暗号化通信が 10 回目の場合, 正常に暗号化通信が終了した時は Raspberry Pi から”1”を受信し, 更新した認証情報と秘匿情報を EEPROM に書き込む. 暗号化通信が正常に終了しなかった時は,Raspberry Pi から”0”を受信して, Arduino の赤 LED が点滅し, 利用者に対して再起動するように求める.

6.4.1　データフォーマット

暗号化通信の際に,$\gamma \leftarrow SD \oplus A_{n+1} \oplus A_n$ の演算によって γ を生成する. γ の生成には SD が必要となり, この SD にはセンシングデータとセンシングデータを取得した日時が格納されている. Arduino と Raspberry Pi 間では, 基本的に unsigned char 型の 128bit 配列でデータの送受信が行われるため, センシングデータと日時を unsigned char 型の 16byte(128bit) 配列 SD に格納するためにデータフォーマットを作成した. 作成したデータフォーマットを図 6.9 に示す. まず, センシングデータは float 型で取得される.float 型は 32bit であることから, センシングデータのビット列は配列 SD[0] から SD[3] に格納する. 次に, 日時の内容は年, 月, 日, 時, 分, 秒である. 年は int 型の 16bit で取得され, 月, 日, 時, 分, 秒はそれぞれ int 型の 8bit で取得される. このことから, 年のビット列は配列 SD[4] から SD[5] に格納し, 月のビット列は配列 SD[6], 日のビット列は配列 SD[7], 時のビット列は配列 SD[8], 分のビット列は配列 SD[9], 秒のビット列は配列 SD[10] に格納される. 残りの配列 SD[11] から SD[15] には乱数を設定し格納する.

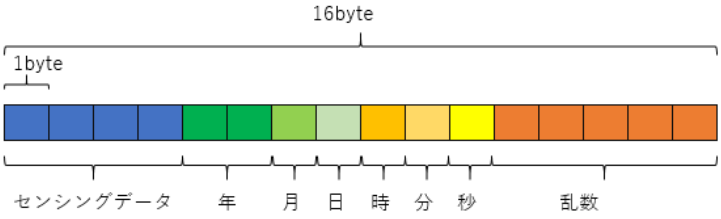

図 6.9　センシングデータと取得日時を格納する際のデータフォーマット

6.5　テスト項目の作成

本節では, 本研究で作成した単体テスト項目と結合テスト項目, 総合テスト項目について説明する.

6.5.1　単体テスト項目

単体テストではＶ字開発モデルに従って, 詳細設計を参考し表 6.1 のテスト項目を作成した.

表 6.1　単体テスト項目

機能枠	確認内容
通信モジュールとの接続	Arduino と ESP8266 を接続した時,ESP8266 との接続が確認できたら, シリアルモニタに”ESP8266 OK”と表示される.
	Arduino と ESP8266 を接続しなければ,ESP8266 との接続が確認できず, 赤 LED が点滅する.
Wi-Fi 接続	Wi-Fi との接続ができれば, シリアルモニタに”connect success”と 表示される.
UDP 接続	IP アドレス”ntp.nict.jp”, ポート番号”123”へ UDP 接続が成功すれば, 時刻同期処理が開始される.
TCP 接続	IP アドレス”192.168.2.110”、ポート番号”49152”へ TCP 接続が成功すれば, シリアルモニタに”create tcp ok” と表示される.
センサ	Arduino とセンサが接続している.
	Arduino とセンサが接続されていなければ赤 LED が点滅する.
	センシングデータを取得する.
時刻取得	センシングデータを取得した時点の年, 月, 日, 時, 分, 秒を取得する.
認証の際の β の演算	$\alpha \oplus A_n \oplus M_n$ の演算結果が正しい. 演算結果は 128bit である.
	$\beta \leftarrow (\alpha \oplus A_n \oplus M_n) + A_n$ の演算結果が正しい. 演算結果は 128bit である.
認証の際の 次回秘匿情報 の計算	$M_{n+1} \leftarrow A_n + M_n$ の演算が正しい. 演算結果は 128bit である.
認証成功時の 認証情報 A_n,A_{n+1}, 秘匿情報 M_n の更新	n 回目認証成功時の A_n,A_{n+1},M_{n+1} を EEPROM のアドレス 0-15,16-31,32-47 にそれぞれ書き込む.
SD の生成	float 型 (32bit) センサ値と,int 型 (16bit) の年,$int8_t$ 型 (8bit) の月, 日, 時, 分, 秒を unsigned char 型の 128bit 配列に格納される.
暗号化通信の際の γ の演算	$\gamma \leftarrow SD \oplus A_{n+1} \oplus A_n$ の演算結果が正しい. 演算結果は 128bit である.
暗号化通信の際の 認証情報 A_{n+2} の演算	$A_{n+2} \leftarrow \alpha \oplus A_{n+1} \oplus M_{n+1}$ の演算結果が正しい. 演算結果は 128bit である.
暗号化通信の際の 次回秘匿情報の計算	$M_{n+2} \leftarrow A_{n+1} + M_{n+1}$ の演算が正しい. 演算結果は 128bit である.
暗号化通信の際の 認証情報, 秘匿情報の更新	10 回目暗号化終了時の A_n,A_{n+1},M_{n+1} を EEPROM の アドレス 0-15,16-31,32-47 それぞれ書き込む.

6.5.2　結合テスト項目

結合テストでは V 字開発モデルに従って, 基本設計を参考し表 6.2 のテスト項目を作成した. Arduino はユーザー,Raspberry Pi はサーバーと定義する.

表 6.2　結合テスト項目

機能枠	確認内容
認証	認証要請の送受信を行う.
	認証の際,α の送受信を行い, サーバーが送信したデータと, ユーザーが受信したデータが一致している.
	認証の際,β の送受信を行い, ユーザーが送信したデータと, サーバーが受信したデータが一致している.
	認証結果の送受信を行う.
	ユーザーで演算した認証情報 A_{n+1}, 秘匿情報 M_{n+1} と, サーバーで演算した認証情報 A_{n+1}, 秘匿情報 M_{n+1} が一致している.
暗号化通信	γ の送受信を行い, ユーザーが送信したデータと, サーバーが受信したデータが一致している.
	ユーザーが取得したセンシングデータと日時が, データベースに保存されたセンシングデータと日時と一致している.
	暗号化通信の際,α の送受信を行い, サーバーが送信したデータと, ユーザーが受信したデータが一致している.
	暗号化通信終了後, ユーザーとサーバーがそれぞれ演算を行った認証情報 A_n と秘匿情報 M_n が一致している.

6.5.3　総合テスト項目

総合テストでは V 字開発モデルに従って, 要件定義を参考し表 6.3 のテスト項目を作成した. Arduino はユーザー,Raspberry Pi はサーバーと定義する.

表 6.3　総合テスト項目

機能枠	確認内容
認証	ユーザーは起動後,1 度時刻同期を行う.
	ユーザーが指定した時間に認証要請を送信する.
	複数のユーザーは同時刻に認証要請を送信するので, 一台の認証が終了するまで, その他のユーザーは待機状態になる.
	ユーザーが認証処理を行っている.
	ユーザーが認証結果をモニタに表示している.
	認証が成功した場合, ユーザーが SAS-L2 に基づいた暗号化通信を行う.
	認証が失敗した場合, ユーザーはコネクションを切断して認証要請送信から再度開始する.
	サーバーが起動したら, 受信待ち状態になる.
	サーバーが認証要請受信後, 認証処理を行っている.
	サーバーが認証結果をモニタに表示している.
	認証が成功した場合, サーバーが SAS-L2 に基づいた暗号化通信を行う.
	認証が失敗した場合, サーバーはコネクションを切断して認証要請の待ち状態となる.
暗号化通信	ユーザーはセンシングデータを取得する.
	ユーザーは, シリアルモニタに取得したセンシングデータとセンシングデータ取得日時を表示する.
	ユーザーはセンシングデータと日時を暗号化してサーバーへ送信する.
	ユーザーは暗号化通信を 10 回行う.
	ユーザーは暗号化通信後, コネクションを切断する.
	サーバーはセンシングデータと日時を復号する.
	サーバーはセンシングデータと取得日時を保存する際に, ユーザーごとのテーブルに分けて保存する.
	サーバーは暗号化通信を 10 回行う.
	サーバーは暗号化通信後, コネクションを切断し認証要請待ち状態となる.
通信	ユーザーで, 何らかのエラーが発生した場合, 赤 LED を点滅させる.
	サーバーは 5 秒以上データを受信できなければ, コネクションを切断して認証要請受信の待機状態となる.
	3 台のユーザーが同時刻に認証要請を送信した場合,3 台のユーザーとの通信が 10 秒以内に完了する.

第7章　検証

本章では,SAS-L2 を利用したセキュアな組込みシステムをシステム設計を基に実装し, 検証を行う. 使用端末, 使用センサ, 開発環境, テスト結果について述べる.

7.1　使用端末

本研究では, エッジデバイスとして Arduino Leonardo を利用し,3 台使用している. Arduino Leonardo とは,Microchip Technology 社の ATmega32U4 を基に作られたマイクロコントローラーボードである.

7.2　使用センサ

本研究で使用したセンサについて, 以下の表 7.1 にまとめる.

表 7.1　使用センサ

UV 指数センサ	SI1145
温度センサ	MCP9808
湿度センサ	HIH-6130

7.3　開発環境

本研究の開発環境について, 以下の表 7.2 にまとめる.

表 7.2　開発環境

OS	windows10
開発環境	Arduino IDE
使用言語	C/C++
通信モジュール	ESP-WROOM-02

7.4　テスト

単体テスト, 結合テスト, 総合テストの順番でテストを行い, テスト結果について表と図を用いてまとめた. 結合テストと総合テストでは,Arduino が認証要請を行う際の指定時刻を毎分 0 秒としてテストを行った.

7.4.1　単体テスト

単体テストでは詳細設計を参考してテストを行った. 単体テスト項目と結果を表 7.3, 表 7.4 に示す. 通信モジュールである ESP-WROOM-02 は, 以降で ESP8266 と定義する.

表 7.3　単体テスト前半

機能枠	確認内容	結果	確認日	確認者	備考欄
通信モジュールとの接続	Arduino と ESP8266 を接続した時,ESP8266 との接続が確認できたら, シリアルモニタに ”ESP8266OK”と表示される.	○	2022/1/10	浅野	
	Arduino と ESP8266 を接続しなければ, ESP8266 との接続が確認できず, 赤 LED が点滅する.	○	2022/1/10	浅野	
Wi-Fi 接続	Wi-Fi との接続ができれば, シリアルモニタに”connect success”と 表示される.	○	2022/1/10	浅野	
UDP 接続	IP アドレス”ntp.nict.jp”, ポート番号”123” へ UDP 接続が成功すれば, 時刻同期処理が 開始される.	○	2022/1/10	浅野	
TCP 接続	IP アドレス”192.168.2.110”, ポート番号 ”49152”へ TCP 接続が成功すれば, シリアルモニタに”create tcp ok”と表示 される.	○	2022/1/10	浅野	
センサ	Arduino とセンサが接続している.	○	2022/1/10	浅野	
	Arduino とセンサが接続されていなければ, 赤 LED が点滅する.	○	2022/1/10	浅野	
	センサ値を取得する.	○	2022/1/10	浅野	
時刻取得	センサ値を取得した時点の年, 月, 日, 時, 分, 秒を取得する.	○	2022/1/10	浅野	
認証の際の β の演算	$A_{n+1} \leftarrow \alpha \oplus A_n \oplus M_n$ の演算 結果が正しい. 演算結果は 128bit である.	○	2022/1/10	浅野	図 7.1 図 7.2 図 7.3 図 7.4
	$\beta \leftarrow A_{n+1} + A_n$ の 演算結果が正しい. 演算結果は 128bit である.	○	2022/1/10	浅野	図 7.2 図 7.4 図 7.5
認証の際の次回秘匿情報の計算	$M_{n+1} \leftarrow A_n + M_n$ の演算が正しい. 演算結果は 128bit である.	○	2022/1/10	浅野	図 7.2 図 7.3 図 7.6
認証成功時の認証情報 A_n,A_{n+1}, 秘匿情報 M_n の更新	n 回目認証成功時の A_n,A_{n+1},M_{n+1} を EEPROM のアドレス 0-15,16-31,32-47 に それぞれ書き込む.	○	2022/1/10	浅野	
SD の生成	float 型 (32bit) センサ値と,int 型 (16bit) の年,int8$_t$ 型 (8bit) の月, 日, 時, 分, 秒を unsigned char 型の 128bit 配列に格納 される.	○	2022/1/10	浅野	

表 7.4　単体テスト後半

機能枠	確認内容	結果	確認日	確認者	備考欄
暗号化通信の際の γ の演算	$\gamma \leftarrow SD \oplus A_{n+1} \oplus A_n$ の演算結果が正しい. 演算結果は 128bit である.	○	2022/1/10	浅野	図 7.7 図 7.4 図 7.2 図 7.8
暗号化通信の際の認証情報 A_{n+2} の演算	$A_{n+2} \leftarrow \alpha \oplus A_{n+1} \oplus M_{n+1}$ の演算結果が正しい. 演算結果は 128bi t である.	○	2022/1/10	浅野	図 7.9 図 7.4 図 7.6 図 7.10
暗号化通信の際の次回秘匿情報の計算	$M_{n+2} \leftarrow A_{n+1} + M_{n+1}$ の演算が正しい. 演算結果は 128bit である.	○	2022/1/10	浅野	図 7.4 図 7.6 図 7.11
暗号化通信の際の認証情報, 秘匿情報の更新	10 回目暗号化終了時の A_n,A_{n+1},M_{n+1} を EEPROM のアドレス 0-15,16-31,32-47 それぞれ書き込む.	○	2022/1/10	浅野	

認証の際に Arduino が生成する β の演算は,$A_{n+1} \leftarrow \alpha \oplus A_n \oplus M_n$,$\beta \leftarrow A_{n+1} + A_n$ と書ける.$A_{n+1} \leftarrow \alpha \oplus A_n \oplus M_n$ の演算は, シリアルモニタに表示させたビット列である図 7.1 と図 7.2, 図 7.3 の 3 つの排他的論理和を演算し, その演算結果が図 7.4 になったことから, 正しく演算されたと分かる. $\beta \leftarrow A_{n+1} + A_n$ の演算は, シリアルモニタに表示させたビット列である図 7.4 と図 7.2 の算術加算の結果が図 7.5 になったことから, 正しく演算されたと分かる.
認証の際に行う次回秘匿情報の演算は $M_{n+1} \leftarrow A_n + M_n$ であり, シリアルモニタに表示させたビット列である図 7.2 と図 7.3 の算術加算の結果が図 7.6 になったことから, 正しく演算されたと分かる.

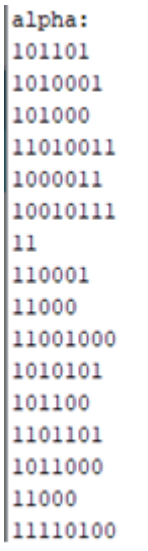

図 7.1　認証の際の α

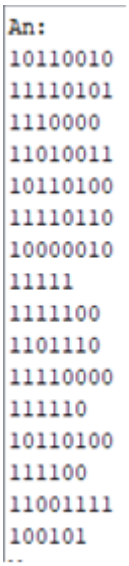

図 7.2　認証情報 A_n

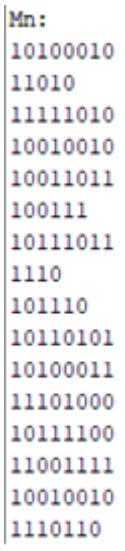

図 7.3　秘匿情報 M_n

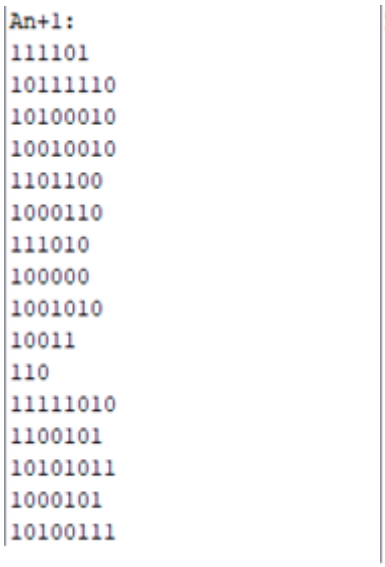

図 7.4　認証情報 A_{n+1}

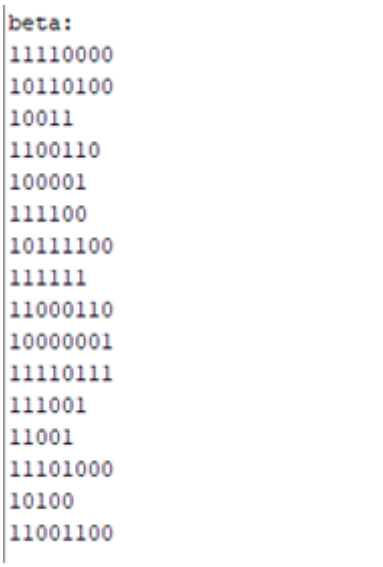

図 7.5　β

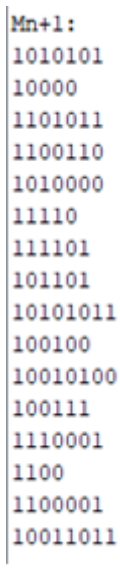

図 7.6　秘匿情報 M_{n+1}

暗号化の際に生成する γ の演算は $\gamma \leftarrow SD \oplus A_{n+1} \oplus A_n$ である. $\gamma \leftarrow SD \oplus A_{n+1} \oplus A_n$ の演算は, シリアルモニタに表示させたビット列である図 7.7 と図 7.4, 図 7.2 の 3 つの排他的論理和を演算し, その演算結果が図 7.8 になったことから, 正しく演算されたと分かる.

暗号化の際に生成する認証情報の演算は $A_{n+2} \leftarrow \alpha \oplus A_{n+1} \oplus M_{n+1}$ であり,$A_{n+2} \leftarrow \alpha \oplus A_{n+1} \oplus M_{n+1}$ の演算は, シリアルモニタに表示させたビット列である図 7.9 と図 7.4, 図 7.6 の 3 つの排他的論理和の結果が図 7.10 になったことから, 正しく演算されたと分かる.

暗号化の際に生成する次回秘匿情報の演算は $M_{n+2} \leftarrow A_{n+1} + M_{n+1}$ であり, シリアルモニタに表示させたビット列である図 7.4 と図 7.6 の算術加算の結果が図 7.11 になったことから, 正しく演算されたと分かる. 図 7.11 では,$M_{n+1} \leftarrow M_{n+2}$ で秘匿情報 M_{n+1} に更新しているため, 演算結果は M_{n+1} としてモニタに表示されている.

```
uint8_t format
111100
10100011
11010111
1010
111
11100110
10
111
10101
11
101000
11010010
100000
10011100
1101101
1011011
```

図 7.7　SD

```
ganma:
10110011
11101000
101
1001011
11011111
1010110
10111010
111000
100011
1111110
11011110
10110
11110001
1011
11100111
11011001
```

図 7.8　γ

```
alpha:
100000
1111
100110
1110000
11010011
1001110
1100011
11010001
10111011
1111011
101010
1000011
10000001
1001110
10011100
11001010
```

図 7.9　暗号化通信の際の α

```
An+2:
1001000
10100001
11101111
10000100
11101111
10110
1100100
11011100
1011010
1001100
10111000
10011110
10010101
11101001
10111000
11110110
```

図 7.10　認証情報 A_{n+2}

```
M+1:
10010010
11001111
1101
11111000
10111100
1100100
1110111
1001101
11110101
110111
10011011
100001
11010110
10110111
10100111
1000010
```

図 7.11　秘匿情報 M_{n+2}

7.4.2　結合テスト

結合テストでは基本設計を参考してテストを行った. 結合テスト項目と結果を表 7.5 に示す. Arduino はユーザー,Raspberry Pi はサーバーと定義する.

表 7.5　結合テスト

機能枠	確認内容	結果	確認日	確認者	備考欄
認証	認証要請の送受信を行う.	○	2022/1/14	浅野 内山田	
	認証の際,α の送受信を行い, サーバーが送信したデータと, ユーザーが受信したデータが一致している.	○	2022/1/14	浅野 内山田	図 7.12 図 7.13
	認証の際,β の送受信を行い, ユーザーが送信したデータと, サーバーが受信したデータが一致している.	○	2022/1/14	浅野 内山田	図 7.14 図 7.15
	認証結果の送受信を行う.	○	2022/1/14	浅野 内山田	
	ユーザーで演算した 認証情報 A_{n+1}, 秘匿情報 M_{n+1} と, サーバーで演算した 認証情報 A_{n+1}, 秘匿情報 M_{n+1} が 一致している.	○	2022/1/14	浅野 内山田	
暗号化通信	γ の送受信を行い, ユーザーが送信したデータと, サーバーが受信したデータが一致している.	○	2022/1/14	浅野 内山田	図 7.16 図 7.17
	ユーザーが取得したセンシングデータと日時が, データベースに保存されたセンシングデータと日時と一致している.	○	2022/1/14	浅野 内山田	図 7.18 図 7.19
	暗号化通信の際,α の送受信を行い, サーバーが送信したデータと, ユーザーが受信したデータが一致している.	○	2022/1/14	浅野 内山田	図 7.20 図 7.21
	暗号化通信終了後, ユーザーとサーバーがそれぞれ演算を行った認証情報 A_n と秘匿情報 M_n が一致している.	○	2022/1/14	浅野 内山田	

サーバーが送信した α は図 7.12 であり, ユーザーが受信した α は図 7.13 であることから, α の値が一致しており, 送受信が正しく行われたことが分かる.

```
//////////////  α ← An+1 XOR An XOR Mn  //////////////
alpha:  12 174 98 76 233 160 251 84 207 57 3 57 146 167 246 23

//////////////  server --- (alpha) ---> client  //////////////
send data
send alpha.
```

図 7.12　認証の際にサーバーが送信した α

```
Received α ok
alpha:
12,174,98,76,233,160,251,84,207,57,3,57,146,167,246,23,
```

図 7.13　認証の際にユーザーが受信した α

ユーザーが送信した β は図 7.14 であり, ユーザーが受信した β は図 7.15 であることから, β の値が一致しており, 送受信が正しく行われたことが分かる.

```
beta:
234,146,190,233,206,11,197,204,167,148,166,38,245,211,148,249,
beta send ok
```

図 7.14　認証の際にユーザーが送信した β

```
//////////// client --- (beta) ---> server ////////////
recieve data
beta:  234 146 190 233 206 11 197 204 167 148 166 38 245 211 148 249
```

図 7.15　認証の際にサーバーが受信した β

ユーザーが送信した γ は図 7.16 であり, ユーザーが受信した γ は図 7.17 であることから, γ の値が一致しており, 送受信が正しく行われたことが分かる.

```
ganma:
41,163,233,195,182,5,176,129,71,171,90,54,169,27,103,174,
send(ganma) ok
```

図 7.16　暗号化通信の際にユーザーが送信した γ

```
//////////// client --- (gamma) ---> server ////////////
recieve data
gamma: 41 163 233 195 182 5 176 129 71 171 90 54 169 27 103 174
```

図 7.17　暗号化通信の際にサーバーが受信した γ

ある時点でユーザーが取得したセンシングデータと日時は図 7.18 であり, サーバーがデータベースに保存したセンシングデータと日時は図 7.19 のデータベースに含まれていることから, センシングデータと日時が正しくデータベースに保存されたことが分かる.

```
====================
UV: 0.04
2022/1/13/16/32/2
```

図 7.18　ユーザーが取得したセンシングデータと日時

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

図 7.19　データベースに保存されたセンシングデータと日時

暗号化通信の際にサーバーが送信した α は図 7.20 であり, ユーザーが受信した α は図 7.21 であることから, α の値が一致しており, 送受信が正しく行われたことが分かる.

```
//////////////  An+2 ← H(S XOR N+2)  //////////////
//////////////  α ← An+2 XOR An+1 XOR Mn+1  //////////////
alpha:  103 224 67 117 164 252 105 152 92 62 46 25 19 216 83 214

//////////////  server --- (alpha) ---> client  //////////////
send data
send alpha.
```

図 7.20　暗号化通信の際にサーバーが送信した α

```
Received α ok
alpha:
103,224,67,117,164,252,105,152,92,62,46,25,19,216,83,214,
```

図 7.21　暗号化通信の際にユーザーが受信した α

7.4.3　総合テスト

総合テストでは要件定義を参考してテストを行った. 総合テスト項目と結果を表 7.6 に示す. Arduino はユーザー,Raspberry Pi はサーバーと定義する.

表 7.6　総合テスト

機能枠	確認内容	結果	確認日	確認者	備考欄
認証	ユーザーは起動後,1 度時刻同期を行う.	○	2022/1/14	浅野 内山田	
	ユーザーが指定した時間に認証要請を送信する.	○	2022/1/14	浅野 内山田	
	複数のユーザーは同時刻に認証要請を送信するので, 一台の認証が終了するまで, その他のユーザーは待機状態になる.	○	2022/1/14	浅野 内山田	
	ユーザーが認証処理を行っている.	○	2022/1/14	浅野 内山田	図 7.23
	ユーザーが認証結果をモニタに表示している.	○	2022/1/14	浅野 内山田	
	認証が成功した場合, ユーザーが SAS-L2 に基づいた 暗号化通信を行う.	○	2022/1/14	浅野 内山田	
	認証が失敗した場合, ユーザーはコネクションを切断して 認証要請送信から再度開始する.	○	2022/1/14	浅野 内山田	
	サーバーが起動したら, 受信待ち状態になる.	○	2022/1/14	浅野 内山田	
	サーバーが認証要請受信後, 認証処理を行っている.	○	2022/1/14	浅野 内山田	図 7.22
	サーバーが認証結果をモニタに表示している.	○	2022/1/14	浅野 内山田	
	認証が成功した場合, サーバーが SAS-L2 に基づいた暗号化通信を 行う.	○	2022/1/14	浅野 内山田	
	認証が失敗した場合, サーバーはコネクションを切断して 認証要請の待ち状態となる.	○	2022/1/14	浅野 内山田	
暗号化通信	ユーザーはセンシングデータを取得する.	○	2022/1/14	浅野 内山田	
	ユーザーは, シリアルモニタに取得したセンシングデータと センシングデータ取得日時を表示する.	○	2022/1/14	浅野 内山田	図 7.24 図 7.25 図 7.26
	ユーザーはセンシングデータと日時を暗号化して サーバーへ送信する.	○	2022/1/14	浅野 内山田	
	ユーザーは暗号化通信を 10 回行う.	○	2022/1/14	浅野 内山田	
	ユーザーは暗号化通信後, コネクションを切断する.	○	2022/1/14	浅野 内山田	
	サーバーはセンシングデータと日時を復号する.	○	2022/1/14	浅野 内山田	
	サーバーはセンシングデータと取得日時を保存する際に, ユーザーごとのテーブルに分けて保存する.	○	2022/1/14	浅野 内山田	図 7.27 図 7.28 図 7.29
	サーバーは暗号化通信を 10 回行う.	○	2022/1/14	浅野 内山田	
	サーバーは暗号化通信後, コネクションを切断し 認証要請待ち状態となる.	○	2022/1/14	浅野 内山田	
通信	ユーザーで, 何らかのエラーが発生した場合, 赤 LED を点滅させる.	○	2022/1/14	浅野 内山田	
	サーバーは,5 秒以上データを受信できなけ れば, コネクションを切断して認証要請 受信の待機状態となる.	○	2022/1/14	浅野 内山田	
	3 台のユーザーが同時刻に認証要請を送信した場合, 3 台のユーザーとの通信が 10 秒以内に完了する.	○	2022/1/14	浅野 内山田	

図 7.22 はサーバーの ID:1 のユーザーとの認証処理の結果であり, 図 7.23 は ID:1 のユーザーの認証処理の結果をモニタに表示した図である. サーバーは認証要請を受信後, 生成した認証情報 A_{n+1} を基に α を生成し, ユーザーに送信している. そして, サーバーはユーザーから β を受信し, 認証結果をユーザーへ送信していると分かる. ユーザーは認証要請を送信後, サーバーから受信した α を基に β を生成し, サーバーに送信している. そして, ユーザーはサーバーから認証結果を受信し, 認証結果から認証が成功したと分かる. サーバーと ID:2,ID:3 の間での認証処理についても, 同様に認証が成功したことを確認した.

```
client 3  End Connect.
//////////  client --- (request) ---> server  //////////
connect: <socket.socket fd=4, family=AddressFamily.AF_INET, type=SocketKind.S
recieve data
request:  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
client1: Start Connect.

//////////  An+1 ← H(S XOR N+1)  //////////
An+1:  111 9 122 16 46 20 58 36 108 132 191 12 89 133 175 130

//////////  α ← An+1 XOR An XOR Mn  //////////
alpha:  12 174 98 76 233 160 251 84 207 57 3 57 146 167 246 23

//////////  server --- (alpha) ---> client  //////////
send data
send alpha.

//////////  client --- (beta) ---> server  //////////
recieve data
beta:  234 146 190 233 206 11 197 204 167 148 166 38 245 211 148 249

//////////  An+1 + An = β ?  //////////

//////////  server --- (result) ---> client  //////////
Success Authentication.
send data
send result.
147 183 161 94 248 58 214 128 211 194 66 73 243 189 162 89
```

図 7.22　サーバーの認証結果

```
create tcp ok
request message ok
Received α ok
alpha:
12,174,98,76,233,160,251,84,207,57,3,57,146,167,246,23,
beta:
234,146,190,233,206,11,197,204,167,148,166,38,245,211,148,249,
beta send ok
Received:Authentication ok
Authentication  is successful.
An+1:
111,9,122,16,46,20,58,36,108,132,191,12,89,133,175,130,
Mn+1:
147,183,161,94,248,58,214,128,211,194,66,73,243,189,162,89,
```

図 7.23　ユーザーの認証結果

ユーザーが取得したセンシングデータと日時は図 7.24 と図 7.25, 図 7.26 のようにシリア

ルモニタに表示されることが確認された. 示した図はそれぞれある時点のものである. サーバーがデータベースに保存したセンシングデータと日時は図 7.27 と図 7.28, 図 7.29 のようにユーザーの ID ごとにテーブルに分て保存されていることが分かる. また, ユーザーが取得したセンシングデータがデータベースに保存されていることから, ユーザーが取得したセンシングデータを暗号化してサーバーに送信し, サーバーがセンシングデータを復号できたことが分かる.

```
====================
UV: 0.04
2022/1/13/16/32/2
ganma:
41,163,233,195,182,5,176,129,71,171,90,54,169,27,103,174,
send(ganma) ok
Received α ok
alpha:
103,224,67,117,164,252,105,152,92,62,46,25,19,216,83,214,
====================
UV: 0.03
2022/1/13/16/32/2
ganma:
200,162,32,164,91,32,190,21,159,220,110,112,140,182,220,212,
send(ganma) ok
```

図 7.24　ID:1 のシリアルモニタ

```
====================
Temp: 22.4375*C
2022/1/13/16/32/3
ganma:
171,101,57,158,100,113,0,246,65,224,106,38,176,180,139,47,
send(ganma) ok
Received α ok
alpha:
63,205,171,109,67,38,225,34,206,107,145,64,241,195,214,60,
====================
Temp: 22.4375*C
2022/1/13/16/32/3
ganma:
62,92,15,88,185,205,113,140,251,158,102,64,71,135,139,92,
send(ganma) ok
```

図 7.25　ID:2 のシリアルモニタ

```
Relative Humidity :33.27 %RH
2022/1/13/16/32/0
ganma:
16,188,13,82,56,27,72,179,244,115,146,219,72,210,101,227,
send(ganma) ok
Received α ok
alpha:
28,231,167,163,92,154,18,234,247,175,231,212,249,30,61,97,
Relative Humidity :33.39 %RH
2022/1/13/16/32/0
ganma:
40,116,92,16,29,131,207,145,90,143,76,148,164,171,76,114,
send(ganma) ok
```

図 7.26　ID:3 のシリアルモニタ

```
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 |
+----+------+-------+-----+------+-----+-----+------+
```

図 7.27　データベースの ID:1 のテーブル

```
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 |
+----+------+-------+-----+------+-----+-----+---------+
```

図 7.28　データベースの ID:2 のテーブル

```
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 |
+----+------+-------+-----+------+-----+-----+---------+
```

図 7.29　データベースの ID:3 のテーブル

7.5　考察

本節では,IoT におけるセンシングデバイスでのセキュアなデータ通信の実装での考察と問題点, 今後の課題について述べる.

まず, 本研究で開発したシステムの考察を述べる. 本研究では,SAS-L2 認証方式をセンシングデバイスに実装することで, デバイス間の認証とセンシングデータの暗号化通信方法の実現に成功した. 具体的に, 認証については図 7.22 と図 7.23 を用いて示したように, 認証が成功したことから SAS-L2 認証の実装が実現できたことが分かる. そして, 暗号化通信では, センシングデバイスが取得したセンシングデータを格納した SD と認証情報 A_n と認証情報 A_{n+1} の排他的論理和をサーバーへ送信し, サーバーがセンシングデータを復号してデータベースへ保存できたことからセンシングデバイスへの暗号化通信方法の実装が実現できたと分かる.

次に, 本システムの問題点と今後の課題について述べる. 問題点の 1 つ目として, 本研究で

はセンシングデバイスが SAS-L2 認証を用いてサーバーとのセキュアなデータ通信を行うにあたり, 初期認証情報と初期秘匿情報がセンシングデバイスに書き込まれていることが前提条件であることである. 実際では, センシングデバイスを生産するにあたり, 初期情報が書き込まれているなら前提条件が満たされているが, 初期情報が書き込まれていないセンシングデバイスに関しては, 安全な方法で初期情報を取得しなければならない. 安全な方法により初期情報を取得しなければ, 悪意のある第三者により初期情報が盗聴され, 以後の認証・暗号化通信も盗聴・改ざんされる可能性がある. 初期情報が書き込まれていないセンシングデバイスである場合は, 初期情報を取得する安全な方法を確保しなければならないことが今後の課題である.

本研究ではエッジデバイスとして Arduino Leonardo を使用したが,Arduino は電源を切った場合に初期化するため, 認証を行う度に認証情報と秘匿情報を不揮発性メモリに書き込まなければならない. 問題点の 2 つ目として, このようにセンシングデバイスが電源を切るたびに初期化してしまうものであった場合, 不揮発性メモリを使用しなくてはならないことである. 不揮発性メモリは, デバイスの電源が切れても初期化されないが, 書き込み回数には限界がある. デバイスにより不揮発性メモリの寿命も様々であるが, 認証回数が増える程にメモリへの書き込み回数も増えるため, メモリの寿命が短くなる. 不揮発性メモリへの書き込みができなくなれば, 認証・暗号化通信もできなくなるため, 一定時間内の認証の回数を決定する際, 不揮発性メモリの書き込み回数の限界を考慮しなければならないことが課題である.

第8章　あとがき

本研究の研究背景として IoT デバイスに対するセキュリティ対策として認証・暗号化通信を行うことが挙げられるが, 処理性能の低い IoT デバイスでは従来法の暗号方式での実装が困難である. そこで, 本研究ではワンタイムパスワード認証方式 SAS-L2 を IoT センシングデバイスに実装することで, デバイス間の認証およびセンシングデータの暗号化通信方法を実現することを目標に開発を行った. SAS-L2 認証方式を導入した IoT システムの設計は UML のユースケース図, クラス図, シーケンス図などを用いて行い, その設計に基づきチームで分担し開発を行った. 実装完了後は,V 字開発モデルに従って, 単体テスト, 結合テスト, 総合テストの順でテストを行った. 検証結果から,SAS-L2 認証方式をセンシングデバイスに実装し, デバイス間の認証と SAS-L2 に基づいたデータ通信の暗号化ができたことが分かる. また, センシングデータの暗号化では，排他的論理演算 2 回で済むため，処理能力の低いセンシングデバイスにおいてリアルタイムでの暗号化通信を実現することができる. そして, 問題点と課題として, センシングデバイスに初期認証情報や初期秘匿情報が書き込まれていない場合, 初期情報を取得する際の安全な方法の確立について, そして, 認証情報と秘匿情報を不揮発性メモリに書き込む場合, 不揮発性メモリの書き込み回数を考慮して認証回数の決定をしなくてはならないという点が挙げられる. それらを改善することで, 本システムでの処理性能の低いセンシングデバイスに認証方式と暗号化通信を導入することによる, さらなるセキュリティ強化に繋がると考える.

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高橋寛教授ならびに甲斐博准教授, 王森レイ講師に深く御礼申し上げます.

また, 審査頂いた本学高橋寛教授, 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, 多大な御協力と貴重な御助言を頂いた本学工学部情報工学科情報システム工学講座高橋研究室の諸氏に厚く御礼申し上げます.

参考文献

［1］ IPA 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.“ 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”.IPA 独立行政法人情報処理推進機構. https://www.ipa.go.jp/files/000009384.pdf, (参照 2022-01-05)

［2］ 斉藤貴之.“ 3 分でわかる AES”. 日経クロステック. https://xtech.nikkei.com/atcl/nxt/keyword/18/00002/030800119/, (参照 2022-01-05)

［3］ “最強の暗号？バーナム暗号”.kakke18’s blog.2019-02-22. https://kakke18.hatenablog.com/entry/2019/02/22/193223, (参照 2022-01-05)

［4］ 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. オーム社.2019.

［5］ 清水明宏.SAS-L ワンタイムパスワード認証方式について.preprint.2020.

［6］ 水野忠則, 中篠直也, 井上雅裕, 山田圀裕. 未来へつなぐデジタルシリーズ 20 組込システム. 共立出版.2017.

［7］ 永和システムマネジメント.“ UML 超入門”. オブラブ. http://objectclub.jp/technicaldoc/uml/umlintro, (参照 2022-01-05)