令和元年度

修士学位論文

SAS の同期問題に関する研究

A study on SAS Asynchronous problems

1225122　　高橋 錬

指導教員　　清水 明宏

2020 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報学コース

要 旨

SAS の同期問題に関する研究

高橋 錬

IoT の普及に伴い，IoT デバイスへの攻撃も増加している．IoT デバイスの中には処理能力などの制限が厳しいものも存在する．このようなデバイスには軽量なセキュリティ技術が求められる．これらのデバイスに適したセキュリティ技術として SAS が提案されている．SAS は共通鍵暗号方式をベースとしており，相互認証・鍵配送・暗号通信を実現している．SAS を導入した場合，デバイス間の通信経路が何らかの原因で遮断されることで認証情報にズレが生じる可能性がある．これを同期問題という．この同期問題によって可用性が損失することはシステム全体に大きな影響を与える．また，この同期問題を意図的に引き起こすことで攻撃にもなり得るため，解決する必要がある．同期問題解決手法として中原らの方式と藤田の方式が提案されている．これらの方式では可用性の担保を実現している一方でなりすましによる安全性への課題が存在する．

本稿では既存方式の課題を解決し，可用性と安全性を担保する方式を提案する．また，同期問題対策による拡張が処理時間に与える影響についても調査し，拡張による処理への影響が限りなく小さいことを示す．

キーワード　IoT，SAS，同期問題，なりすまし，拡張による影響

Abstract

A study on SAS Asynchronous problems

Takahashi Ren

With the spread of IoT, attacks on IoT devices are also increasing.Some IoT devices have severe limitations such as processing power.Such devices require lightweight security technology.SAS has been proposed as a security technology suitable for these devices.SAS is based on common key cryptography, and realizes mutual authentication, key distribution, and encrypted communication.When SAS is introduced, the communication path between devices may be interrupted for some reason, causing a shift in authentication information. This is called a synchronization problem.Loss of availability due to this synchronization problem has a significant impact on the entire system.In addition, since intentionally causing this synchronization problem can be an attack, it needs to be resolved.Nakahara et al.'S method and Fujita's method have been proposed as methods for solving synchronization problems.While these methods ensure availability, there are security issues due to spoofing.In this paper, we propose a method that solves the problems of the existing method and ensures availability and security.In addition, we investigate the effect of the extension due to the synchronization problem on the processing time, and show that the effect of the extension on the processing is infinitely small.

key words IoT, SAS, synchronization issues, Spoofing, the impact of expansion

目次

図目次

表目次

第 1 章

序論

近年，自動化による人手不足解消や管理の簡略化を目的として，IoT(Internet of Things) の広がりが著しい．その例として農業では IoT デバイスを用いて温度や湿度をセンシングし，分析することで農業ハウスの温度を自動的に管理する動きが見られている．その一方で，IoT デバイスを標的とした攻撃が増加しており，今後も増加することが予想される [1][2]．従って IoT で用いられるデバイスにおいてもセキュリティが必要である [3]．

IoT デバイスには様々なものがあり，メモリや CPU のリソースに制約のあるものからデータを集約する高度な処理が可能なデバイスまで多岐に渡る．リソースの制限が厳しいデバイスを利用する場合には軽量なセキュリティ技術が求められる．この要件を満たす軽量な相互認証方式として SAS がある．この方式は共通鍵暗号方式をベースとして軽量な相互認証を実現している．しかし，何らかの障害により通信が切断された場合にサーバ側とユーザ側で認証情報のズレが発生する．これを同期問題と呼ぶ．可用性の観点からこの同期問題が発生した場合でも継続して通信を行える必要がある．

この同期問題を解決する手法として中原らの方式と藤田の方式がある．中原らの方式では認証情報をバックアップすることで拡張の影響を最小限に押さえ，同期問題を解決している．しかし，リプレイ攻撃によるなりすましが課題としてあげられる．この課題を藤田の方式ではチャレンジ&レスポンス方式を採用することで解決している．藤田の方式では同期問題を解決し，リプレイ攻撃によるユーザ側なりすまし耐性を実現している．しかし，サーバ側になりすますことが可能である．そこで本論文では，SAS を導入する際に起こりうる同期問題を解決し，なりすましへの安全性を確立することで安全性と可用性を満たす方式を提案する．また，処理時間を計測し，拡張による影響が少ないことを示す．

本論文は全 7 章で構成される．第 2 章では SAS について説明し，同期問題がどのように引き起こされるかを明らかにする．第 3 章では同期問題を解決する既存方式について取り上げ，その課題を明確にする．第 4 章では本研究の目的を明らかにし，既存方式の課題を解決する方式を提案する．第 5 章は提案方式の処理負荷を処理時間の観点から調査するための実験について述べる．第 6 章では提案方式の安全性と実験結果について評価し，SAS の適用箇所について述べる．最後に第 7 章で結論を述べる．

第 2 章

SAS

本章では SAS について述べる．まず，ワンタイムパスワード認証方式について述べる．そして相互認証を実現している SAS-2 について説明する．次に，軽量なデバイスへの適用を目的として提案されている SAS-L について説明する．最後に同期問題について説明する．

2.1 ワンタイムパスワード認証方式

一般的なネットワークで用いられる認証技術として TLS(Transport Layer Security)[4] がある．この方式は公開鍵暗号方式を用いた電子署名によって認証を行う．TLS は広く普及しており，安全性が高いことが特徴である．しかし，計算処理が複雑で処理負荷が重いことからリソースの制限が厳しいデバイスへの適用は不向きである [6]．IoT デバイス向けの軽量な認証方式として SAS(Simple And Secure password authentication protocol) が提案されている [5]．SAS はその他の共通鍵暗号方式と比較して一方向性関数の適用回数が少ないことが特徴である．また，SAS はワンタイムパスワード認証方式でもあり，通信路上を流れる認証情報が毎回変化することからリプレイ攻撃やなりすましへの耐性がある．SAS にはいくつかの種類があり，相互認証の実現や IoT デバイス向けに初期の SAS よりもさらに一方向性関数の適用回数を削減した方式が提案されている．

2.2 SAS-2

SAS のの 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ている．また，鍵配送と暗号通信も同時に実現してい

る．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，軽量さが特徴である．SAS-2 は初回登録フェーズと認証フェーズで構成されている．初回登録フェーズは安全な通信路を用いて行われる．ここから，各フェーズについて説明する．

2.2.1　定義

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．

用語・記号	定義
認証情報	サーバとユーザで共有された認証に用いる情報
今回認証情報	現在のセッションで認証に用いる認証情報
次回認証情報	次のセッションで認証に用いる新しい認証情報
i	セッションの回数を表す 1 以上の整数値
N_i	i 回目の認証情報を生成時に用いる乱数
A_1	初回認証情報
A_i	i 回目セッションで用いる今回認証情報
A_{i+1}	次回認証情報
P	ユーザのパスワード
ID	ユーザを識別する ID
$H(\cdot)$, $H(\cdot)$, $F(\cdot)$	一方向性関数
$\oplus$	排他的論理和 (XOR)
$+$	加算

表 2.1　SAS-2 の定義

2.2.2 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を用いて行う．登録フェーズの図 2.1 を以下に示す.

- ユーザ
  1. 乱数 N_1 を生成し，ID と P と共に初回認証情報 $A_1 = H(ID|P \oplus N_1)$ を生成する
  2. 生成した ID と A_1 を安全な通信路を用いてサーバに送信する
  3. 送信した N_1 を保存する
- サーバ
  1. ユーザから送られた A_1 を ID に紐付けて保存する

2.2.3 認証フェーズ

i 回目認証フェーズの図 2.2 を以下に示す.

- ユーザ
  1. 保存してある乱数 N_i と ID，P と共に今回認証情報 $A_i = H(ID|P \oplus N_i)$ を生成する
  2. 新たに乱数 N_{i+1} を生成し，ID と P と共に次回認証情報 $A_{i+1} = H(ID|P \oplus N_{i+1})$ を生成する
  3. 生成した A_{i+1} から $H(A_{i+1})$ 生成する
  4. A_i と A_{i+1}，$H(A_{i+1})$ を用いて $\alpha = A_{i+1} \oplus (H(A_{i+1}) + A_i)$ と $\beta = H(A_{i+1}) \oplus A_i$ を生成する
  5. ID と α，β をサーバに送信する
  6. $F(H(A_{i+1}))$ を生成する
  7. サーバから γ を受け取り，生成した $F(H(A_{i+1}))$ と比較する
  8. 一致すれば認証成功となり，N_i を N_{i+1} に更新して終了する

2.2 SAS-2

- サーバ

  1. ユーザから送られた β，と保存してある A_i を用いて $H(A_{i+1})$ を算出する
  2. 算出した $H(A_{i+1})$，保存してある A_i，送られた α を用いて A_{i+1} を算出する
  3. 算出した A_{i+1} から $H(A_{i+1})$ を生成し，β から算出した $H(A_{i+1})$ と比較する．一致すれば認証が成立となり以降の処理を行う．
  4. 認証が成立した場合，$\gamma = F(H(A_{i+1}))$ を生成し，ユーザに送信する
  5. A_i を A_{i+1} に更新して終了する

2.2　SAS-2

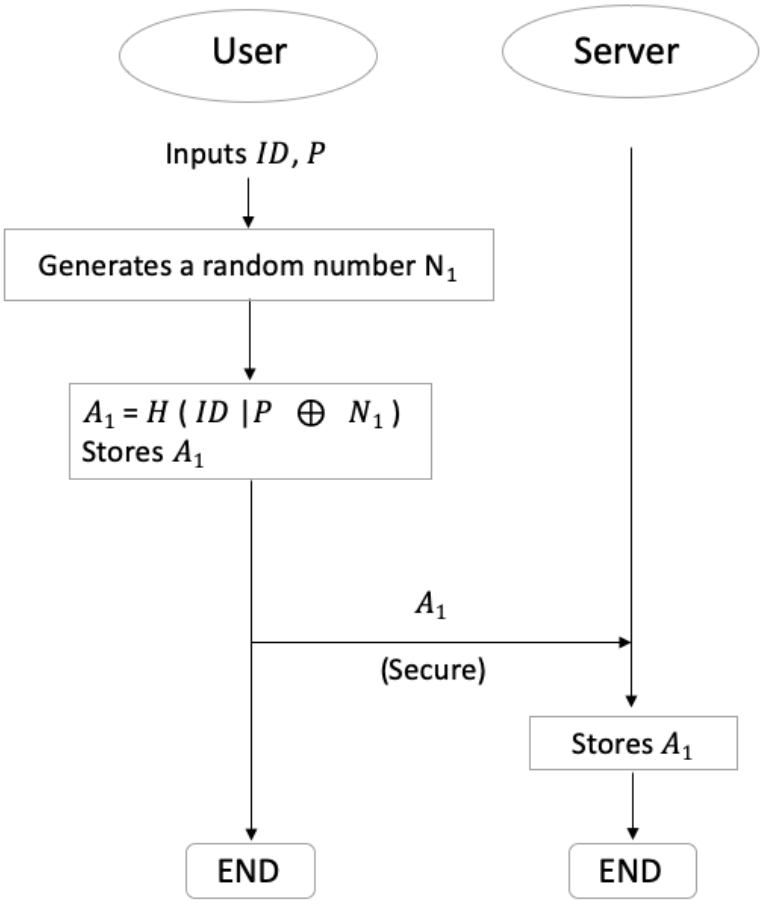

図 2.1　SAS-2 の初回登録フェーズ

2.2　SAS-2

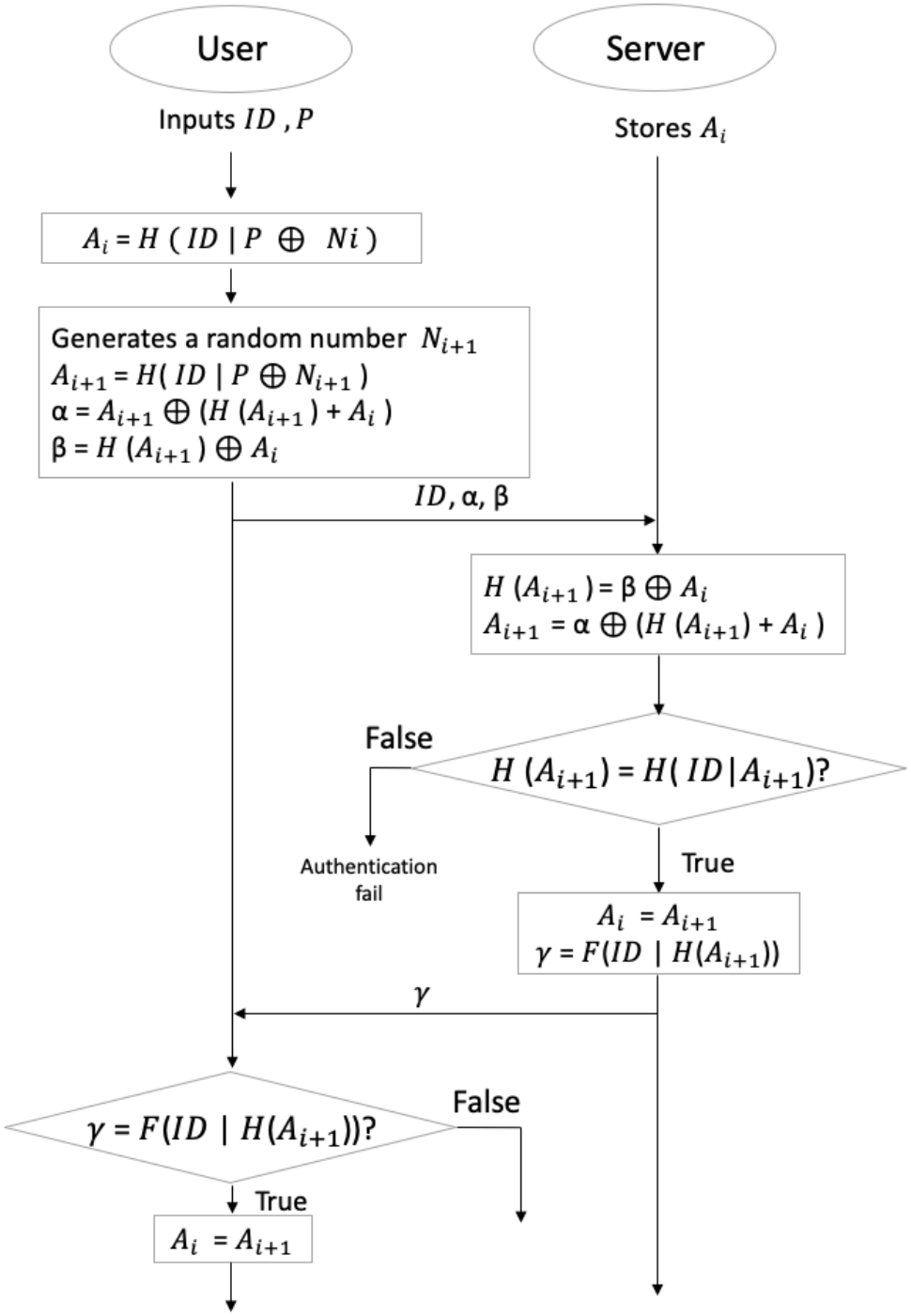

図 2.2　SAS-2 の i 回目認証フェーズ

2.3 SAS-L

SAS の 1 つとして SAS-L がある．この方式は IoT デバイス向けに提案されており，一方向性関数の適用回数が SAS の中で最も少なく，相互認証を実現している．さらに，鍵配送と暗号通信も同時に実現している．SAS-L はクライアント側の一方向性関数の適用回数が 1 回，サーバ側が 0 回であるためサーバに接続する端末数が多い場合にサーバの処理負荷を軽減できるという特徴がある．また，相互認証方式であることからサーバとクライアントの端末を入れ替えることでリソースの低いデバイスへの適用も可能である [7]．SAS-L は初回登録フェーズと認証フェーズで構成されている．初回登録フェーズは安全な通信路を用いて行われる．ここから，各フェーズについて説明する．

2.3.1 定義

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の追加部分を記載する．

用語・記号	定義
i	セッションの回数を表す 1 以上の整数値
M_1	初回マスク値
M_i	i 回目セッションで用いるマスク値
M_{i+1}	次回マスク値

表 2.2 SAS-L の定義

2.3.2 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を用いて行う．登録フェーズの図 2.3 を以下に示す．

- クライアント

1. 乱数 N_1 を生成し，ID と P と共に初回認証情報 $A_1 = H(ID|P \oplus N_1)$ を生成する
2. 初回マスク値 M_1 を生成する
3. 生成した ID と A_1，M_1 を安全な通信路を用いてサーバに送信する
4. 送信した N_1，M_1 を保存する

- サーバ
  1. クライアントから送られた A_1 と M_1 を ID に紐付けて保存する

2.3.3 認証フェーズ

i 回目認証フェーズの図 2.4 を以下に示す.

- クライアント
  1. 保存してある乱数 N_i と ID，P と共に今回認証情報 $A_i = H(ID|P \oplus N_i)$ を生成する
  2. 新たに乱数 N_{i+1} を生成し，ID と P と共に次回認証情報 $A_{i+1} = H(ID|P \oplus N_{i+1})$ を生成する
  3. A_i と A_{i+1}，M_i を用いて $\alpha = A_{i+1} \oplus M_i \oplus A_i$ と $\beta = A_{i+1} + M_i$ を生成する
  4. ID と α，β をサーバに送信する
  5. $M_{i+1} = M_i + A_{i+1}$ を生成する
  6. サーバから γ を受け取り，生成した $A_i \oplus M_{i+1}$ と比較する
  7. 一致すれば認証成功となり，N_i を N_{i+1}，M_i を M_{i+1} に更新して終了する
- サーバ
  1. クライアントから送られた α，と保存してある A_i，M_i を用いて A_{i+1} を算出する
  2. 算出した A_{i+1}，保存してある M_i を用いて送られた $\beta = A_{i+1} + M_i$ を比較する
  3. 一致すれば認証が成立となり $M_{i+1} = A_{i+1} + M_i$ を生成後，以降の処理を行う.
  4. 認証が成立した場合，$\gamma = A_i \oplus M_{i+1}$ を生成し，クライアントに送信する
  5. A_i を A_{i+1}，M_i を M_{i+1} に更新して終了する

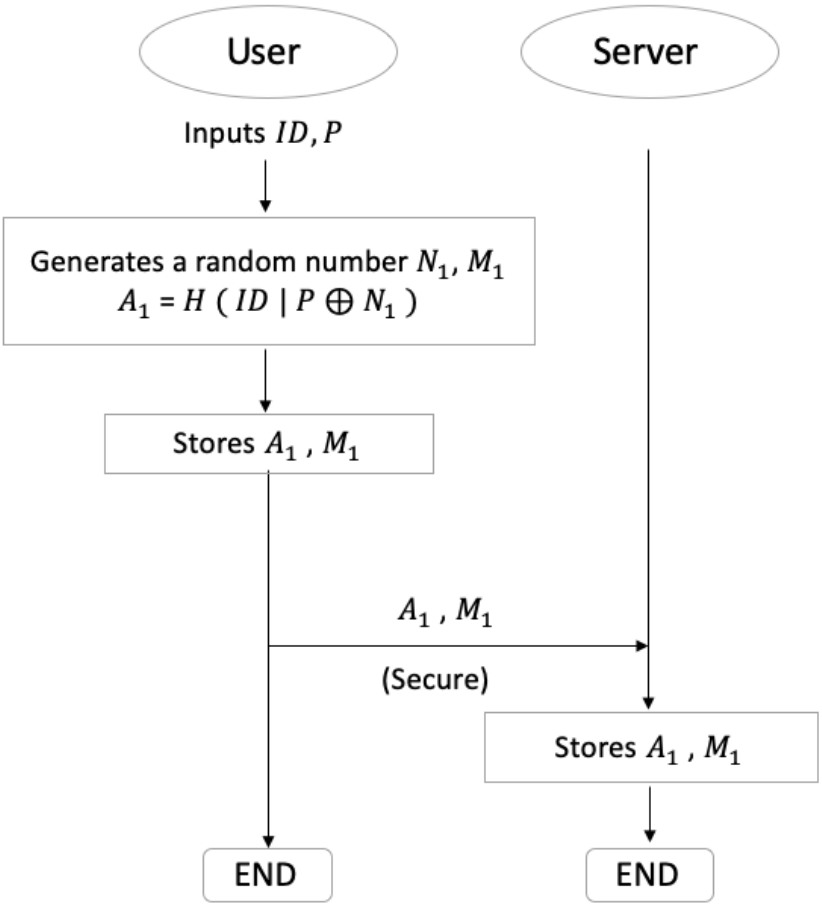

図 2.3　SAS-L の初回登録フェーズ

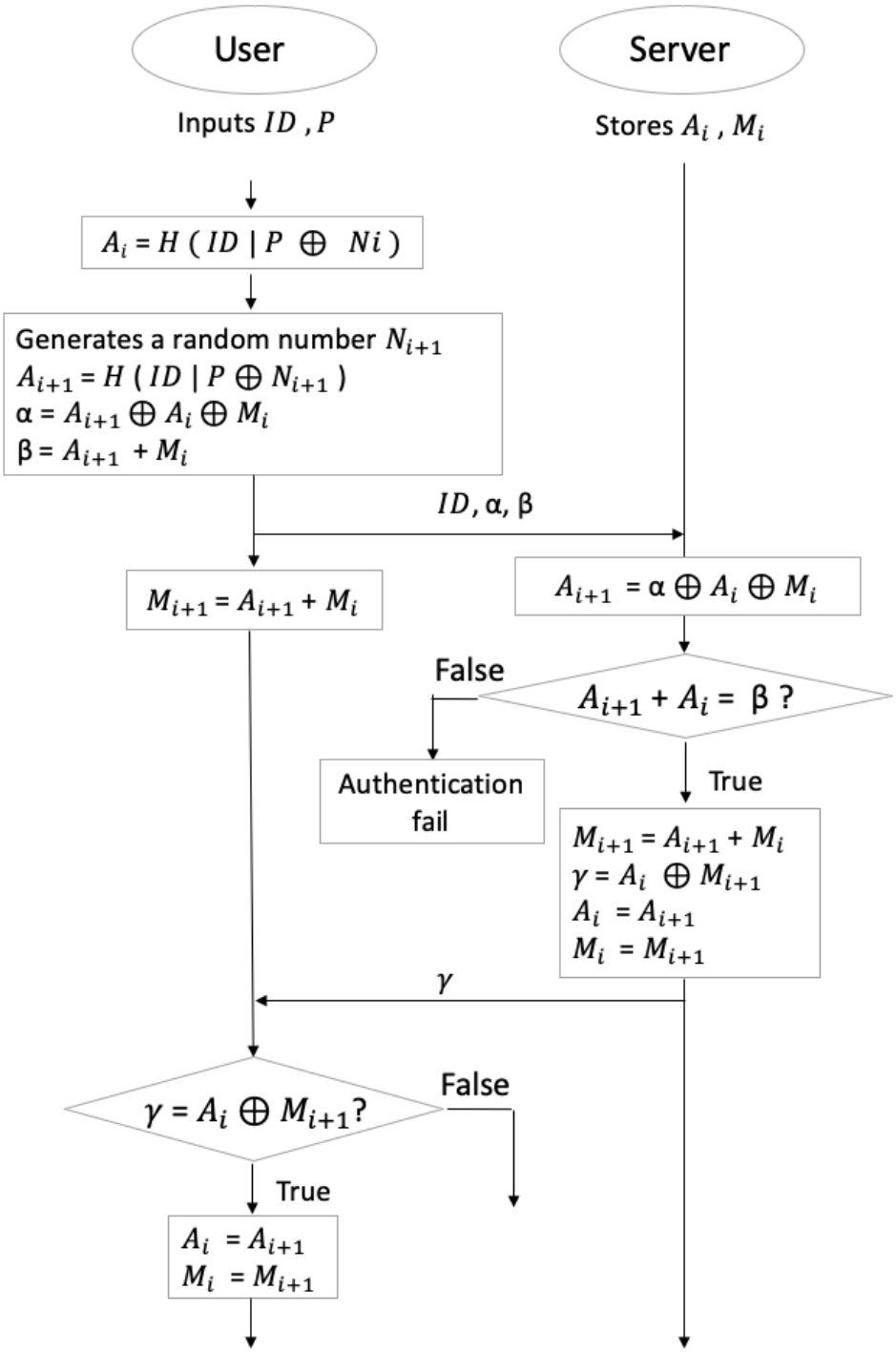

図 2.4　SAS-L の i 回目認証フェーズ

2.4　SAS における同期問題

システムを継続的に利用するための要件として可用性がある．これは何らかの障害により通信が切断された場合やパケットロスが発生した場合でも，その後の通信を継続できることを指す．SAS-2 の適用を検討した場合に同期問題によって可用性が保証されない場合がある．同期問題とは何らかの影響でパケットが相手側の端末に到達しなかった場合に端末間の認証情報にズレが生じることで次回以降の認証ができなくなる問題のことである．IoT で利用するネットワークの中には屋内だけでなく屋外でも利用されることが予想されるためこの同期問題が起こるリスクも高くなる．また，意図的に通信を遮断し同期問題を引き起こす攻撃 (同期ズレ攻撃) も可能となるためこの問題への対策は重要となる．同期問題を図 2.5 に示す．

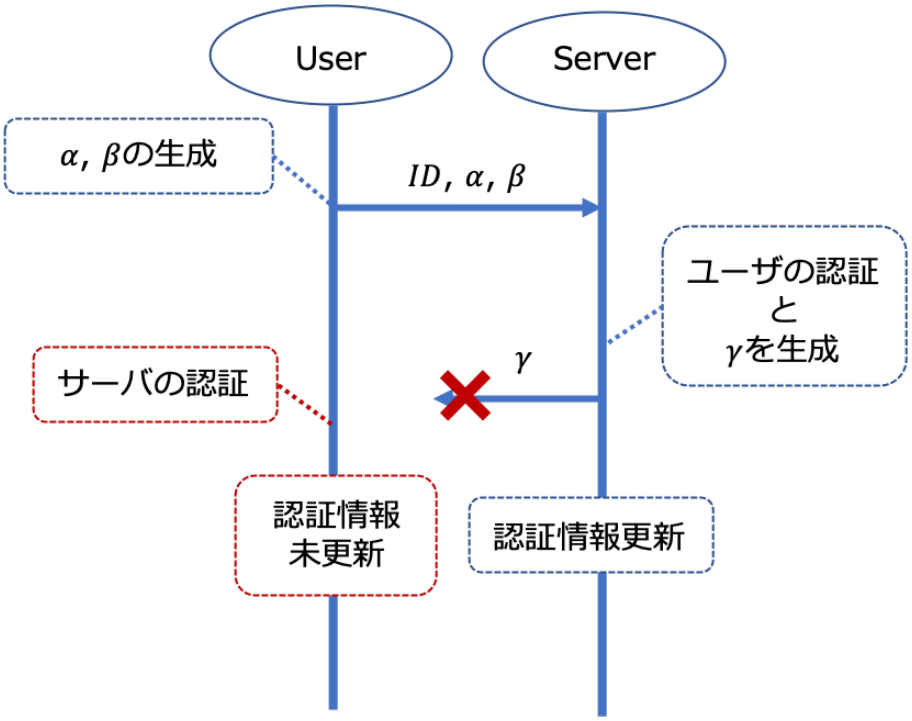

図 2.5　同期問題発生時の認証フェーズ

第 3 章

同期問題対策手法

本章では同期問題を解決するために提案されている手法について説明する．まず，中原らの方式と藤田の手法について説明する．その後，藤田の手法の課題を明確にする．

3.1　中原らの方式 (Back Up 方式)

SAS の同期問題を解決する手法として中原らの方式がある．中原らの方式では前回認証情報をバックアップとして保持することで SAS-2 における同期問題に対応している．構成は初回登録フェーズと認証フェーズからなる．初回登録フェーズは安全な通信路を用いて行われる．以下に中原らの方式の概要を示す．

3.1.1　定義

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の追加部分を記載する．

用語・記号	定義
Y_A	バックアップした今回認証情報

表 3.1　中原らの方式で用いる用語の定義

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を用いて行う．初回登録フェーズを図 3.1 に示す.

- ユーザ
  1. 乱数 N_1 を生成し，ID と P と共に初回認証情報 $A_1 = H(ID|P \oplus N_1)$ を生成する
  2. 生成した ID と A_0 を安全な通信路を用いてサーバに送信する
  3. 送信した N_1 を保存する
- サーバ
  1. ユーザから送られた A_1 を ID に紐付けて保存する
  2. さらにバックアップ情報 $Y_A = A$ を保存する

3.1.3　認証フェーズ

i 回目認証フェーズを図 3.2 に示す.

- ユーザ
  1. 保存してある乱数 N_i と ID，P と共に今回認証情報 $A_i = H(ID|P \oplus N_i)$ を生成する
  2. 新たに乱数 N_{i+1} を生成し，ID と P と共に次回認証情報 $A_{i+1} = H(ID|P \oplus N_{i+1})$ を生成する
  3. 生成した A_{i+1} から $H(A_{i+1})$ 生成する
  4. A_i と A_{i+1}，$H(A_{i+1})$ を用いて $\alpha = A_{i+1} \oplus (H(A_{i+1}) + A_i)$ と $\beta = H(A_{i+1}) \oplus A_i$ を生成する
  5. ID と α，β をサーバに送信する
  6. $F(H(A_{i+1}))$ を生成する
  7. サーバから γ を受け取り，生成した $F(H(A_{i+1}))$ と比較する

8. 一致すれば認証成功こなり，N_i を N_{i+1} に更新して終了する

- サーバ

1. ユーザから送られた β，と保存してある A_i を用いて $H(A_{i+1})$ を算出する
2. 算出した $H(A_{i+1})$，保存してある A_i，送られた α を用いて A_{i+1} を算出する
3. 算出した A_{i+1} から $H(A_{i+1})$ を生成し，β から算出した $H(A_{i+1})$ と比較する．一致しなければ 6 以下の処理を行う．一致すれば認証が成立となり 4 以降の処理を行う．
4. A_i を A_{i+1} に更新して終了する
5. 次回の同期ズレに備えて Y_A を A_i に更新し，10 の処理を行う
6. 認証不正立の場合は前回認証情報である Y_A を用いて β から $H(A_{i+1})$ を算出する
7. 算出した $H(A_{i+1})$ と α を元に A_{i+1} を算出する
8. 算出した A_{i+1} から認証情報を生成し，$H(A_{i+1})$ と比較する．一致すれば以下の処理を行う
9. 次回の同期ズレに備えて Y_A を A_i に更新する
10. $\gamma = F(H(A_{i+1}))$ を生成し，ユーザに送信する

3.1　中原らの方式 (Back Up 方式)

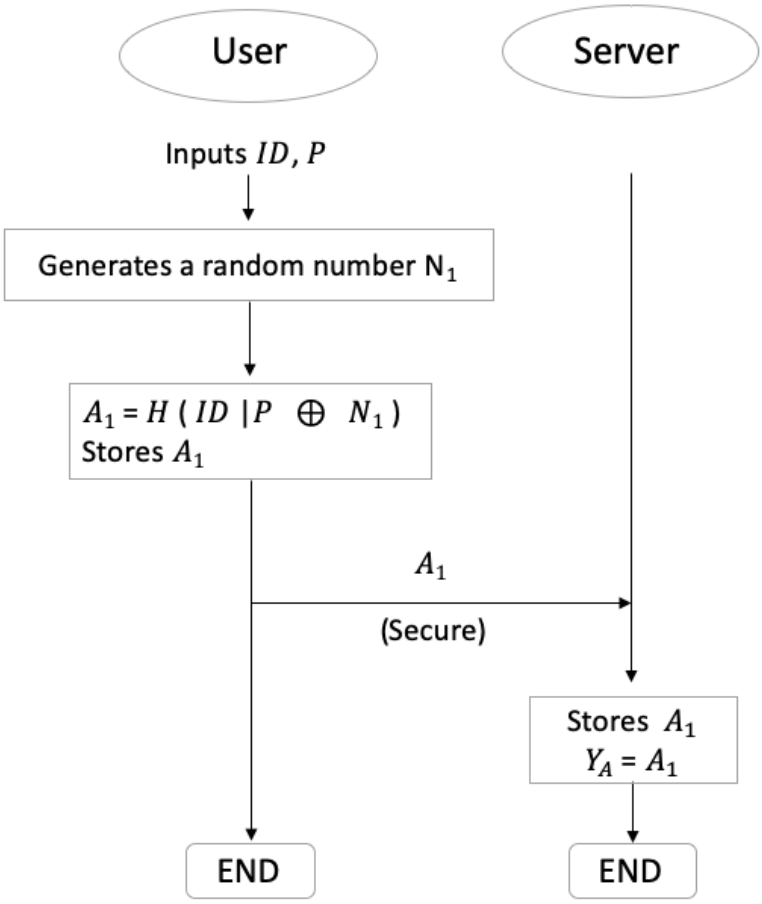

図 3.1　中原方式の初回登録フェーズ

3.1　中原らの方式 (Back Up 方式)

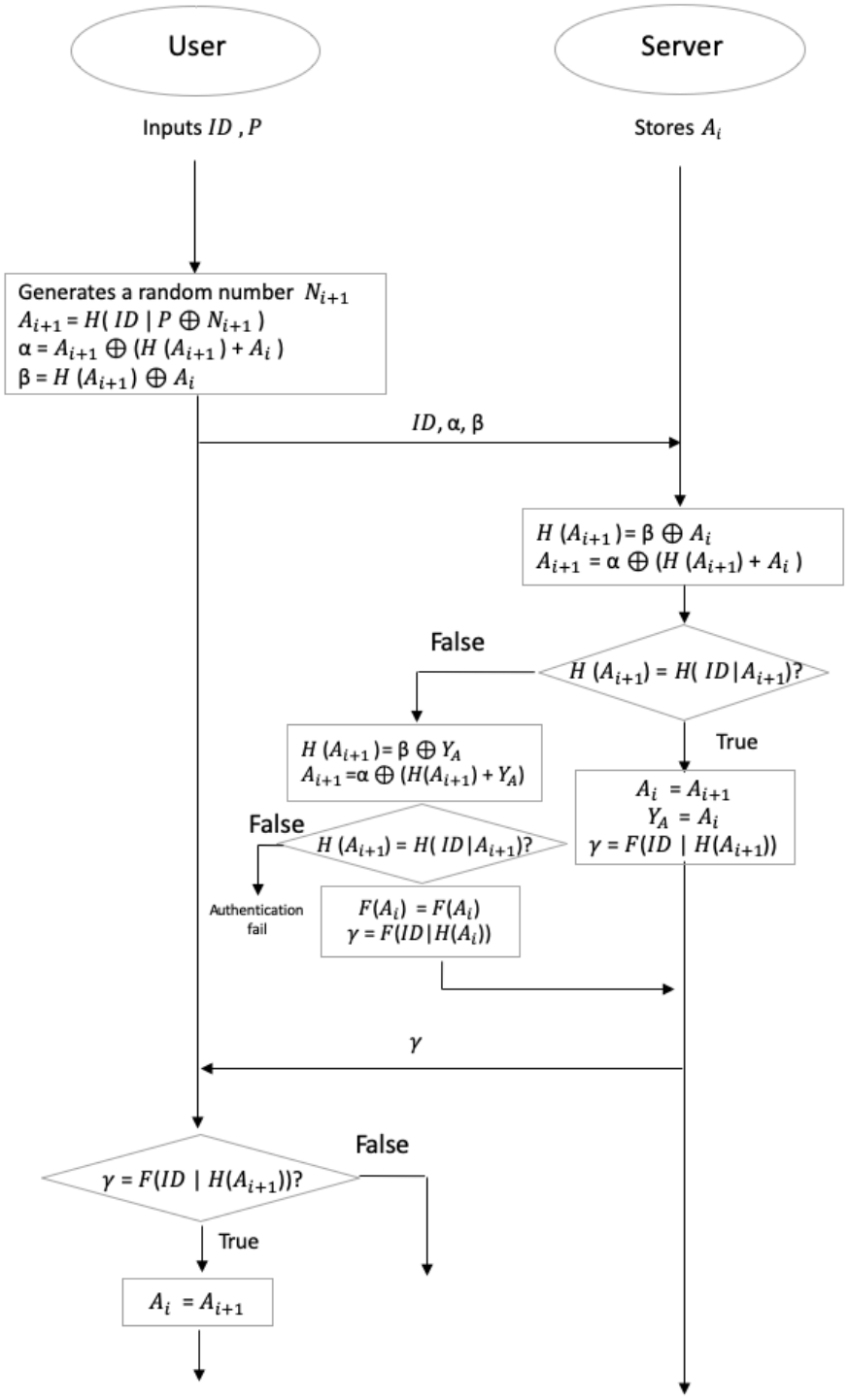

図 3.2　中原方式の i 回目認証フェーズ

3.1.4　バックアップ方式の課題

中原らの方式では前回利用した認証情報を保持することで同期問題を解決している．しかし，前回認証情報を保持することで攻撃者が通信路上から前回利用した α 及び β を盗聴することでリプレイ攻撃が可能となる．以下に中原らの方式において想定されるリプレイ攻撃について説明する．

対策前の SAS において次回認証に用いる情報は以下に示す 1 通りの情報のみである．そのため，攻撃者がこの情報を盗聴したとしても新しい情報を作成することが困難であることが安全性の根拠となっていた．

- $\alpha = G \oplus (H(G) + A_{i+1})$
- $\beta = H(G) \oplus A_{i+1}$

しかし，中原らの方式では以下に示す 2 通りの情報が有効となる．

- 今回情報
  - $\alpha = G \oplus (H(G) + A_{i+1})$
  - $\beta = H(G) \oplus A_{i+1}$
- 前回情報
  - $\alpha = G \oplus (H(G) + A_i)$
  - $\beta = H(G) \oplus A_i$

前回情報には今回の通信で流れた情報もその形に当てはまる．従って攻撃者は今回流れた情報をサーバに送ることで認証が成功してしまう．これが中原らの方式の課題である．

3.2　藤田の方式 (Challenge&Response 方式)

中原らの方式では前回認証情報をバックアップすることで同期問題を解決した．しかし，その一方でリプレイ攻撃によるなりすましが可能となってしまう．この問題を解決する手法

として藤田の方式がある．藤田の方式では Challenge&Response 方式 (以降 C&R 方式) を採用している..

以下で藤田の方式の説明を行う．中原らの方式と同様に初回登録会フェーズと認証フェーズから構成されている．

3.2.1　定義

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の追加部分を記載する．

用語・記号	定義
Y_A	バックアップした今回認証情報
Y_{A+1}	バックアップした次回認証情報
CR	チャレンジ情報

表 3.2　藤田の方式で用いる用語の定義

3.2.2　認証フェーズ

i 回目認証フェーズを図 3.3 に示す．

- ユーザ
  1. サーバ側にチャレンジをリクエストする
  2. 保存してある乱数 N_i と ID，P と共に今回認証情報 $A_i = H(ID|P \oplus N_i)$ を生成する
  3. 新たに乱数 N_{i+1} を生成し，ID と P と共に次回認証情報 $A_{i+1} = H(ID|P \oplus N_{i+1})$ を生成する
  4. 生成した A_{i+1} から $H(A_{i+1})$ 生成する
  5. サーバから受け取った CR と A_i，A_{i+1}，$H(A_{i+1})$ を用いて $\alpha = A_{i+1} \oplus (CR + A_i)$

と $\beta = H(A_{i+1}) \oplus A_i$ を生成する

6. ID と α，β をサーバに送信する
7. List に CR を保存する
8. $Y_A = A_i$, $Y_{A+1} = A_{i+1}$ を保存する
9. $F(H(A_{i+1}), CR)$ を生成する
10. サーバから γ を受け取り，生成した $F(H(A_{i+1}))$ と比較する
11. サーバ側から一定時間レスポンスがない場合は再度 1 から処理を行う．ただし，新しいチャレンジを受け取った時の α および β を生成する場合は $A_i = Y_A$，$A_{i+1} = Y_{A+1}$ とする
12. 一致すれば認証成功こなり，A_i を A_{i+1} に更新して終了する

- サーバ

1. ユーザからリクエストを受け取ると CR を用いて乱数 $CR = CR \oplus (CR + Y_A)$ を生成する
2. 生成した CR をユーザに送信する
3. ユーザから送られた β，と保存してある A_i を用いて $H(A_{i+1})$ を算出する
4. 保存してある A_i と CR，送られた α を用いて A_{i+1} を算出する
5. 算出した A_{i+1} から $H(A_{i+1})$ を生成し，β から算出した $H(A_{i+1})$ と比較する．一致しなければ 6 以下の処理を行う．一致すれば認証が成立となり 4 以降の処理を行う．
6. 次回の同期ズレに備えて Y_A を A_i に更新し，10 の処理を行う
7. 認証不正立の場合は前回認証情報である Y_A を用いて β から $H(A_{i+1})$ を算出する
8. 算出した $H(A_{i+1})$ と α を元に A_{i+1} を算出する
9. 算出した A_{i+1} から認証情報を生成し，$H(A_{i+1})$ と比較する．一致すれば以下の処理を行う
10. A_i を A_{i+1} に更新する
11. $\gamma = F(H(A_{i+1}))$ を生成し，ユーザに送信し終了する

3.2　藤田の方式 (Challenge&Response 方式)

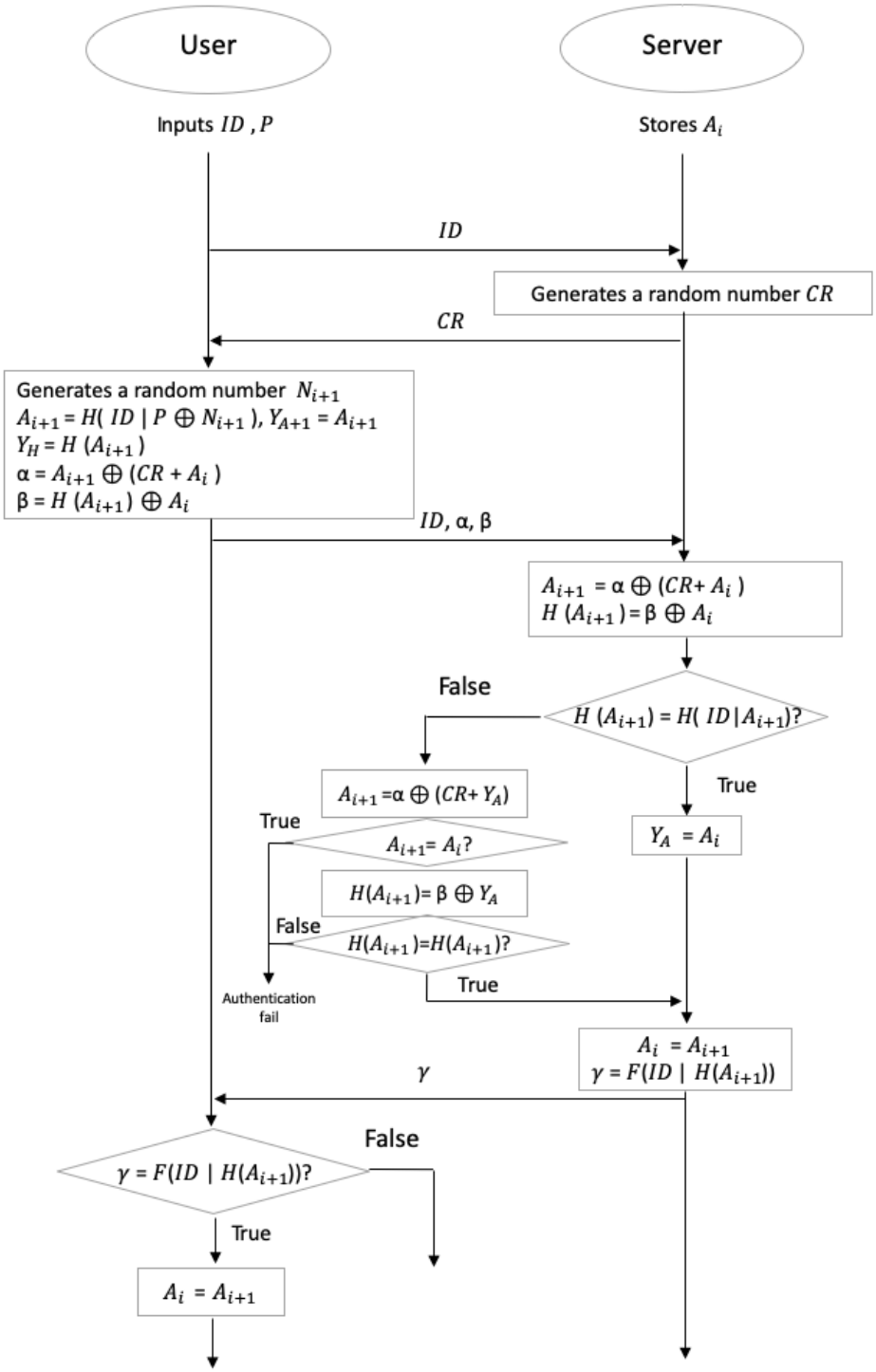

図 3.3　藤田方式の i 回目認証フェーズ

3.3　課題

以下に藤田の方式の課題を示す．藤田の方式では C&R 方式を用いることで同期問題解決時のユーザ側のなりすましを防ぐことが可能である．しかし，通信路を流れるチャレンジの値とサーバ側が返す γ の値を盗聴することでサーバ側になりすますことが可能となる．以下に攻撃者がサーバ側のなりすましについて説明する．

藤田の方式ではサーバから送られるチャレンジに対してユーザはチャレンジを組み込んだ情報をサーバに送信する．サーバ受け取った情報を用いて認証を行い，ユーザに γ を送る．この時，通信路上を流れる情報を以下に示す．

- $\alpha = G \oplus (C + A_{i+1})$
- $\beta = H(G) \oplus A_{i+1}$

藤田の方式では同期問題が発生した場合は再度チャレンジをリクエストするところから開始され，今回認証情報及び次回認証情報は同期ズレが起きていない時に生成したものを使用する．そのため以下の情報が通信路を流れる．

- $\alpha = G \oplus (D + A_{i+1})$
- $\beta = H(G) \oplus A_{i+1}$

つまり，攻撃者はチャレンジ D とそれに伴う α と β をユーザに送信することでなりすましが可能である．これは同期問題が発生した回に通信路上を流れる情報と同じ形をとる．以上のことから藤田の方式ではサーバ側のなりすましが可能であると言える．

第 4 章

提案方式

本章では，藤田の方式の課題を踏まえて，同期問題解決手法の安全性を確保するための拡張を行う．まず，研究の目的について説明する．その後，提案方式の説明を行い，SAS-L に適用させた場合の説明を行う．

4.1 目的

SAS を導入する際に起こりうる同期問題を解決し，安全性を満たすことを目的として既存方式の課題であるなりすましに耐性のある方式を提案する．また，実際に提案方式の実行時間を測定し，同期問題対策による拡張の影響が限りなく少ないことを示す．

4.2 概要

既存方式では同期問題を解決し，クライアントへのなりすましに対して耐性があることが示された．しかし，依然としてサーバ側のなりすましの脅威があることから対策が必要である．そこで，クライアント側でチャレンジの値をリストで保持し，サーバ側 γ を生成する際にチャレンジの値を引数に増やす方式を提案する．以下で提案方式の説明を行う．本方式は前述の方式と同様のフェーズで構成される．以降で用いられる表記は藤田の方式と同様であるため割愛する．

4.2.1　定義

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．

用語・記号	定義
Y_A	バックアップした今回認証情報
Y_{A+1}	バックアップした次回認証情報
CR	チャレンジ情報
L	リスト

表 4.1　提案方式で用いる用語の定義

4.2.2　認証フェーズ

i 回目認証フェーズを図 4.1 に示す．

- クライアント
  1. サーバ側にチャレンジをリクエストする
  2. 受け取ったチャレンジ CR がリストに記録された値と一致しないことを確認する
  3. 保存してある乱数 N_i と ID，P と共に今回認証情報 $A_i = H(ID|P \oplus N_i)$ を生成する
  4. 新たに乱数 N_{i+1} を生成し，ID と P と共に次回認証情報 $A_{i+1} = H(ID|P \oplus N_{i+1})$ を生成する
  5. 生成した A_{i+1} から $H(A_{i+1})$ 生成する
  6. サーバから受け取った CR と A_i，A_{i+1}，$H(A_{i+1})$ を用いて $\alpha = A_{i+1} \oplus (CR + A_i)$ と $\beta = H(A_{i+1}) \oplus A_i$ を生成する
  7. ID と α，β をサーバに送信する
  8. List に CR を保存する
  9. $Y_A = A_i$, $Y_{A+1} = A_{i+1}$ を保存する

10. $F(H(A_{i+1}), CR)$ を生成する
11. サーバから γ を受け取り，生成した $F(H(A_{i+1}), CR)$ と比較する
12. サーバ側から一定時間レスポンスがない場合は再度 1 から処理を行う．ただし，新しいチャレンジを受け取った時の α および β を生成する場合は $A_i = Y_A$，$A_{i+1} = Y_{A+1}$ とする
13. 一致すれば認証成功となり，A_i を A_{i+1} に更新し，List を初期化して終了する

- サーバ

1. クライアントからリクエストを受け取ると CR を用いて乱数を新たな $CR = CR \oplus (CR + Y_A)$ を生成する
2. 生成した CR をクライアントに送信する
3. クライアントから送られた β と保存してある A_i を用いて $H(A_{i+1})$ を算出する
4. 保存してある A_i と CR，送られた α を用いて A_{i+1} を算出する
5. 算出した A_{i+1} から $H(A_{i+1})$ を生成し，β から算出した $H(A_{i+1})$ と比較する．一致しなければ 6 以下の処理を行う．一致すれば認証が成立となり 4 以降の処理を行う．
6. A_i を A_{i+1} に更新して終了する
7. 次回の同期ズレに備えて Y_A を A_i に更新し，10 の処理を行う
8. 認証不正立の場合は前回認証情報である Y_A を用いて β から $H(A_{i+1})$ を算出する
9. 算出した $H(A_{i+1})$ と α を元に A_{i+1} を算出する
10. 算出した A_{i+1} から認証情報を生成し，$H(A_{i+1})$ と比較する．一致すれば以下の処理を行う
11. 次回の同期ズレに備えて Y_A を A_i に更新する
12. $\gamma = F(H(A_{i+1}), CR)$ を生成し，クライアントに送信する

4.2　概要

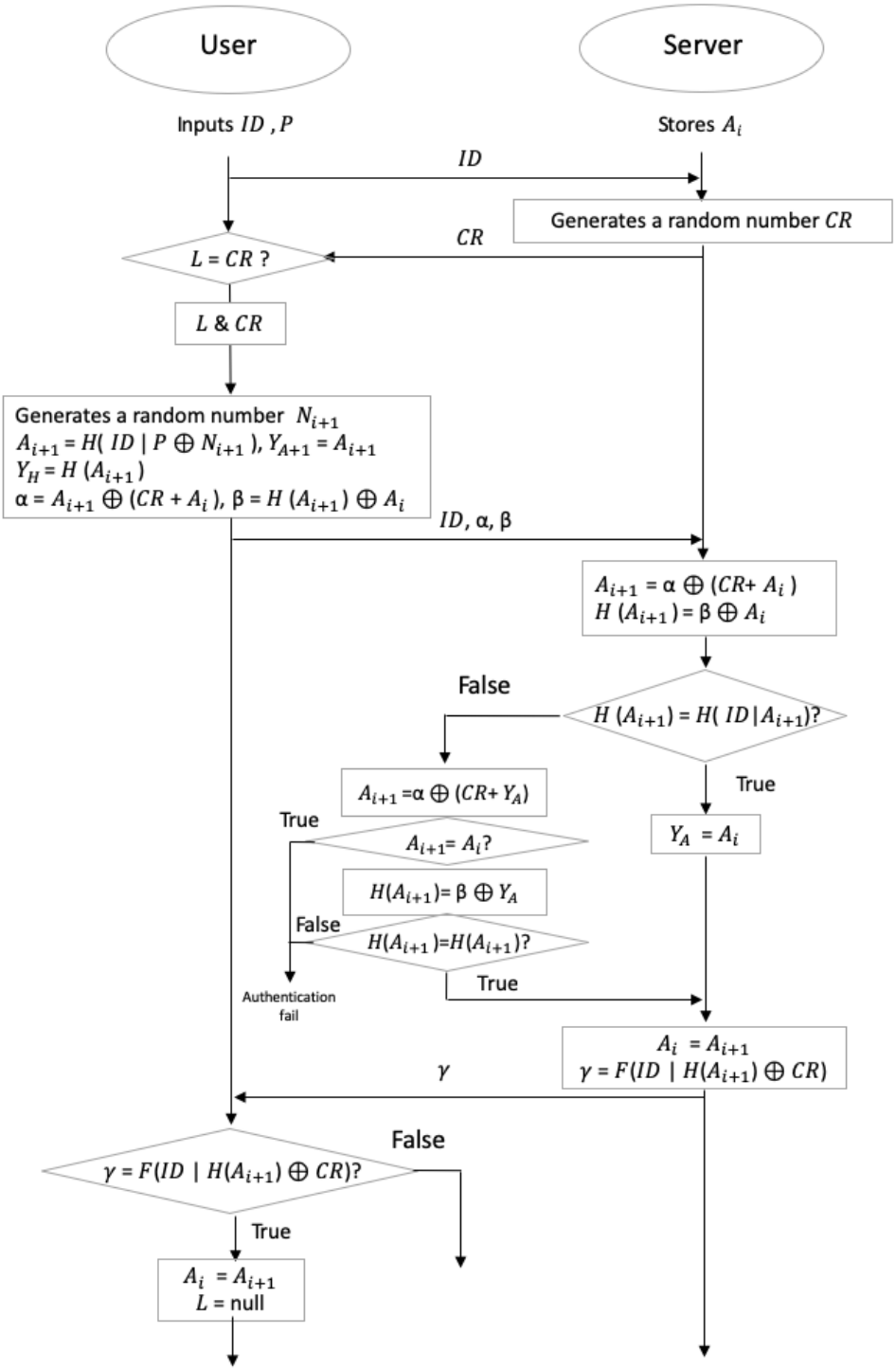

図 4.1　提案方式の i 回目認証フェーズ

4.3　提案方式を SAS-L に適用

既存方式と提案方式は共に SAS-2 をベースに提案されている．そこで，提案方式を SAS-L に導入できるように拡張する．

4.3.1　定義

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．

用語・記号	定義
M_1	初回マスク値
M_i	i 回目セッションで用いるマスク値
Y_A	前回認証情報のバックアップ
Y_M	前回マスク値のバックアップ
Y_{A+1}	次回認証情報のバックアップ
M_{i+1}	次回マスク値
CR	チャレンジ情報
L	リスト

表 4.2　SAS-L に提案方式を導入する際の用語の定義

4.3.2　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を用いて行う．初回登録フェーズを図 4.2 に示す．

- ユーザ
  1. 乱数 N_1 を生成し，ID と P と共に初回認証情報 $A_1 = H(ID|P \oplus N_1)$ を生成する
  2. チャレンジの元となる乱数 CR とマスク値 M_1 を生成する
  3. 生成した ID と A_1,M_1,CR を安全な通信路を用いてサーバに送信する

4. 送信した N_1, M_1 を保存する

- サーバ

1. ユーザから送られた A_1,CR を ID に紐付けて保存する
2. さらにバックアップ情報 $Y_A = A$ を保存する

4.3.3　認証フェーズ

i 回目認証フェーズを図 4.3 に示す.

- ユーザ

1. 受け取ったチャレンジ CR が既存の値でないことを確認する
2. 保存してある乱数 N_i と ID，P と共に今回認証情報 $A_i = H(ID|P \oplus N_i)$ を生成する
3. 新たに乱数 N_{i+1} を生成し，ID と P と共に次回認証情報 $A_{i+1} = H(ID|P \oplus N_{i+1})$ を生成する
4. サーバから受け取った CR と A_i，A_{i+1}，M_i を用いて $\alpha = A_{i+1} \oplus (CR + A_i) \oplus M_i$ と $\beta = A_{i+1} + A_i + M_i$ を生成する
5. ID と α，β をサーバに送信する
6. L に CR を保存する
7. $Y_A = A_i$, $Y_M = M_i$ を保存する
8. $M_{i+1} = A_{i+1} + M_i$ を生成する
9. サーバから γ を受け取り，生成した $M_{i+1} \oplus (A_i + CR)$ と比較する
10. サーバ側から一定時間レスポンスがない場合は再度 1 から処理を行う．ただし，新しいチャレンジを受け取った時の α および β を生成する場合は $A_i = Y_A$，$M_i = Y_M$，$A_{i+1} = Y_{A+1}$ とする
11. 一致すれば認証成功こなり，A_i を A_{i+1} に更新し，L を初期化して終了する

- サーバ

4.3 提案方式を SAS-L に適用

1. CR を用いて乱数を新たな $CR = CR \oplus (CR + Y_A)$ を生成する
2. 生成した CR と ID をユーザに送信する
3. ユーザから送られた α と保存してある A_i，M_i，CR を用いて A_{i+1} を算出する
4. 算出した A_{i+1} と保存してある A_i，M_i を用いて $A_{i+1} + A_i + M_i$ を生成し，β と比較する．一致しなければ 6 以下の処理を行う．一致すれば認証が成立となり 5 以降の処理を行う．
5. $Y_A = A_i$，$Y_M = M_i$ に更新し，10 以下の処理を行う．
6. 認証不正立の場合は前回認証情報である Y_A を用いて α から A_{i+1} を算出する
7. 算出した A_{i+1} と保存してある A_i，M_i を元に $A_{i+1} + A_i + M_i$ を生成する
8. 算出した $A_{i+1} + A_i + M_i$ と β を比較する．一致すれば以下の処理を行う
9. $M_{i+1} = A_{i+1} + M_i$ を生成する
10. $\gamma = M_{i+1} \oplus (A_i + CR)$ を生成し，ユーザに送信する
11. 次回の同期ズレに備えて $A_i = A_{i+1}$，$M_i = M_{i+1}$ に更新して終了する

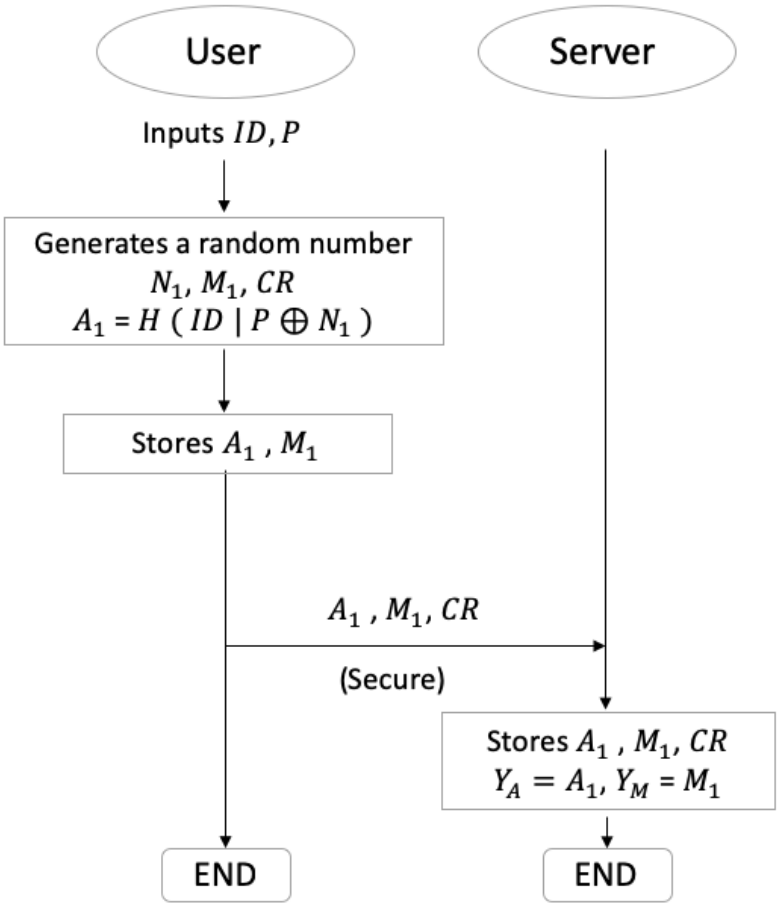

図 4.2　提案方式導入後の SAS-L の初回登録フェーズ

4.3　提案方式を SAS-L に適用

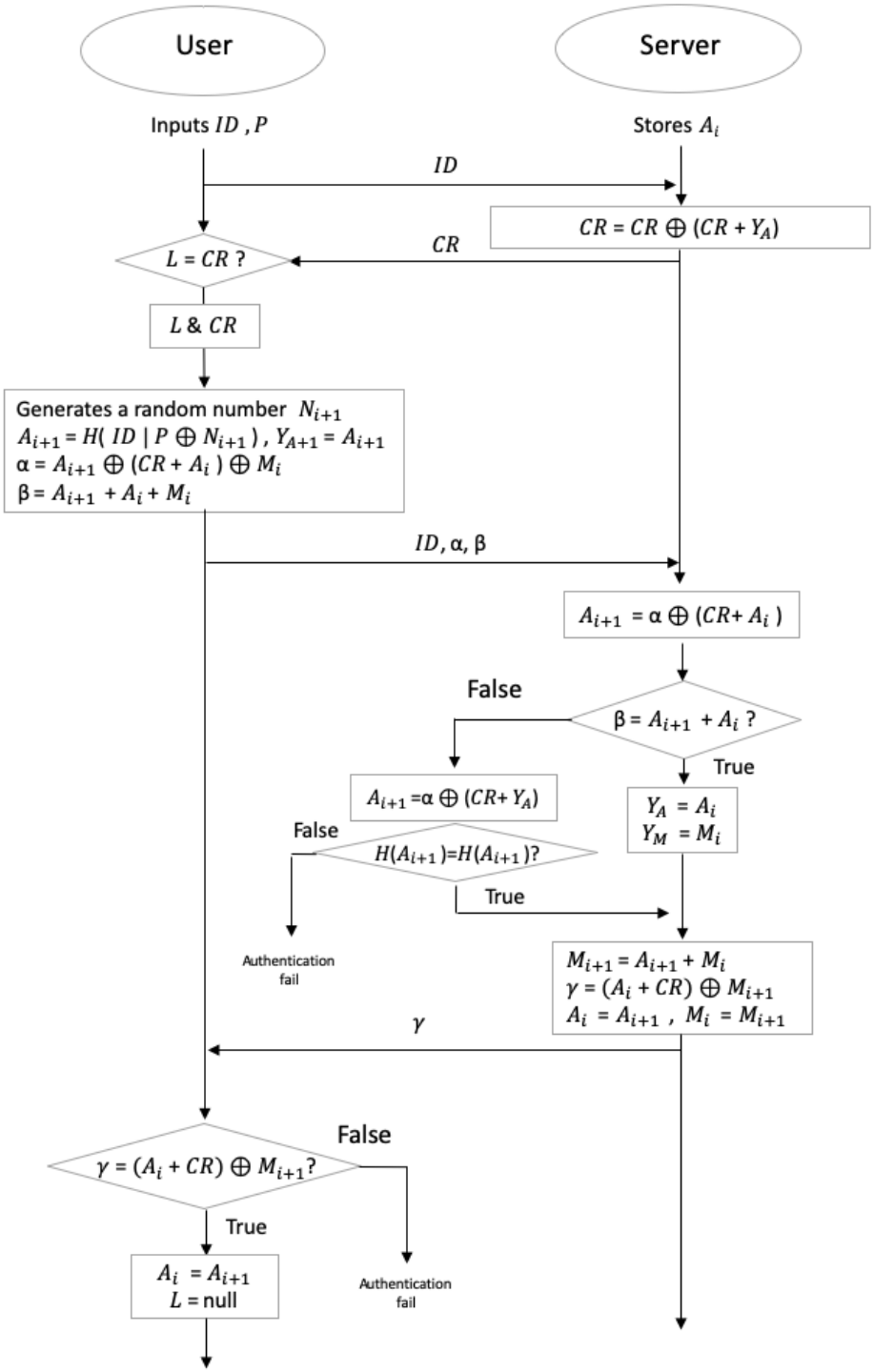

図 4.3　提案方式導入後の SAS-L の i 回目認証フェーズ

第 5 章

実験環境

本稿では実験環境について述べる．

5.1 実験の目的

SAS を導入する場合，可用性の観点から同期問題対策手法は必須である．同期問題対策手法は既存の SAS と比較して処理工程が増加する．したがって，導入後の処理工程の増加が処理全体にどれほど影響しているかを検討する必要がある．そこで本実験では提案方式と既存方式の処理時間を計測することで拡張がどのような影響を与えるか検討する．

5.1.1 実験内容

本実験では，サーバ側を処理能力のある PC とし，ユーザ側を処理能力の低い Raspberry Pi ZERO とした．以下にデバイスの情報を示す．本実験では 10 回実行した時の処理時間の平均を比較する．

5.1.2 実験環境

以下に端末情報と測定範囲を示す．言語は Python を用いて実装した．

	ユーザ側
端末	Raspberry Pi ZERO WH
メモリ	512MB RAM
OS	Raspbian
CPU	160MHz

表 5.1　ユーザ側環境

	サーバ側
端末	MacBook Pro
メモリ	8 GB 1867 MHz DDR3
OS	Mac OSX
CPU	2.7GHz

表 5.2　サーバ側環境

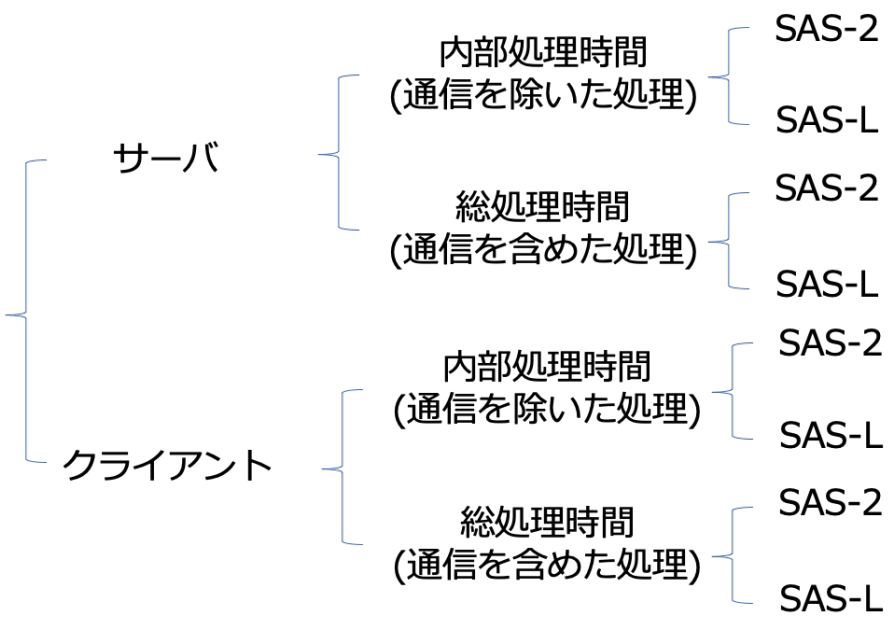

図 5.1　比較方式

第 6 章

評価

本稿では提案方式の安全性と測定時間の評価について述べる．まず，提案方式の各通信路が遮断された場合を想定して安全性の評価を行う．そして，実験で測定したデータを元に処理時間の観点から拡張の影響を評価する．

6.1 安全性評価

SAS を利用する際に通信路を流れる情報は攻撃者が入手することが可能である．SAS の安全性は攻撃者が通信路上を流れる情報を入手したとしても認証に用いる情報を作成することが困難であることから成り立っている．SAS において攻撃者が入手可能な情報は，ID と α，β，γ である．藤田の方式および提案方式ではこれに加えてチャレンジの値である CR を入手される．これらの情報が入手された場合でも提案方式が安全であることを各通信経路が意図的に偽装および遮断されることを想定することで評価する．

6.1.1 サーバに ID を送信した場合 (通常時)

攻撃者がユーザになりすましてサーバに正規のユーザの ID を送信した場合を考える．サーバ側では CR を生成し，攻撃者に送信する．攻撃者は前回の通信で取得した ID と α，β をサーバに送信する．サーバ側では受け取った ID と α，β から攻撃者の認証を試みる．しかし，攻撃者が取得した α，β はその回で送られた CR を元に生成されているため認証が失敗する．したがって攻撃者がユーザになりすますことは不可能である．

6.1.2　ユーザに偽造した CR を送信した場合 (通常時)

攻撃者がサーバ側になりすましてユーザに任意のチャレンジ値 CR を送信する場合を考える．正規のユーザは CR を受け取るとリストに含まれているかを確認する．前回の認証が正常に動作した場合は認証情報と共にリストが更新されているため，新たに CR をリストに追加する．正規のユーザは受け取った CR を元に α，β を生成し，ID と共に攻撃者に送信する．攻撃者は前回の通信で取得した γ をユーザに送信する．ユーザは γ を元に攻撃者の認証を試みるが，γ は今回の次回認証情報を元に生成されているため認証が成立しない．従って安全である．

6.1.3　α，β を再利用した場合

(通常時) 攻撃者が通信路上の α，β を用いて攻撃を行う場合を考える．攻撃者は前回の通信から取得した α，β をサーバ側に送信する．サーバ側は保持している認証情報と受け取った情報を元に認証を行う．しかし，α，β にはその回の CR が含まれており，α，β 内の CR のみを変更することは困難である．従って，認証が成立しないため安全である．

6.1.4　γ を再利用した場合 (通常時)

攻撃者が γ を再利用した場合を考える．攻撃者は前回の通信から γ を取得しており，ユーザに対して前回の γ を送り，サーバ側になりすます場合を考える．サーバ側で生成する γ には今回の通信で用いられる次回認証情報に加えて CR の値も組み込まれており，毎回異なる．従って認証が成立しないため安全である．

正常時と変化する部分の処理における安全性の検証を行う．全ての通信において相手側にパケットが何らかの理由で到達しなかった場合は同期問題が発生したとみなしてステップ 1 からやり直す．

CR が到達しなかった場合 (同期問題発生時)

攻撃者が新たな CR を受け取った場合でも前回の α，β と今回の CR から今回の α，β を生成することが困難であるため安全である．

α，β が到達しなかった場合 (同期問題発生時)

攻撃者が今回の α，β を取得し，ユーザが新たに ID をサーバに送るより前に α，β をサーバに送信した場合には認証が成立してしまう．これは SAS のどのバージョンおよび同期問題解決方式でも解決できない．

γ が到達しなかった場合 (同期問題発生時)

攻撃者はサーバからユーザに送られる γ を遮断し，同期問題を引き起こすことが可能である．ユーザはタイムアウトになるまで γ が到達することを待ち，タイムアウトになると一から認証を再開する．同期問題が発生した場合でも γ の値は毎回変化する．攻撃者は前回の CR とその CR を元に生成した γ を保持することが可能である．そこで ID がユーザから送信された場合に前回 CR を送信することが考えられる．しかし，リストと比較し同じ CR を用いることが不可能であるため，認証が失敗する．

6.2　安全性評価のまとめ

既存方式では攻撃者が意図的に同期問題を引き起こすことでサーバ側になりすますことが可能である．提案方式では γ の生成時にその回で用いられる CR の値を含めることで防止した．さらに，前回の CR と γ が利用された場合を想定してリストを用い，認証情報が更新されるまでの間に利用された CR を再度利用できないように拡張した．これにより，同期問題を解決すると共になりすましへの対策を実現した．

6.3　処理時間計測結果

以下に測定結果を示す．サーバ側の内部処理時間比較 6.1 では SAS-2 は提案方式が既存方式の 1.5 倍，SAS-L が 1.92 倍の処理時間となった．ユーザ側の内部処理時間比較 6.2 では SAS-2 が 1.01 倍，SAS-L が 1.03 倍となった．サーバ側の総処理時間比較 6.3 では SAS-2 は提案方式が既存方式の 3.01 倍，SAS-L は 26.59 倍となった．ユーザ側の総処理時間比較 6.4 では SAS-2 は提案方式が既存方式の 1.06 倍，SAS-L の 2.16 倍となった．

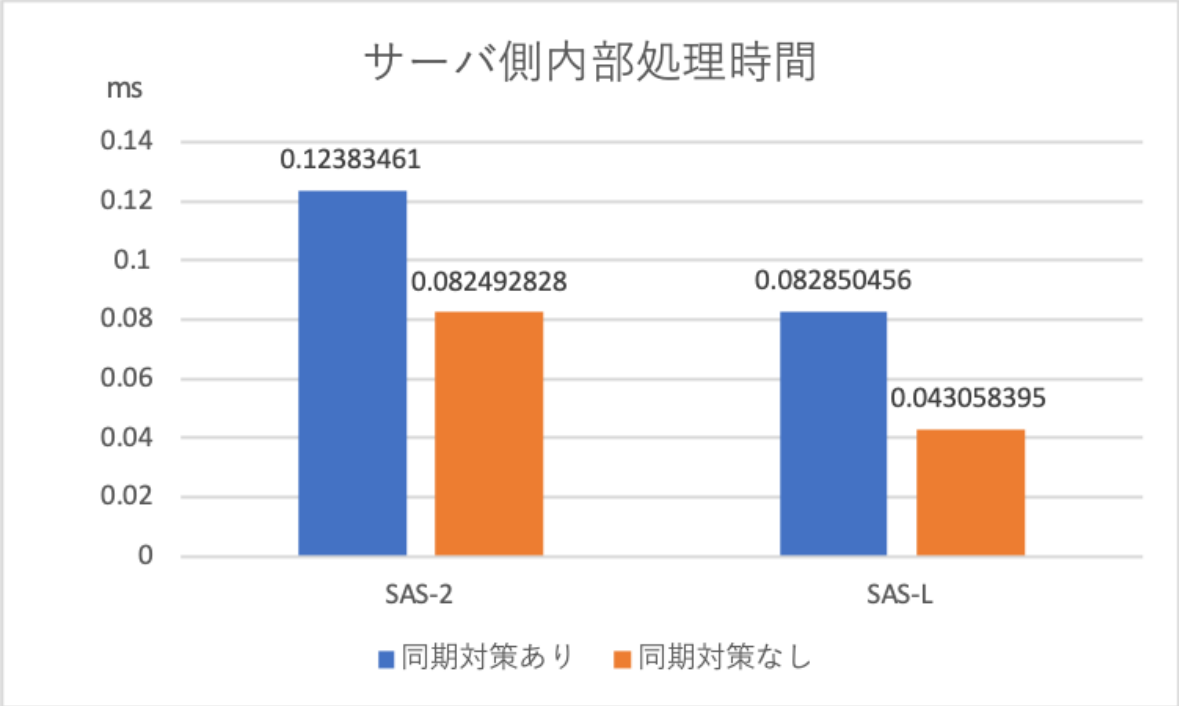

図 6.1　サーバ側内部処理 (通信を除く) 時間比較

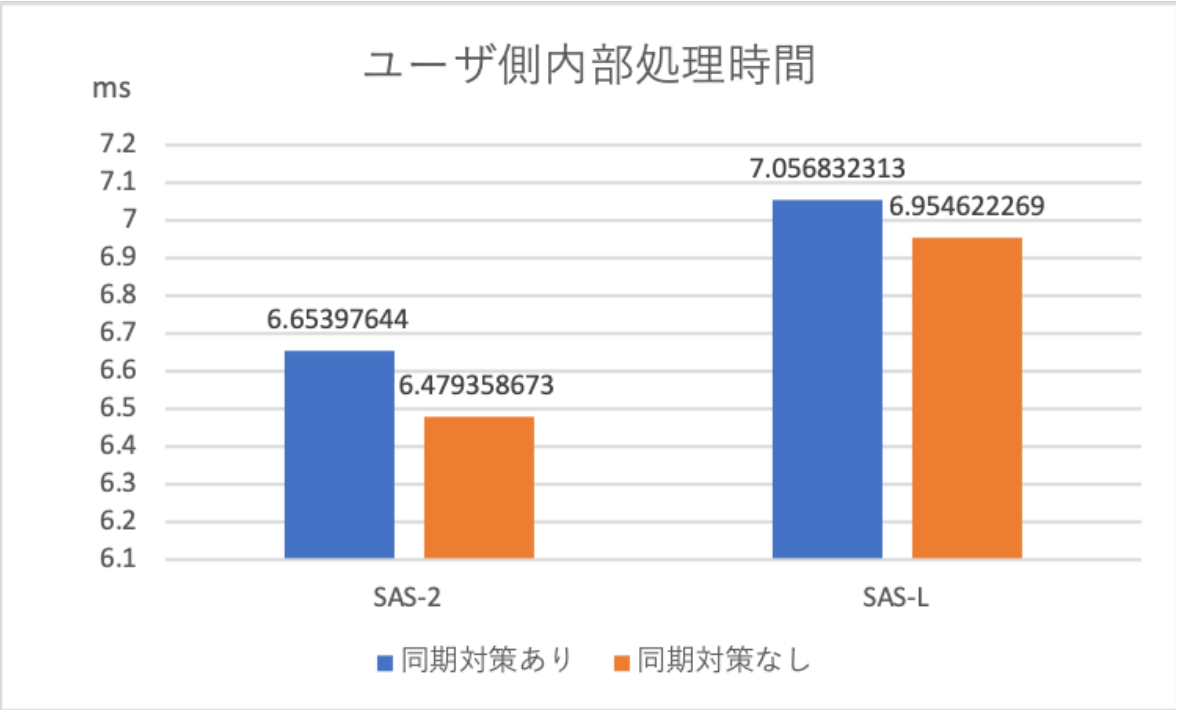

図 6.2　ユーザ側内部処理 (通信を除く) 時間比較

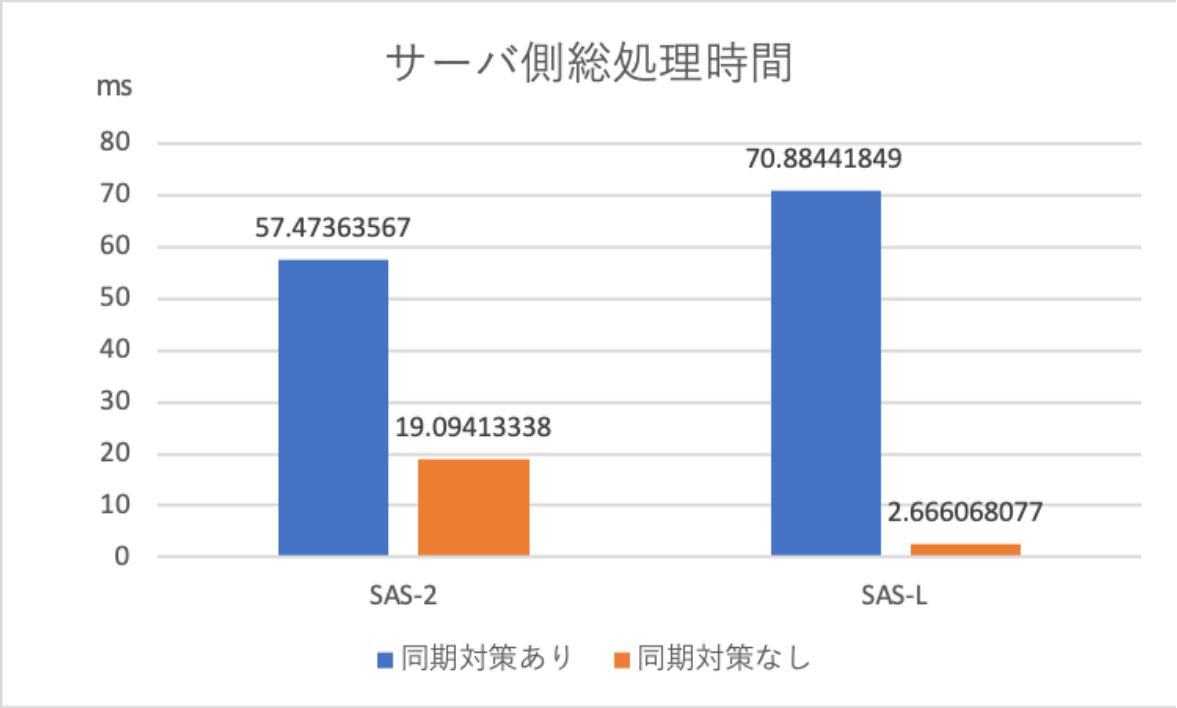

図 6.3　サーバ側総処理 (通信を含める) 時間比較

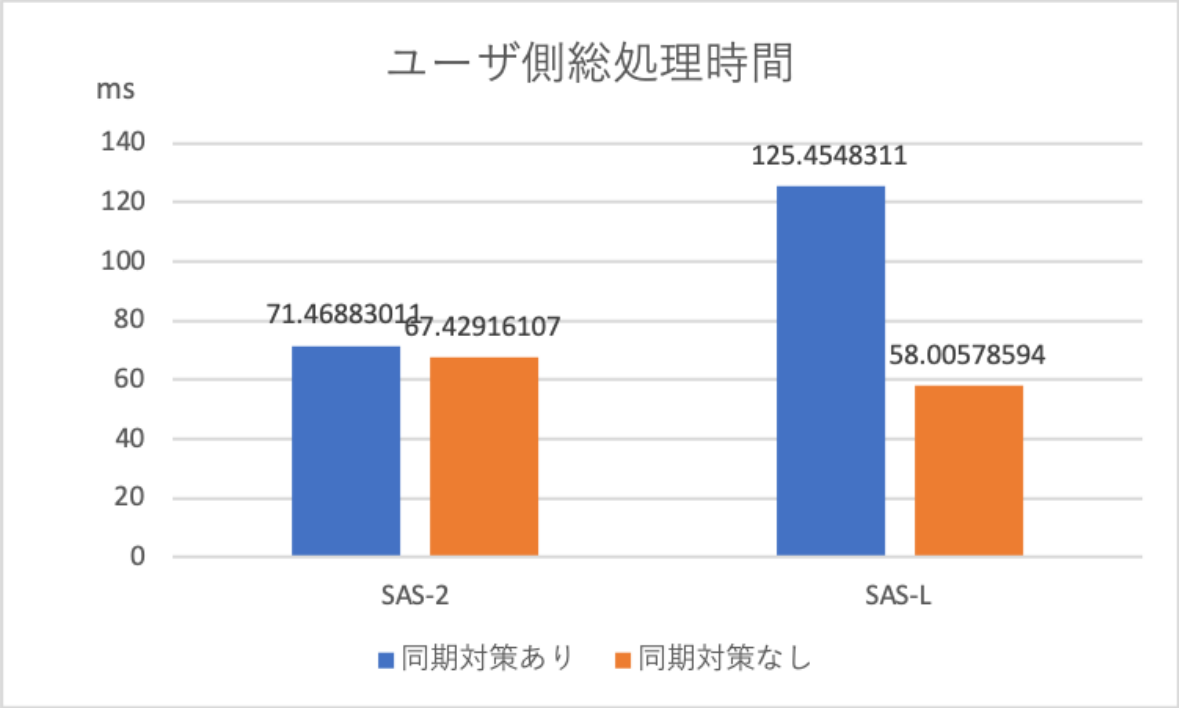

図 6.4　ユーザ側総処理 (通信を含める) 時間比較

6.3.1　内部処理時間比較

サーバ側の内部処理比較において処理時間の増加量が SAS-2 では 50%と SAS-L では 92%増加となった．この原因として SAS-2 では内部処理に一方向性関数の実行が含まれており，拡張部分が一方向性関数を含めたその他の処理と比較して処理時間が短いため割合として増加量が少なくなったと考えられる．一方で SAS-L はサーバ側の内部処理に一方向性関数の処理がないため，加算などの演算子一つあたりの処理が処理時間全体のうちの大きな割合を占めている．これらのことから同期問題対策として拡張した場合のサーバ側への影響は一方向性関数処理より小さく演算子数回分であることがわかる．したがって，拡張による影響が限りなく少ないと言える．ユーザ側の内部処理比較においては増加量が SAS-2 で 3%，SAS-L では 1%の増加となった．この要因はサーバ側と同様でユーザ側は SAS-2 と SAS-L 共に，一方向性関数を数回行うため，全体の大部分をこの一方向性関数の処理時間が占めている．これらのことからユーザ側においても増加量が少ないため拡張による影響が限りなく少ないと言える．

6.3.2　総処理時間比較

総処理時間の比較ではサーバとユーザで数倍から数十倍の増加が見られた．この原因は通信の処理時間が全体の処理時間の大部分を占めているからである．内部処理時間は多くて数ミリ秒程度であるのに対して総処理時間では数十ミリ秒から数百ミリ秒となっている．従って提案方式は通信の回数が増加するため全体の処理時間への影響が増加すると言える．

6.4　処理時間における評価

提案方式は通信回数に伴う通信処理時間の影響を受けることから即時性が求められる環境には適さない．しかし，内部処理の処理負荷は数ミリ秒と小さいため内部処理負荷を下げることを目的にサーバの接続端末数が多い場合に有効であると言える．

第 7 章

結論

IoT デバイスの種類は多様化しており，処理性能が低いものも存在する．このようなデバイスに適したセキュリティ技術として SAS が提案されている．SAS は共通鍵をベースとし，相互認証・鍵配送・暗号通信を同時に実現可能である．しかし，SAS を導入した際に，通信経路を遮断することで端末間の認証情報にズレが生じることで可用性が損失する危険性がある (同期問題)．これは意図的に発生させることが可能であり SAS の脅威となるため対策が必要である．

この同期問題を解決する手法として中原らの方式がある．中原らの方式では認証情報をバックアップすることで拡張に影響を最小限にしつつ可用性を担保した．しかし，リプレイ攻撃への対策が課題である．この課題を解決する手法としてチャレンジ&レスポンス方式を導入した藤田の方式がある．この方式は可用性を担保し，ユーザ側のなりすましを防ぐ方式となっている．しかし，サーバ側のなりすましが可能であることが課題である．

本稿では藤田の方式の課題を解決する方式を提案すると共に同期対策による拡張が内部処理 (通信を除く) 時間に影響しないこと示した．しかし，通信回数が増加しており，通信を含めた総処理時間が増加することも明らかとなった．そこで，提案方式の適用箇所としては内部処理負荷を下げることを目的にサーバ側の接続端末数が多い場合に有効であると言える．

今後の課題として，現在一般的に利用されている TLS の内部処理と比較して SAS の有用性を示すと共に適用範囲の検討を行うこと等があげられる．

謝辞

研究を行うにあたって，研究の指導や論文作成に関してご指導いただきました高知工科大学情報 学群 清水明宏副学長に心より感謝申し上げます．また，本研究の副査を担当していただきました高知工科大学情報学群 敷田 幹文教授，植田 和憲講師に深く御礼申し上げます．

また，有益な議論を交わしていただいた高知工科大学 清水研究室の関係者各位に深く感謝致します．特に，学部 4 年の池内氏には自身の研究にも取り組みながら有益な意見交換と議論をして頂きました．また，以西氏と稲留氏におかれましても研究室活動においてたくさんの助力を頂きました．心より感謝申し上げます．

最後に大学生活を支えて下さった両親，妹，祖父母に感謝申し上げます．

参考文献

[1] “第一部 第一節　世界と日本の ICT 市場の動向，”　情報通信白書平成 30 年版：世界と日本の ICT，総務省，p. 7，https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/pdf/n1100000.pdf，(参照 2020-1-15).

[2] “IoT セキュリティ総合対策について，”　総務省 情報流通行政局 サイバーセキュリティ課　課長補佐 澤谷 航，平成 30 年 2 月 28 日，https://www.soumu.go.jp/main_content/000543066.pdf，(参照 2020-1-15).

[3] “IoT セキュリティ総合対策プログレスレポート 2019，”　総務省 セキュリティタスクフォース，令和元年 5 月，https://www.soumu.go.jp/main_content/000623344.pdf，(参照 2019-12-18).

[4] E.Rescorla，齋藤　孝道 (訳)，鬼頭　利之 (訳)，古森　貞 (訳)，“マスタリング TCP/IP SSL/TLS 編，” オーム社，2003.

[5] T. Tsuji and A. Shimizu,“A One-Time Password Authentication Method for Low Spec Machines and on Internet Protocols,” IEICE Trans. COMMUN, vol. E87-B, no.6, pp. 1594—1600, 2004.

[6] 小山　峻矢，“IPv6 無線センサネットワークにおける信頼性を考慮したセキュア通信方式，” 函館未来大学修士学位論文, 2019.

[7] 太田 愛里，“IoT に適したワンタイムパスワード認証 方式に関する研究，”　高知工科大学修士学位論文，2018.

[8] 中原　知也，辻　貴介，清水　明宏，“SAS-2 認証方式の同期問題に関する検討，” 電子情報通信学会技術研究報告，OIS，vol.104，no.714，pp.83-87，2005.

[9] 藤田 寛泰，“SAS-2 の同期問題対策時における なりすまし防止に関する研究，”　高知工科大学学士学位論文，2017.

[10] 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，”

オーム社，2012.