SAS-Mizoguchi

Simple And Secure Mutual Authentication Protocol

MIZOGUCHI Koki[1]

Kochi University of Technology

November 17, 2022

KOCHI UNIVERSITY OF TECHNOLOGY

[1]情報セキュリティシステム研究室

概要

特徴

SAS-M（仮）は，SAS-L を基盤として，Client と Server が相互認証する機能を追加した認証プロトコル.

相互認証の必要性

SAS-L では，Server から Client の認証は可能だが，Server から登録されている Client は Server がが正当なものとして通信が行われる．つまり，Client は Server を真には認証していない．Server を真に認証するとこで，重要な情報を送る Client は Server の正当性を確かめることに十分な意味を見出せる.

認証手順

Server 側の処理

Server 生成データ

$$A_i = E_1(\mathrm{SID} \mid S \oplus N_i)$$

$$A_{i+1} = E_1(\mathrm{CID} \mid S \oplus N_{i+1})$$

$$A_{i+2} = E_1(\mathrm{CID} \mid S \oplus N_{i+2})$$

初回認証情報

$$A_i \oplus A_{i+1}$$

1.Secure to Client

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big)$$

$$\beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

2 to Client

送信用データ

[1]SID：サーバ固有 ID
[2]CID：クライアント固有 ID

Server 側の処理

Server 生成データ

$$A_i = E_1(\mathrm{SID} \mid S \oplus N_i)$$

$$A_{i+1} = E_1(\mathrm{CID} \mid S \oplus N_{i+1})$$

$$A_{i+2} = E_1(\mathrm{CID} \mid S \oplus N_{i+2})$$

初回認証情報

$$A_i \oplus A_{i+1}$$

1.Secure to Client

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big)$$

$$\beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

2 to Client

送信用データ

[1]SID：サーバ固有 ID
[2]CID：クライアント固有 ID

Server 側の処理

Server 生成データ

$$A_i = E_1(\mathrm{SID} \mid S \oplus N_i)$$

$$A_{i+1} = E_1(\mathrm{CID} \mid S \oplus N_{i+1})$$

$$A_{i+2} = E_1(\mathrm{CID} \mid S \oplus N_{i+2})$$

初回認証情報

$$A_i \oplus A_{i+1}$$

1.Secure to Client

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big)$$

2 to Client

$$\beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

送信用データ

[1]SID：サーバ固有 ID
[2]CID：クライアント固有 ID

Server 側の処理

Server 生成データ

$$A_i = E_1(\mathrm{SID} \mid S \oplus N_i)$$

$$A_{i+1} = E_1(\mathrm{CID} \mid S \oplus N_{i+1})$$

$$A_{i+2} = E_1(\mathrm{CID} \mid S \oplus N_{i+2})$$

初回認証情報

$$A_i \oplus A_{i+1}$$

1.Secure
to Client

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big)$$

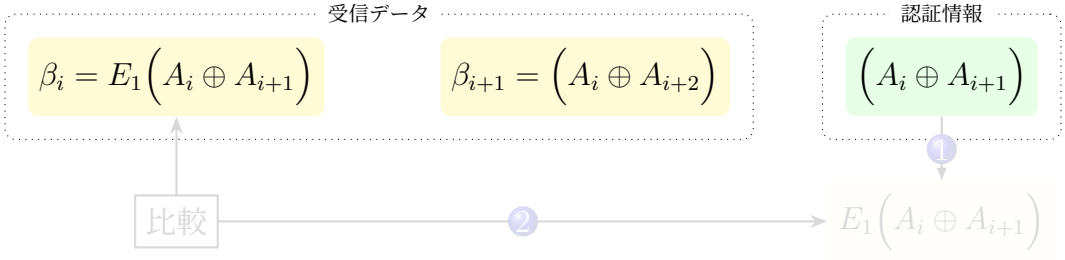
$$\beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

2
to Client

送信用データ

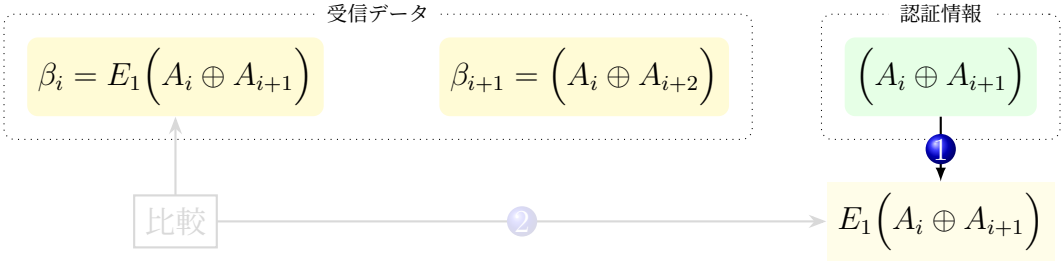
[1]SID：サーバ固有 ID
[2]CID：クライアント固有 ID

Client 側の処理



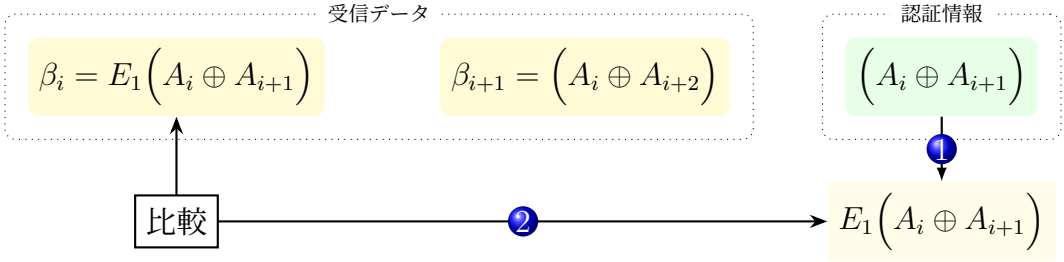
1. 認証情報に一方向ハッシュ関数を施す.
2. 比較 で Client が Server を検証する.
   - 不一致の場合，認証不成立.（Server が不正である可能性あり）

Client 側の処理



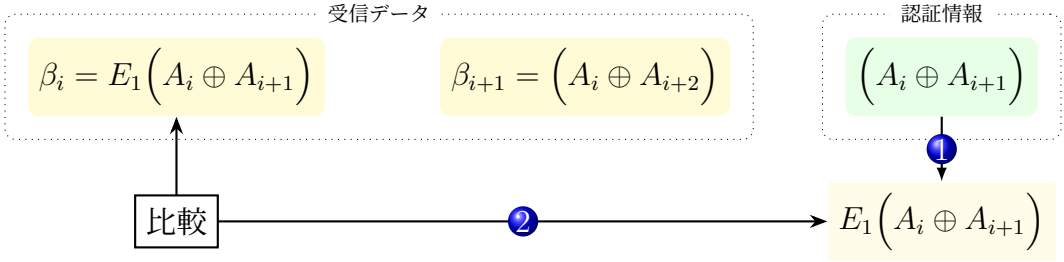
1. 認証情報に一方向ハッシュ関数を施す.
2. 比較 で Client が Server を検証する.
   - 不一致の場合，認証不成立.（Server が不正である可能性あり）

Client 側の処理



1. 認証情報に一方向ハッシュ関数を施す.
2. 比較 で Client が Server を検証する.
   - 不一致の場合，認証不成立.（Server が不正である可能性あり）

Client 側の処理



1. 認証情報に一方向ハッシュ関数を施す.
2. 比較 で Client が Server を検証する.
   - 不一致の場合，認証不成立.（Server が不正である可能性あり）

Client 側の処理

受信データ

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big) \qquad \beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

認証情報

$$\Big(A_i \oplus A_{i+1}\Big)$$

$$\beta_{i+1} \oplus \Big(A_i \oplus A_{i+1}\Big) = \Big(A_i \oplus A_{i+2}\Big) \oplus \Big(A_i \oplus A_{i+1}\Big) \quad (3)$$

$$= A_{i+1} \oplus A_{i+2}$$

3. 次回認証情報, $A_{i+1} \oplus A_{i+2}$ を Client に保存.
4. $\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成.
5. γ_i を Server へ送信.

Client 側の処理

受信データ

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big) \qquad \beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

認証情報

$$\Big(A_i \oplus A_{i+1}\Big)$$

$$\beta_{i+1} \oplus \Big(A_i \oplus A_{i+1}\Big) = \Big(A_i \oplus A_{i+2}\Big) \oplus \Big(A_i \oplus A_{i+1}\Big) \quad (3)$$

$$= A_{i+1} \oplus A_{i+2}$$

3. 次回認証情報, $A_{i+1} \oplus A_{i+2}$ を Client に保存.
4. $\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成.
5. γ_i を Server へ送信.

Client 側の処理

受信データ

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big) \qquad \beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

認証情報

$$\Big(A_i \oplus A_{i+1}\Big)$$

$$\beta_{i+1} \oplus \Big(A_i \oplus A_{i+1}\Big) = \Big(A_i \oplus A_{i+2}\Big) \oplus \Big(A_i \oplus A_{i+1}\Big) \quad (3)$$

$$= A_{i+1} \oplus A_{i+2}$$

3. 次回認証情報, $A_{i+1} \oplus A_{i+2}$ を Client に保存.
4. $\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成.
5. γ_i を Server へ送信.

Client 側の処理

受信データ

$$\beta_i = E_1\Big(A_i \oplus A_{i+1}\Big) \qquad \beta_{i+1} = \Big(A_i \oplus A_{i+2}\Big)$$

認証情報

$$\Big(A_i \oplus A_{i+1}\Big)$$

$$\beta_{i+1} \oplus \Big(A_i \oplus A_{i+1}\Big) = \Big(A_i \oplus A_{i+2}\Big) \oplus \Big(A_i \oplus A_{i+1}\Big) \quad (3)$$

$$= A_{i+1} \oplus A_{i+2}$$

3. 次回認証情報, $A_{i+1} \oplus A_{i+2}$ を Client に保存.
4. $\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成.
5. γ_i を Server へ送信.

Server 側の処理

受信データ

$$\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$$

生成済みデータ

A_i A_{i+1} A_{i+2}

6. 生成済みデータから，$E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成する．
7. 受信データ γ_i と比較する．
   - 不一致の場合，認証不成立．（Client が不正である可能性あり．）
   - 一致した場合，認証成立．
8. γ_i を利用した共通鍵暗号方式で通信を行う．

Server 側の処理

受信データ

$$\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$$

生成済みデータ

A_i A_{i+1} A_{i+2}

6. 生成済みデータから，$E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成する．
7. 受信データ γ_i と比較する．
   - 不一致の場合，認証不成立．（Client が不正である可能性あり．）
   - 一致した場合，認証成立．
8. γ_i を利用した共通鍵暗号方式で通信を行う．

Server 側の処理

受信データ

$$\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$$

生成済みデータ

A_i A_{i+1} A_{i+2}

6. 生成済みデータから，$E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成する．
7. 受信データ γ_i と比較する．
   - 不一致の場合，認証不成立．（Client が不正である可能性あり．）
   - 一致した場合，認証成立．
8. γ_i を利用した共通鍵暗号方式で通信を行う．

Server 側の処理

受信データ

$$\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$$

生成済みデータ

A_i A_{i+1} A_{i+2}

6. 生成済みデータから，$E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成する．
7. 受信データ γ_i と比較する．
   - 不一致の場合，認証不成立．（Client が不正である可能性あり．）
   - 一致した場合，認証成立．
8. γ_i を利用した共通鍵暗号方式で通信を行う．

Server 側の処理

受信データ

$$\gamma_i = E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$$

生成済みデータ

A_i A_{i+1} A_{i+2}

6. 生成済みデータから，$E_2\Big(A_{i+1} \oplus A_{i+2}\Big)$ を生成する．
7. 受信データ γ_i と比較する．
   - 不一致の場合，認証不成立．（Client が不正である可能性あり．）
   - 一致した場合，認証成立．
8. γ_i を利用した共通鍵暗号方式で通信を行う．

軽量度

Server と Client の一方向ハッシュ関数の利用回数・排他的論理和の排他的論理和の演算回数は以下.

演算	Client	Server
一方向ハッシュ関数	3	6
排他的論理和	1	6

結果のように，Client では一方向性ハッシュ関数の適用が 3 回である．これは，SAS-L の **0 回**と比べて軽量とは言えない．Client もある程度の処理能力は必要であろう.