Vol.2019-CSEC-87 No.9
2019/12/4

ユーザの利用環境に応じた ポリシーを設定可能なパスワードマネージャ

松本 大輝[1,a)]　鈴木 達也[2]　木村 隼人[2]　大東 俊博[1]

概要：近年，漏洩したパスワードを悪意を持った第三者がリスト化し，それを利用してオンラインサービスへの不正ログインを行う攻撃であるパスワードリスト攻撃の被害が増加している．パスワードリスト攻撃の原因として複数の Web サイトで同じパスワードを使いまわすことが挙げられる．パスワードリスト攻撃を防ぐためには推測が困難なパスワード使用することやパスワードを Web サイトごとで使い分けることが求められているが，ユーザが記憶できるパスワードには限界がある．そのため，パスワードマネージャを用いることがパスワードリスト攻撃の対策の 1 つとして挙げられる．パスワードマネージャはユーザのパスワードと ID の組を記憶・管理できるアプリケーションであり，ユーザは自身が設定したマスターキーを用いた認証を行う事でパスワードや ID を取り出すことができる．なお，マスターキーとして扱う情報はパスワードに限らずユーザの生体情報も利用できるアプリケーションも存在している．ただ，安全な多要素認証をサポートしており，パスワードマネージャの使用環境に応じてポリシーを柔軟に変更できるパスワードマネージャは知られていない．そこで本稿では，線形秘密分散法を用いて多要素認証を柔軟に変更できるパスワードマネージャを提案し，実装・評価を行う．

キーワード：パスワードマネージャ，線形秘密分散法，多要素認証

A Password Manager with the Flexible Policy depend on Your Environment

Hiroki Matsumoto[1,a)]　Tatsuya Suzuki[2]　Hayato Kimura[2]　Toshihiro Ohigashi[1]

1. はじめに

近年，パスワードリスト攻撃によるオンラインサービスへの不正ログインの被害が増加している．パスワードリスト攻撃とは，悪意の持った第三者が Web サイトから漏洩した ID とパスワードを入手してリストを作成し，それを利用してオンラインサービスへの不正ログインを行うことである．ユーザがオンラインサービスでパスワードを使い回しているとき，このリストの情報を用いた不正ログインが可能になる．パスワードリスト攻撃を防ぐためパスワードをサービスごとに使い分けることが推奨されている．しかしながら，パスワードを使い分けた場合には膨大なパスワードを使用することになりユーザによるパスワードの管理が難しくなる．その問題を解決できる手段の 1 つとしてパスワードマネージャがある．パスワードマネージャはサービスごとに事前にユーザの ID とパスワードの組み合わせを記憶させ，SNS 等のアプリケーション認証時にユーザの代わりに認証情報を提供するアプリケーションである．多くのパスワードマネージャは，認証情報を使用する場合にマスターキーを用いる．マスターキーとしてユーザが設定したパスワードや顔や指紋といったユーザの生体情報を利用しているアプリケーションも増えてきている．

パスワードのみの認証では，容易に推測可能なパスワードを設定してしまうと秘密情報が流失してしまうことがあ

[1] 東海大学 情報通信学部，School of Information and Telecommunication Engineering，Tokai University
[2] 東海大学大学院 情報通信学研究科, Graduate School of Information and Telecommunication Engineering，Tokai University
[a)] 6bjt2210@mail.u-tokai.ac.jp

る．そのためパスワードなどの知識情報のみの認証ではなく，知識情報・生体認証・所持認証の内 2 つ以上組み合わせる多要素認証と呼ばれる認証のニーズが高まってきている．さらにサービスの利用のために用いる認証を環境の応じて変更したいことがある．例えば USB メモリにデータを保存し使用する場合は社内ではパスワードで保護し，社外で使用する場合では生体認証に対応している機器を用いるなどがある．このように秘密情報（ID やパスワードの組み合わせ）を扱う際に社内（屋内），社外（屋外），海外出張時などでユーザ要求すべきセキュリティレベルは異なることから，それぞれの利用時にシーンで異なる認証情報を要求したいことがある．同じコンセプトを PC の認証などで実現しており製品として Digital Persona[*1]などが知られている．しかしながら，利用環境に応じて多要素認証を柔軟に変更可能なパスワードマネージャは我々の知る限り知られていない．

そこで本研究では線形秘密分散法 (Linear Secret Sharing Scheme， LSSS) を使ってユーザの利用環境に応じたパスワードマネージャの開発を行う．LSSS は複数の条件のどちらも満たす必要がある AND 条件と各条件のいずれかを満たせばよい OR 条件を実現することができる．これを用いることでパスワードマネージャを使用するときに要求する認証情報を柔軟に組み合わせることができる．さらに，シェアを LAN 内にブロードキャストするような場面に対応するために LSSS によって分散されるシェアの一部を S/KEY ワンタイムパスワードの技術を用いて動的に変更する方法も提案する．また，本稿では提案方式を実装し，マスターキーの復号処理時間の評価を行う．

2. 準備

本章では，2.1 節で既存のパスワードマネージャについて述べた後に，2.2 節で提案方式で用いる線形秘密分散について説明する．またポリシーから行列への変換方式として Lewko らの方法 [1] を説明する．2.3 節ではシェアを LAN 内にブロードキャストような場面に対応するために用いる S/KEY ワンタイムパスワードについて説明する．

2.1 パスワードマネージャ

パスワードマネージャは事前にユーザの ID とパスワードの組み合わせを記憶させ，SNS 等のアプリケーション認証時にユーザの代わりに認証情報を提供するアプリケーションである．トレンドマイクロのパスワードマネージャー[*2]では登録した情報をマスターパスワードまたは指紋認証や顔認証を用いることでアクセスできる．dashlane のパスワードマネージャー[*3]ではマスターパスワードを用いることでアクセスでき，サービスのパスワードを自動で変更することもできる．中村ら [2] は逐次生成型パスワード管理システムを提案している．キーロガーに対応するために中村らはパスワードと URL を連結した結果にハッシュ処理を行い携帯端末上で QR コードとして表示して使用する．

2.2 線形秘密分散法

LSSS[1][3] は秘密分散法 [4] の一種であり，秘密情報をいくつかの分散情報 (シェア) に分散し，あらかじめ設定された条件を満たす分散情報を集めた場合に秘密情報が復元される方式である．Shamir が提案した閾値型秘密分散法 [4] は秘密情報を分散させる前に設定した閾値以上のシェアが集めることで復元できる．一方で，LSSS は秘密情報を分散させる前に秘密情報を復元可能なシェアの論理式 (アクセスポリシー) を設定し，アクセスポリシーを満たすシェアが集まった時に秘密情報を復元できる．なお，LSSS のアクセスポリシーがシェアの論理式で表現できる事から論理和 (OR) や論理積 (AND) にてシェアを結合させた形式で表現可能である．例えば，A, B, C, D, E のシェアがあったとき，$(A \wedge B) \vee (C \wedge (D \vee E))$ のような表現が指定できる．なお，本稿では $\wedge$ は AND，$\vee$ は OR として表記する．

LSSS は秘密情報をいくつかのシェアに分けるが，単純に秘密情報を分割したりコピーするわけではない．ここで，秘密情報を AND 表現に対応するように 2 つのシェアへ分散し，分散した 2 つのシェアから秘密情報を復元する方法を簡単に紹介する [1]．まず，秘密情報 (s) に乱数 (r) を足した分散情報 (シェア A) と乱数 ($-r$) の分散情報 (シェア B) に分ける．このとき，秘密情報の大きさと乱数の大きさを等しくする．シェア A は秘密情報に同サイズの乱数 r が足されているため，乱数の情報が無ければシェア A から秘密情報が漏れることはない．また，シェア B は乱数のみであるため，秘密情報を取り出すことは出来ない．そのためシェア $A(s+r)$ とシェア $B(-r)$ を取得しているとき，$(s+r)+(-r)=s$ によって秘密情報を復元できる．秘密情報を OR 表現に対応するように複数のシェアへ分散させるときは各シェアに同じ情報を割り当てる．LSSS ではポリシーを行列の形で表現し，式 (1) のように秘密情報や乱数を用いてシェアを計算する．

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix} \tag{1}$$

ポリシーから行列への変換方法は複数あるが，本研究で

[*1] https://www.digitalpersona.jp/
[*2] https://www.trendmicro.com/ja_jp/forHome/products/pwmgr.html
[*3] https://www.dashlane.com/ja/features/password-manager

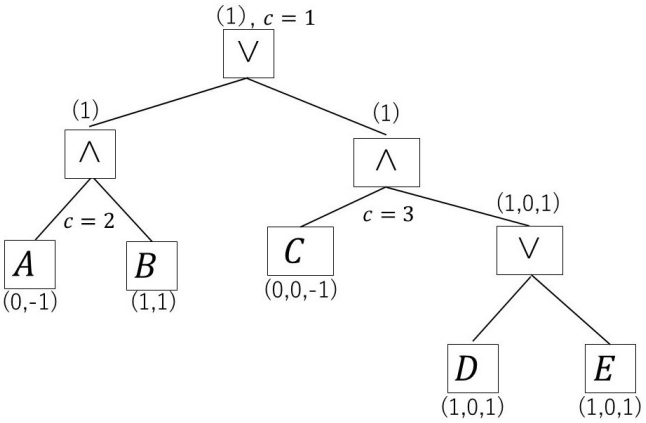

図 1　Lewko らの方法 [1] によって作成したアクセスポリシーの木構造の例 $(A \wedge B) \vee (C \wedge (D \vee E))$

は Lewko らの方法 [1] を採用してポリシーを表現した行列を木構造にて構成する．直観的には，作成したいポリシーに沿って論理式をルートノードや中間ノードに割り当て，シェアを葉ノードへ割り当てることで木構造にて LSSS を表現する．以下で $(A \wedge B) \vee (C \wedge (D \vee E))$ をポリシーとした行列 (式 (1)) への変換方法を紹介する．まず初めにルートノードにラベル (1) をセットする．その後，AND と OR によって異なるルールで中間ノードや葉ノードにラベルをセットしていく．また，行列で必要な列の数を管理するカウンタ c を $c = 1$ のようにセットする．ラベルの伝播はルートノードから前置順で行う．AND の場合，カウンタを 1 増加させ，左側の子ノードのラベルの c 番目の要素に -1，それ以外の要素には 0 をセットする．右側の子ノードのラベルの c 番目の要素に 1，それ以外の要素には親ノードと同じラベルをセットする．そのため，2 つの子ノードのラベルを加算することで親ノードの情報に戻すことができ，木構造を用いた AND 表現が実現できる．OR の場合，左右の子ノードに親ノードと同じ要素をセットする．$(A \wedge B) \vee (C \wedge (D \vee E))$ を木構造に置き換えたものを図 1 に示す．

2.3　S/KEY

S/KEY ワンタイムパスワード認証方式（S/KEY) は，Haller が提案したワンタイムパスワード認証 [5] の一種であり，パスワードに一方向関数を用いたものを使用して認証する方法である．S/KEY では初めに認証回数の上限を表すシーケンス番号 max を決める．次にパスワード $pass$ にシード (乱数文字列)$seed$ を用いて複数回ハッシュ処理をしたものをワンタイムパスワードとし，使用するごとにハッシュ処理の回数を 1 ずつ減らしていく．具体的には x 回目のワンタイムパスワードは $h^{(max-x)}(pass||seed)$ とする．ここで $h^{(max-x)}(pass||seed)$ は $h(h(h(\ldots h(pass||seed)\ldots)))$ のようにハッシュ関数を $max - x$ 回実行する処理を差す．図 2 のようにハッシュ関数の一方向性により過去のワンタイムパスワードから未来のワンタイムパスワードの推測は困難となる．

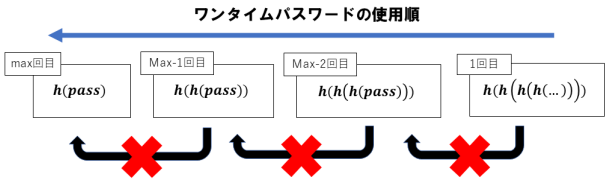

図 2　S/KEY

3.　線形秘密分散法を用いたパスワードマネージャ

3.1　概要

本章ではユーザの利用環境に応じて多要素認証のポリシーを切り替えることが可能なパスワードマネージャを提案する．本方式は LSSS のシェアのそれぞれを認証成功時に得られるようにすることで実現している．具体的には，シェアを保護する手段として記憶認証・所持認証などをサポートし，LSSS の AND 条件では対応する認証が全て成功したときに秘密情報 $pass$ が手に入るようにすることで多要素認証を実現する．さらに，LSSS の OR 条件を用いることで利用環境ごとに設定した多要素認証のポリシーのうちいずれかが満たされたときに秘密情報 $pass$ が得られるようにできる．この $pass$ をパスワードマネージャに設定することでユーザの利用環境に応じたポリシーを設定可能なパスワードマネージャが実現できる．

3.2　シェアの保護手段

提案するパスワードマネージャではシェアを保護する方法を複数用意することで柔軟なポリシーの設定を実現している．以下に 4 種類のシェアの保護手段を示す．

(1) パスワードによる保護
　パスワードを秘密鍵として共通鍵暗号によってシェアを暗号化する．シェアを取得時には，パスワードが必要になるため記憶認証に対応する保護手段と言える．

(2) 所持デバイスによる保護
　USB メモリ等のデバイス内にシェアを直接保存することでデバイスを所持していることを条件にできる．内部に保存機能が無いデバイスの場合は，共通鍵暗号の鍵にデバイス ID を用いるなどで保護することも可能ではあるが，デバイス ID を推測されることなどで安全性が低下する場合がある．この方法は所持認証を実現するための手段と言える．

(3) 場所による保護
　場所による保護とはその場所に行かないと手に入らないシェアのことである．例えば，LAN 内に接続され

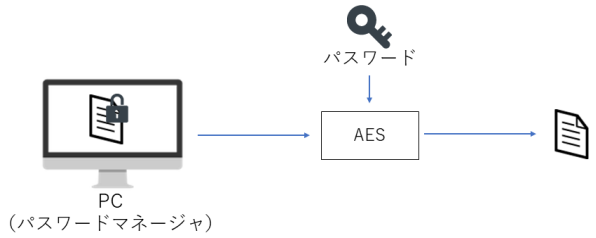

図 3　パスワードによる保護

た PC へアクセスしてシェアを得る方法や LAN 内でシェアをブロードキャストする方法などで実現できる．ユーザの行動による認証を実現するための手段と言える．

(4) 認証サーバを用いる保護
外部サーバにシェアを保存しておき，認証をした上で暗号化通信路を使ってシェアを安全に配布する方式である．認証サーバは外部に公開することで出先でのパスワードマネージャにも対応可能であり，認証サーバがサポートしている認証方式をそのまま用いることができるため多要素認証などセキュリティレベルを高めることも可能となる．

3.3　実装

提案方法のプロトタイプを C 言語を用いて実装した．LSSS の整数演算のために多倍長の計算を行う必要があるため，多倍長計算ライブラリである GMP(Ver6.1.2) を用いた．また，パスワードを用いた認証のために OpenSSL(Ver1.0.2g) を用いた．シェアの生成に用いる乱数とパスワードマネージャのマスターキーは共に 256 ビットとした．

本論文の実験ではパスワードマネージャは既存の製品を使用するのではなく，複数の ID とパスワードの組を保存したテキストファイルを AES の CTR モードで暗号化したものを疑似的にパスワードマネージャとしてみなしている．ここでマスターキーは AES の秘密鍵に対応する．シェアの保護の手段として 4 つの方法を実装した．図 3 のようにパスワードによる保護は，256 ビット鍵の AES の CTR モードを用いて実装した．暗号化されたシェアはパスワードマネージャと同じ場所に保存され，ローカル環境でパスワードを入力することでシェアを取得できる．所持デバイスによる保護は，図 4 のように USB メモリにシェアを保存することで実装した．図 5 のように場所による保護は，LAN 内に Web サーバを設置し，そこにシェアを保存する．ユーザは，LAN 内に接続できる場合に wget などによりシェアを取得できる．図 6 のように認証サーバを用いる保護は，インターネット接続可能な認証サーバにシェアを保存する．認証付きの安全なシェアの配布は scp により実現した．

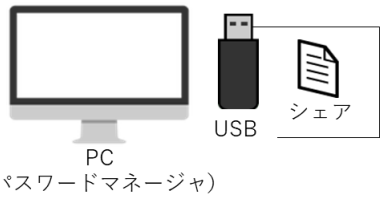

図 4　所持デバイスによる保護

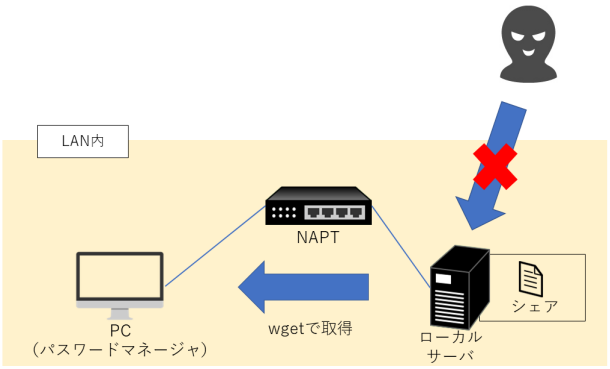

図 5　場所による保護

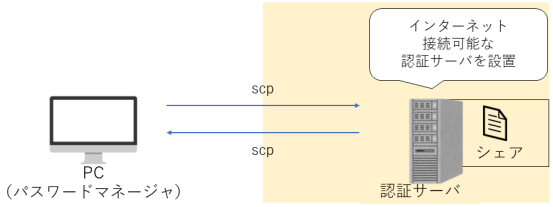

図 6　認証サーバを用いる保護

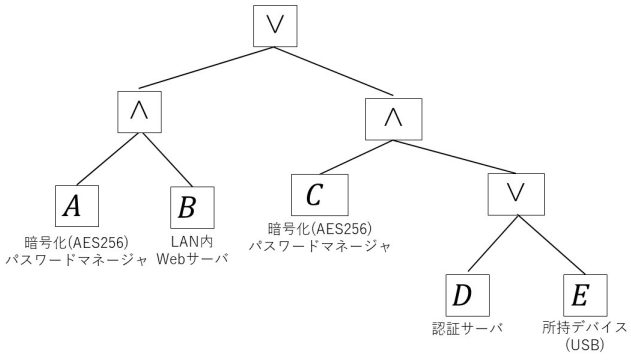

図 7　ユースケースを実現するためのポリシー

3.4　ユースケース

本稿では 2 つのユースケースを想定し，提案方法の動作を説明する．具体的には，上記の保護手段を図 7 のように $(A \wedge B) \vee (C \wedge (D \vee E))$ のポリシーで実装することで以下の利用環境が異なる 2 つの場面での利用が可能になる．

(1) ケース 1

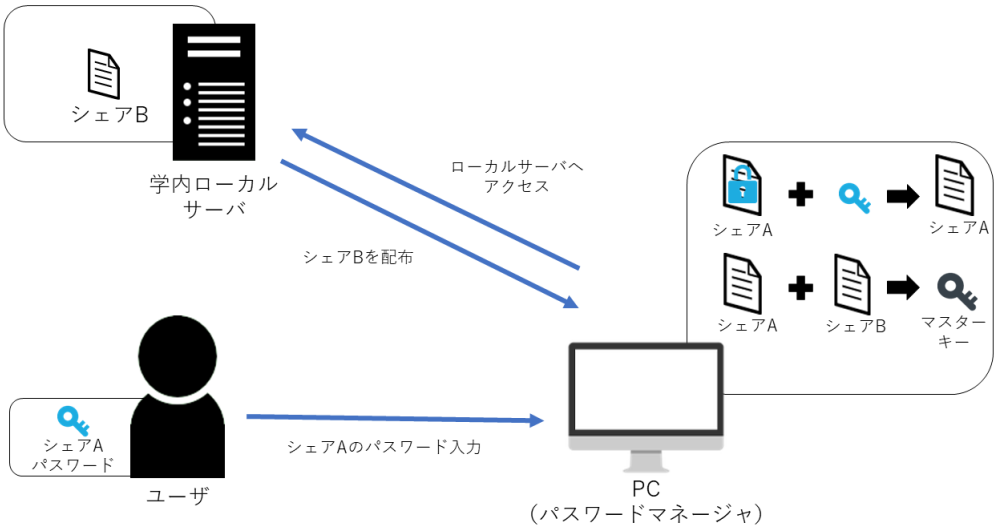

図 8　ケース 1

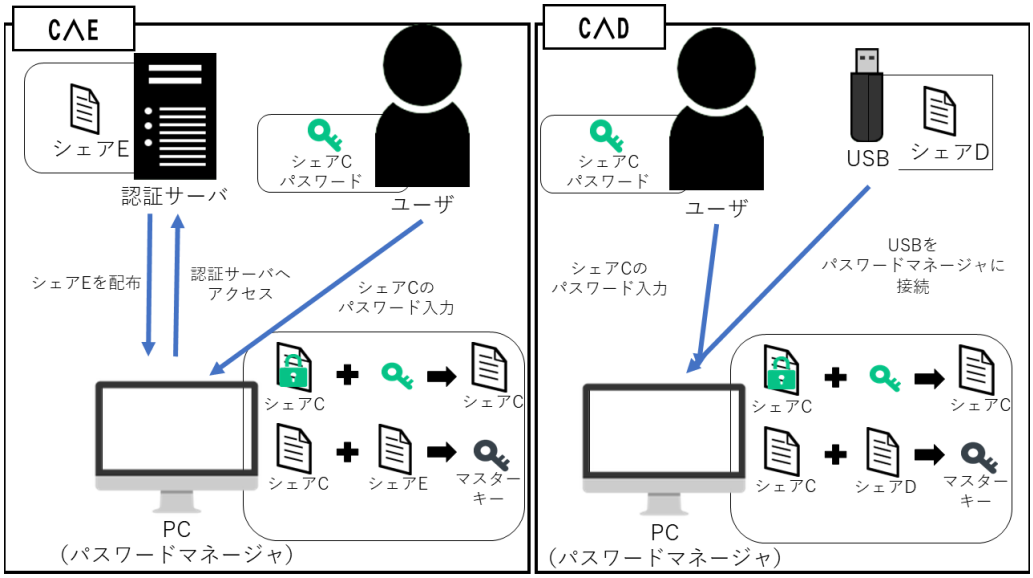

図 9　ケース 2

図 8 のように大学内で保護された LAN の使用を想定する．この場合，パスワードマネージャが要求するセキュリティレベルは比較的低いものを使うことができる．具体的には，$A \vee B$ の条件に対応しており，パスワードを知っていて LAN 内にアクセスできればパスワードマネージャを実行できるようになる．

(2) ケース 2

図 9 のように外出時の高いセキュリティレベルが必要な場合を想定している．このとき，パスワードマネージャを実行するためにパスワードを知っていること以外に事前にパスワードマネージャに登録された USB デバイス持っていることもしくはインターネットを介して認証サーバへアクセスして正しい ID とパスワードで scp を実行することのどちらかが必要である．ケース 2 は $C \vee (D \wedge E)$ に対応する．

3.5　動的シェアを用いた改良

3.4 節のユースケースではシェアは全て固定であるため，LAN 内のサーバからシェアを取得する場合では，シェアを取得した後にローカルに保存されると学外でもパスワードのみでパスワードマネージャを実行できてしまう．そこで，シェアを 1 日 1 回など期間を決めて変更することで現状より安全になる可能性が考えられる．具体的には，シェアへ埋め込む乱数の一部を変更することでシェアの一部を変更すればよい．例えば，図 1 のポリシーで r_1 を変化させると式 (1) より A と B のみシェアの値を変えることができる．通常，乱数を変更する方法としては乱数を $r_1^{(1)}, r_1^{(2)}, r_1^{(3)}, \ldots, r_1^{(max)}$ まで作り，それに対応したシェアを $A^{(1)}, A^{(2)}, A^{(3)}, \ldots, A^{(max)}$ と $B^{(1)}, B^{(2)}, B^{(3)}, \ldots, B^{(max)}$ を作成する．ここで max

はシェアの最大更新回数である．例えば，ユースケース 1 でのシェア B を $B^{(1)}, B^{(2)}, B^{(3)}, \ldots, B^{(max)}$ として日付けで変化することで対応する．しかしながら，この場合，$A^{(1)}, A^{(2)}, A^{(3)}, \ldots, A^{(max)}$ をそれぞれ暗号化してパスワードマネージャに保存することになるため，ユーザが暗号化されたシェア群を保存している場合，過去に取得したシェア A とパスワードによってマスターパスワードを復号できてしまうため根本的な解決にならない．したがって，この対応を有効にするためには，$A^{(1)}, A^{(2)}, A^{(3)}, \ldots, A^{(max)}$ も認証サーバ等に設置してサーバ上でパスワードを確認する仕組みを導入した上で，日時によって過去のシェアを削除する仕組みが必要となる．

動的シェアを用いた上記の改良では，サーバは各ユーザについて max 個のシェアを保存する必要がある．部署内の LAN 内のサーバは利用者が比較的少ないことを想定すると，$B^{(1)}, B^{(2)}, B^{(3)}, \ldots, B^{(max)}$ の保存する個数は少ないと考えられるが，認証サーバで配布する $A^{(1)}, A^{(2)}, A^{(3)}, \ldots, A^{(max)}$ は複数の部署が共同で運用することを想定するため，保存するシェアのサイズが大きくなってしまうことが予想される．そこでサーバへの負荷を減らすために暗号学的一方向関数を用いた方法を導入する．しかし，単純に一方向関数を用いるだけでは過去のシェアから未来のシェアを推測されてしまう可能性がある．そこでシェアの推測をされないために S/KEY タイプのワンタイムパスワードの仕組みを用いた方法を提案する．S/KEY を用いる方法では認証が最大 max 回のとき $r_1^{(max-x)} = h^{(x)}(r_1)$ のように計算する．$h^{(x)}(\ldots)$ は $h(h(h(\ldots)))$ のようにハッシュ関数を x 回実行する処理を差す．h は一方向関数であるため $r_1^{(x-1)}$ から $r_1^{(x)} = h^{-1}(r_1^{(x-1)})$ の推測は困難である．この一方向性関数を用いた方法によって r_1 に対応したシェア $A^{(1)}, A^{(2)}, A^{(3)}, \ldots, A^{(max)}$ は r_1 を一つ保存すれば生成できることから，サーバに保存するデータサイズを削減可能となる．ユーザ数を 10 万人でシェアを 1 年間に 1 回更新しユーザが 1 日に 24 回認証をする場合を想定する．シェア 1 個分の大きさは 32 byte であるため，S/KEY の仕組みを使用しないときシェア A を保存する認証サーバは $100000 \times 8760 \times 32(\text{byte}) = 26.1(\text{GB})$ のシェアを保存する必要があるが，S/KEY の仕組みを使用する場合は $8760 \times 32(\text{byte}) = 273.8(\text{KB})$ の乱数を保存すれば良くなる．

また，可変シェアは 1 度のみしか利用できないようになっているため，max 回がシェアの更新回数の上限になる．これを回避するためには，単純な方法はシェアを全て再生成し直して再度配布することであるが，USB メモリ等に保存しているシェアを更新するのは比較的コストが大きいと考えられる．そこで，提案する動的シェアは乱数 r_1 とマスターキーに依存していることから，(1) パスワードマネージャを起動するのに必要なシェアを集める，(2) 復元されたマスターキーと新しい乱数 r_1' を使って動的シェアのみ再生成する，(3) 再生成したシェアを乱数を認証サーバやローカルサーバにアップロードするという手順で更新するシェアの数を減らすことが可能になると考えられる．

表 1 パスワードマネージャ用機器の仕様

CPU	intel(R) core(TM) i5-7200U @ 2.50GHz
RAM	8GB
OS	Ubuntu 16.04.6 LTS

表 2 LAN 内に設置した Web サーバの仕様

CPU	intel(R) core(TM) i5-4590 @ 3.30GHz
RAM	4GB
OS	CentOS Linux release 7.6.1810(core)
Apache version	Apache/2.4.6

表 3 インターネット接続可能な認証サーバの仕様

CPU	CPU(Intel(R) Core(TM) i7-6950X CPU @ 3.00GHz
RAM	64GB
OS	Debian GNU/Linux 8.10

表 4 各シェアの取得処理時間の比較 [sec]

	取得処理	シェア復号処理	合計
AES	0.8×10^{-5}	4.8×10^{-6}	1.3×10^{-5}
USB メモリ	0.8×10^{-6}	—	0.8×10^{-6}
wget	1.7×10^{-2}	—	1.7×10^{-2}
scp	1.4×10^{-1}	—	1.4×10^{-1}

表 5 シェア復号処理時間 [sec]

取得処理/復号処理	演算処理	合計
1.4×10^{-1}	2.0×10^{-4}	1.4×10^{-1}

4. 評価実験

本章では，パスワードマネージャがシェアからマスターキーを復号するのにかかる処理時間を評価する．実験ではまずパスワードマネージャが 4 つの方法でシェアを取得する際にかかる処理時間を計測した．(1). 256 ビット鍵の AES を用いて暗号化されたシェアをユーザーが入力したパスワードを用いて復号する方法，(2). USB メモリからシェアを取得する方法，(3). LAN 内に設置した Web サーバから wget を用いて取得する方法，(4). インターネットに接続可能な認証サーバから scp を用いて取得する方法についてそれぞれ 100 回実行した平均時間を計測した．実験環境としてパスワードマネージャの仕様を表 1 に，LAN 内に設置した Web サーバの仕様を表 2 に，インターネットに接続可能な認証サーバの仕様を表 3 に示す．表 4 は各シェアの取得・復号処理時間を示したものである．

次に取得したシェアを演算しマスターキーを復号するのにかかる処理時間を計測した．この処理時間はシェアの取得・復号処理時間とシェアの整数演算処理時間を合計したものとする．なお，シェアを取得する処理時間は並列で処

理するため最も処理時間がかかったものを処理時間として表 5 のようにまとめた．この表より，マスターキーの復号に関する処理は全体で 0.135 秒程度で行えることが分かった．

5. まとめ

本稿では線形秘密分散法に基づくパスワードマネージャを実装・評価を行った．その結果，実験環境において各保存方式からシェアを取得しマスターキーを復号する処理に要する時間は 0.135 秒程度であることがわかった．この方式は，柔軟なポリシーを設定可能であり，複数の異なる利用環境での使用が要求される場面で有効である．また，シェアの一部を定期的に変更する方法についても提案・実装を行った．今後の課題としてシェアの配布にかかる処理時間についての評価を行う予定である．

謝辞

本研究の一部は JSPS 科研費 19K11971 の助成を受けたものである．

参考文献

[1] Lewko, A. B. and Waters, B.: Decentralizing Attribute-Based Encryption, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pp. 568–588 (online), DOI: 10.1007/978-3-642-20465-4_31 (2011).

[2] 照屋寛直，中村　航，古本啓祐，天願　健，田邊勝義，森井昌克：パスワードリスト攻撃に対抗するパスワード管理とシステム構築，電子情報通信学会技術研究報告，Vol. 113, No. 502, pp. 49–52 (2014).

[3] Cramer, R., Damgård, I. and Maurer, U. M.: General Secure Multi-party Computation from any Linear Secret-Sharing Scheme, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, pp. 316–334 (online), DOI: 10.1007/3-540-45539-6_22 (2000).

[4] Shamir, A.: How to Share a Secret, *Commun. ACM*, Vol. 22, No. 11, pp. 612–613 (online), DOI: 10.1145/359168.359176 (1979).

[5] Haller, N.: The S/KEY One-Time Password System, *RFC*, Vol. 1760, pp. 1–12 (online), DOI: 10.17487/RFC1760 (1995).