Vol.2019-CSEC-87 No.2
2019/12/3

アルゴリズム変換型プロキシ再暗号化を用いたオンラインストレージシステムの実装・評価

岡部　大地[1,a)]　柳　宏之[2]　木村　隼人[2,4]　鈴木　達也[2,4]　石橋　拓哉[2]　渡邉　英伸[3]
大東　俊博[1,4]

概要：IOTS 2018 において岡部らは暗号アルゴリズムを変更可能な共通鍵暗号型プロキシ再暗号化方式を提案した．岡部らの方式はプロキシ再暗号化を使わない方式（従来方式）と比べて，再暗号化時の通信量を 1/2 倍まで削減できる．しかしながら，この方式はオンラインストレージ側に再暗号化処理を行う機能を追加する必要があるため Dropbox などの既存のオンラインストレージに適用できない．そこで本稿ではユーザとオンラインストレージ間に再暗号化処理を行う機能を追加したプロキシサーバを導入し，既存のオンラインストレージにも対応可能な暗号化オンラインストレージシステムを提案する．提案方式ではプロキシサーバを使用するため通信回数が増加し，ファイルの保存するまでの合計時間は従来方式より増加する可能性がある．そこで，本研究ではプロキシサーバの設置位置を工夫することによりユーザの処理時間の削減やファイルの保存までの時間の削減を試みる．実験の結果，ユーザに近い位置にプロキシサーバを設置することでユーザの処理時間を従来方式の約 1/4 まで削減できることがわかった．さらに，本稿では通信量をさらに削減するために Intel SGX を用いたプロキシ再暗号化方式によって再暗号化鍵のサイズを従来の暗号文と同程度のサイズから鍵長程度のサイズまで削減する方法を検討する．

キーワード：暗号アルゴリズムの更新，プロキシ再暗号化，オンラインストレージ，ストリーム暗号，共通鍵暗号

Implementation and Evaluation results on an Online Storage System for the Convert type of Cryptographic Algorithms

Daichi Okabe[1,a)]　Hiroyuki Yanagi[2]　Hayato Kimura[2,4]　Tatsuya Suzuki[2,4]　Takuya Ishibashi[2]
Hidenobu Watanabe[3]　Toshihiro Ohigashi[1,4]

1. はじめに

オンラインストレージ上に長期保存しているデータの安全性はユーザがデータを暗号化することによって高度に保たれている．一方，オンラインストレージサービスの責任共有モデルにより，ユーザがデータのセキュリティを確保することが要求され，長期データ保存に対するセキュリティ維持管理の重要性が増している．例えば，暗号の 2010 年問題のように安全性を保つために政府機関から鍵長が長い暗号アルゴリズムへの移行を促された場合，暗号アルゴリズムの脆弱性 [1], [2], [3], [4] が発見された場合やサービスの運用上のルール変更により鍵長やアルゴリズムを更新しなければならない場合などに暗号アルゴリズムの変更が求められる．さらに，複数のサービスを統合する際に鍵長や暗号アルゴリズムを統一したい場合などにも同様に暗号アルゴリズムを変更する必要がある．保存している暗号化

[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
[3] 広島大学情報メディア教育研究センター
Information Media Center，Hiroshima University
[4] 国立研究開発法人 情報通信研究機構
National Institute of Information and Communications Technology
[a)] 6bjt2208@mail.tokai-u.jp

データのアルゴリズムを変更する際，ユーザは膨大な暗号文に対して再暗号化処理を行う必要がある．一般的には，暗号アルゴリズムの更新を行う場合は暗号文を一度復号し平文を取り出し，新しいアルゴリズムで再度暗号化する必要がある（以後，この方式を従来方式と呼ぶ）．従来方式の手順を図 1 に示す．従来方式では再暗号化処理を行うため当該暗号化ファイルのダウンロードおよびアップロードの通信が必要となってしまい，通信量に関するユーザの負担は比較的大きい．

ユーザを介さずに暗号文を別の暗号文へ再暗号化できる技術としてプロキシ再暗号化が知られている [5], [6], [7], [8]．プロキシ再暗号化は再暗号化鍵を用いることで暗号文を復号することなく別のユーザの暗号文に変換できる方式であり，異なるユーザへ復号する権限を委譲できる．プロキシ再暗号化は公開鍵暗号の一種として提案された [6], [7], [9]．この方式では (暗号文を生成するために利用した公開鍵と対となる) 秘密鍵及び委譲先のユーザが持つ公開鍵を利用して再暗号化鍵の生成を行う．ユーザはこの再暗号化鍵を用いて暗号文を再暗号化することにより，委譲先のユーザが持つ秘密鍵を用いて暗号文を復号できる．一方，共通鍵暗号型プロキシ再暗号化 [5] ではストリーム暗号の暗号化処理で用いる擬似乱数列を別の秘密鍵から生成された擬似乱数列に置き換えることで暗号文を復号できる鍵を変更できる．これらのプロキシ再暗号化ではユーザが再暗号化鍵を提供すればオンラインストレージ側に再暗号化処理を委託できるため通信量や処理コストを削減可能となる．

上記のようにプロキシ再暗号化は復号できる鍵の変更を主目的としている．暗号アルゴリズムの変更を目的とする方式はほとんどなく，共通鍵暗号よりも低速な公開鍵暗号に対する方式が理論的に検討されているのみである [7]．岡部らは IOTS2018 において復号時のアルゴリズムの変更を目的としたアルゴリズム変換型共通鍵暗号型プロキシ再暗号化を提案し，アルゴリズムの更新に対応可能なオンラインストレージシステムに応用した [8]．このシステムでは暗号アルゴリズムの更新が必要になった際にユーザが更新前の暗号アルゴリズムと更新後の暗号アルゴリズムのキーストリームを排他的論理和（XOR）することにより再暗号化鍵を生成し，オンラインストレージにアップロードする．次に再暗号化鍵と更新前の暗号アルゴリズムにて生成された暗号文を用いてオンラインストレージは再暗号化処理を行うことで更新後の暗号アルゴリズムにて生成された暗号文へ置き換える．従来方式では暗号化ファイルの送受信が必要なのに対し，岡部らの方式は暗号化ファイルと同サイズの再暗号化鍵の送信のみで良いため，岡部らの方式は従来方式と比べ通信量を 1/2 に削減することができる．しかしながら，岡部らの方式は再暗号化処理を行う機能をオンラインストレージに追加しなければならないため既存のオンラインストレージを使用することができない．

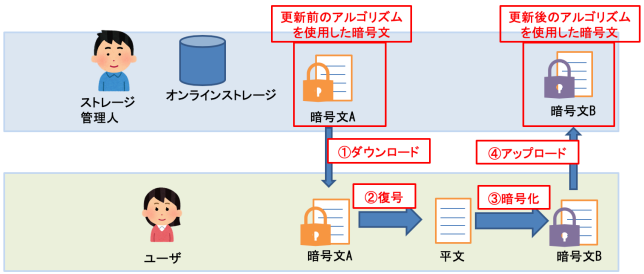

図 1　従来方式

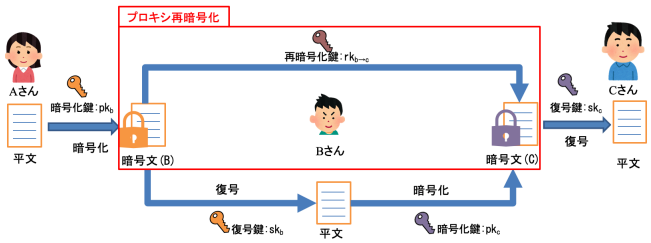

図 2　プロキシ再暗号化

そこで本研究では，ユーザとオンラインストレージ間に再暗号化処理を行うプロキシサーバを導入することで既存のオンラインストレージを利用できるようにする．提案方式ではユーザとオンラインストレージの間にプロキシサーバを設置しているため通信回数が増加し，全体の処理時間が遅くなる可能性がある．そこで，本研究ではプロキシサーバの設置位置を工夫することで効果的な再暗号化処理を行うことを試みる．具体的には，(1) ユーザの処理時間の軽減するためにエッジに近い位置にプロキシサーバを設置する，(2) プロキシサーバとオンラインストレージ間でファイルの送受信時間の削減やファイルを保存するまでの時間を減らすためにオンラインストレージと同じリージョンにプロキシサーバを設置する．特に，(2) は途中の通信路の通信量が従来方式の 1/2 になるため ISP 等への負担を軽減できることもメリットとなる．さらに，提案方式への認証暗号の適用，再暗号化鍵のサイズを鍵長サイズ程度まで削減可能な Intel SGX を用いた方式についての考察を行った．

なお，文献 [10] においては我々はプロキシサーバの試作及び基本性能の評価を行っているが，この論文はそのフルバージョンであり設置位置の検討を含めた詳細な評価や考察も加えたものである．

2. プロキシ再暗号化

2.1 概要

プロキシ再暗号化は暗号文を復号することなく，再暗号化鍵を用いて別の復号鍵で復号できる暗号文に変換する方式である．再暗号化鍵から復号鍵の情報が漏れないように設計することから再暗号化処理を第三者に委託することができる．公開鍵暗号型プロキシ再暗号化は Blaze らによっ

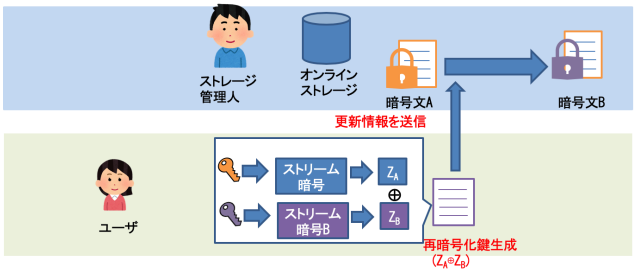

図 3　岡部らの方式 [8]

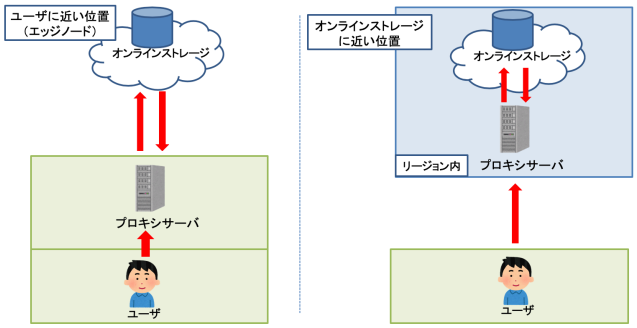

図 4　プロキシサーバの設置位置

て初めて提案された [6].

ここで公開鍵暗号型のプロキシ再暗号化の手順を説明する．ユーザ A はユーザ B 用の暗号化鍵 pk_B を用いて平文 M を暗号化し，暗号文を送信する．この暗号文はユーザ B の復号鍵 sk_B で復号できる．ここで，この暗号文をユーザ C の復号鍵 sk_C で復号できるように変換することを考える．プロキシ再暗号化では，ユーザ B は自身の復号鍵 sk_B とユーザ C の暗号化鍵 pk_C を用いて再暗号化鍵 $rk_{B\to C}$ を生成し，再暗号化鍵を使って sk_C で復号できるように暗号文を再暗号化する．プロキシ再暗号化の手順を図 2 に示す．

2.2　共通鍵暗号型プロキシ再暗号化

共通鍵暗号型プロキシ再暗号化はストリーム暗号に基づく方法であり，高速に動作する特徴を持つ．本節では，ストリーム暗号およびそれを応用した共通鍵暗号型プロキシ再暗号化方式について説明する．

2.2.1　ストリーム暗号

ストリーム暗号は共通鍵暗号方式の一つのクラスであり高スループットが要求する環境で用いられる．ストリーム暗号では秘密鍵を擬似乱数生成器のシードとして入力し，擬似乱数生成器によって生成された擬似乱数列 (キーストリーム) と平文を排他的論理和することで暗号化を行う．復号時にも同じ秘密鍵をシードとしてキーストリームを生成すれば暗号文から平文を得ることができる．具体的には，秘密鍵を K を擬似乱数生成器 $f()$ の入力情報としたとき，キーストリーム $Z = f(K)$ を得る．平文を M，暗号文を C としたとき，暗号化は $C = M \oplus Z = M \oplus f(K)$，復号は $M = C \oplus Z = M \oplus Z \oplus Z$ と書ける．

2.2.2　従来の共通鍵暗号型プロキシ再暗号化方式

共通鍵暗号型プロキシ再暗号化はストリーム暗号が平文とキーストリームの排他的論理和によって暗号化されていることに注目した方式であり [5], [7], [11]，復号可能なユーザを置き換える他，同一ユーザの復号鍵の更新を目的として使われている．

共通鍵暗号型プロキシ再暗号化では，共通鍵 K_A で暗号化した暗号文を共通鍵 K_B で復号可能にするために再暗号化鍵 $Z_A \oplus Z_B = f(K_A) \oplus f(K_B)$ を使用する．文献 [5] では第三者機関が安全なプロトコルで 2 者から再暗号化鍵を受け取る仕組みを作っているが，鍵更新を目的とする場合は自身で作成することもできる．再暗号化鍵を用いることで共通鍵 K_A で作成された暗号文 $C_A = M \oplus f(K_A) = M \oplus Z_A$ から共通鍵 K_B が復号できる暗号文 $C_B = M \oplus f(K_B)$ へ $C_B = C_A \oplus (Z_A \oplus Z_B) = M \oplus Z_A \oplus (Z_A \oplus Z_B) = M \oplus Z_B$ のように変換できる．

再暗号化のためのデータサイズを暗号文長ではなく鍵長に依存をさせた方式として，鍵準同型擬似ランダム関数（鍵準同型 PRF）を用いた方式が提案されている [7], [11] Boneh らは鍵準同型 PRF を用いた方式を提案し，共通鍵のペア (K_A, K_B) を保護した状態で送信し，再暗号化鍵 $Z_A \oplus Z_B$ の生成をサーバ側で実行できるようにして再暗号化時の通信量を削減することに成功している [11]．暗号化ファイルの改ざんを保護する完全性を与えるためにメッセージ認証符号と暗号化を組み合わせた方式として認証暗号が知られている．Everspaugh らは Boneh らの要素技術を用いることで認証暗号に対応した方式を提案している [7]．さらに Everspaugh らはプロキシ再暗号化における新たな安全性定義を行い，Boneh らの安全性定義が十分ではないことを示している．これらの方法は再暗号化時のデータの通信量を削減できるというメリットはあるが，使っている要素技術の処理速度が低速であるため実用的とは言い難い．

以上のように，共通鍵暗号型プロキシ再暗号化の提案はあったが，暗号アルゴリズムの変更が可能な方式は我々の知る限り知られていなかった．

2.3　岡部らの方式 [8]

岡部らの方式は共通鍵暗号型プロキシ再暗号化と同じくキーストリームの置き換えによって暗号アルゴリズムの変更を可能にしている．ストリーム暗号の本質は擬似乱数生成器であり，あるストリーム暗号の擬似乱数生成器を関数 $f()$ と表記したとき，異なるストリーム暗号の擬似乱数生成器は別の関数 $f'()$ を用いた方式と定義できる．このとき，共通鍵 K_A で暗号化された暗号文のアルゴリズムを $f()$ のものから $f'()$ に変換する場合，再暗号化鍵を $Z_A \oplus Z'_A$ のように生成すればよい．変更前の暗号アルゴリズムで作成

した暗号文 $C_A = M \oplus f(K_A) = M \oplus Z_A$ から更新後の暗号アルゴリズムで作成した暗号文 $C'_A = M \oplus f'(K_A)$ へ変換するには，再暗号化鍵を用いて $C'_A = C_A \oplus (Z_A \oplus Z'_A) = M \oplus Z_A \oplus (Z_A \oplus Z'_A) = M \oplus Z'_A$ のように計算する．

以上のように，共通鍵を変更するときも暗号アルゴリズムを変更するときも本質的にはキーストリームの置き換えであることに注目して暗号アルゴリズムの変換機能を導入している．岡部らの方式はストリーム暗号用であるが，CTR モードなどを用いることでブロック暗号のアルゴリズムにも対応可能である．なお，暗号アルゴリズムの変換と同時に共通鍵の更新をすることも原理的には可能である．図 3 は岡部らの方式をオンラインストレージシステムに適用した場合の処理を示している．この場合，岡部らの方式はユーザからはオンラインストレージに再暗号化鍵を送信するのみで良いため，送受信が必要な従来方式と比べて通信量を 1/2 に削減できる．

3. 提案方式

岡部らの方式は再暗号化処理をオンラインストレージ側で実行する必要があるため，既存のオンラインストレージサービスへの適用は難しい．そこで，本稿ではユーザとオンラインストレージ間に再暗号化処理を行うプロキシサーバを設置することで既存のオンラインストレージサービスへも適用可能な暗号化オンラインストレージシステムを提案する．

提案方式で暗号アルゴリズムの更新を行う場合の具体的な手順を示す．ユーザは自身の共通鍵を用いて更新前の暗号アルゴリズムと更新後の暗号アルゴリズムから生成したキーストリームの組を排他的論理和することで再暗号化鍵を生成し，プロキシサーバに送信する．プロキシサーバはオンラインストレージから再暗号化対象の暗号文をダウンロードし，再暗号化鍵と暗号文を排他的論理和することで更新後の暗号アルゴリズムで暗号化された暗号文へ変換する．最後にプロキシサーバはオンラインストレージに変換後の暗号文をアップロードすることで更新後の暗号文に置き換わる．図 5 は提案方式の処理を図示したものである．

提案方式ではユーザからプロキシサーバに再暗号化鍵を送信するまでで処理が終了するためユーザの負担は軽減される．しかしながら，ユーザ・プロキシサーバ間，プロキシサーバ・オンラインストレージ間でそれぞれ通信が発生するため提案方式で再暗号化後のファイルの保存が終了するまでの時間は従来方式より長くなる可能性がある．そこで，この影響を抑えるためにプロキシサーバの効果的な設置位置について検討する．具体的には，(1) ユーザが必要とする処理時間を軽減するためにプロキシサーバをユーザに近い位置に設置する，(2) プロキシサーバとオンラインストレージ間の送受信時間を削減するためにオンラインストレージと同一リージョン内にプロキシサーバを設置する．(1) はエッジコンピューティングと呼ばれている考え方を採用しており，ユーザが処理を終えるまでの時間を短縮する提案方式のメリットを増加させることを狙っている．(2) はクラウドにオンラインストレージを設置していることを想定しており，クラウドサービスの同一リージョン内にプロキシサーバを設置することでプロキシサーバ・オンラインストレージ間の通信が高速になることを期待したものである．これにより，プロキシサーバ・オンラインストレージ間の通信速度がユーザ・プロキシサーバ間の通信速度より十分に大きいとすれば，提案方式は岡部らの方式と同程度の通信時間まで削減できることが期待できる．さらに，(2) はクラウドサービス外の通信が再暗号化鍵の送信のみになるため，インターネット通信量は従来方式の 1/2 に抑えることができ，ISP 等への負担を軽減できる．

3.1 実装

本研究では，オンラインストレージとして ownCloud を使用している．AES-CTR と RC4 は C 言語で実装し，AES-CTR については openssl ライブラリを使用した．Python のプログラムを Apache で動作させるために mod_wsgi モジュールと，RESTful API を実装するために Flask を使用した．ユーザとプロキシサーバ間の送受信については Python の Requests ライブラリで行い，プロキシサーバと ownCloud 間の送受信については ownCloud の API を使用した．これらの通信は HTTPS で行う．暗号化・復号・再暗号化鍵生成・再暗号化処理に関しては C 言語の実行ファイルを用意し，subprocess で呼び出し実行する．プログラムの一連の流れとしてはプロキシサーバのアドレスにファイルを POST することによって，ファイルのダウンロード，再暗号化処理，アップロードの処理が逐次行われる．

4. 実験

提案方式は従来方式と異なりユーザの通信回数が 1/2 に減っていることから通信に要する時間が短縮されている．しかし，ファイルを保存するまでの合計時間は従来方式より長くなってしまう．これに対して，プロキシサーバの設置位置を工夫することでユーザの処理時間の削減やファイルの保存までの合計時間の削減できるか試みる．本章では，暗号アルゴリズムの更新する際に従来方式と比べて提案方式がユーザが要する時間をどの程度短縮できるかを評価する．また，プロキシサーバをユーザに近いエッジ側に設置する場合とオンラインストレージの同一リージョンに設置した場合の提案方式の処理時間を計測し，ユーザの処理時間の削減やファイルを保存するまでの処理時間の短縮が可能であるかを検討する．

4.1 実験環境

本実験で用いている機器はユーザ用のクライアント，エッ

ジ側のプロキシサーバ，オンラインストレージ用のクラウドサーバ，オンラインストレージと同じリージョン内のプロキシサーバの 4 台を使用している．ユーザ用の機器は暗号化ファイルおよび再暗号化鍵の送受信を行う．エッジ側のプロキシサーバはユーザと同じ LAN 内に設置しており再暗号化処理やオンラインストレージへ送受信を行う．オンラインストレージ用のクラウドサーバはさくらのクラウドの石狩リージョンのサーバを使用し，ownCloud によりオンラインストレージを構築する．また，プロキシサーバをクラウド上に設置する場合，オンラインストレージと同じくさくらのクラウドの石狩リージョンのサーバを使用する．それぞれのソフトウェアの構成とハードウェアの構成を表 1，2，3 に示す．

4.2 結果

本実験では，1KB，10KB，100KB，1MB，10MB のファイルを用意し，従来方式，プロキシサーバをエッジ側に設置した場合とオンラインストレージと同一リージョン内に設置した場合の提案方式の処理時間を計測した．計測では再暗号化を 100 回実行した平均時間を求め，その結果を表 4，5，6 に示す．

4.2.1 提案方式と従来方式との比較

暗号アルゴリズムの更新する際に従来方式と比べて提案方式がユーザが要する時間をどの程度短縮できるか検討するため従来方式とプロキシサーバをエッジ側に設置した提案方式の比較を行った．従来方式ではユーザが要する時間はダウンロード，復号，暗号化，アップロードの処理時間の合計となる．一方，提案方式ではユーザが要する時間は再暗号化鍵の生成，アップロードの処理時間の合計となる．表 4 より従来方式のユーザの処理時間を 10MB で見た場合，ダウンロード+復号+暗号化+アップロードの処理時間の合計となり約 4.9 秒であった．表 5 より提案方式はユーザの処理時間は再暗号化鍵の生成+アップロードの処理時間の合計となり約 1.1 秒となった．従来方式と提案方式を比較するとユーザの処理時間は約 1/4 程度まで短縮できることがわかった．

4.2.2 設置位置の違いによる比較

ユーザの処理時間の削減やファイルを保存するまでの処理時間の短縮が可能であるかを検討するためプロキシサーバをエッジ側に設置した場合とオンラインストレージと同一リージョンに設置した場合の提案方式の比較を行った．

まず初めに，ユーザの処理時間を比較する．提案方式のユーザの処理時間は再暗号化鍵生成とアップロードの処理時間の合計となる．10MB の処理時間で見た場合，表 5 よりプロキシサーバをエッジ側に設置した場合の処理時間は約 1.1 秒要し，表 6 よりオンラインストレージと同一リージョンに設置した場合の処理時間は約 1.3 秒となった．ユーザの処理時間はプロキシサーバをオンラインストレージと同一リージョンに設置した場合よりもエッジ側に設置した方が約 0.2 秒処理が少ないことがわかった．

次に，合計処理時間の比較を行う．10MB の処理時間で見た場合，表 5 よりプロキシサーバをエッジ側に設置した場合の処理時間は約 4.3 秒要し，表 6 よりオンラインストレージと同一リージョン内に設置した場合の処理時間は約 4.2 秒要することがわかった．オンラインストレージと同一リージョン内にプロキシサーバを設置した場合の方がエッジ側に設置した場合より約 0.1 秒処理が速いことがわかった．

表 1　ユーザ用マシンの構成

項目	名称
OS	Debian 8.10
CPU	Intel(R) Core(TM) i7-6950X @ 3.00GHz
メモリ	64GB
プログラム言語	Python 3.4.2
ライブラリ	Requests 2.22.0

表 2　エッジノードの構成

項目	名称
OS	Ubuntu 18.04.2 LTS
CPU	Intel(R) Core(TM) i5-4690 @ 3.50GHz
メモリ	16GB
プログラミング言語	Python 3.6.8
ライブラリ	Flask 0.12.2
モジュール	mod_wsgi 4.5.11, subprocess
Web サーバ	Apache 2.4.29 (Ubuntu)

表 3　クラウドサーバの構成

項目	名称
OS	Centos 7.6.1810
CPU	Intel(R) Core(TM) i5-4690 @ 3.50GHz
メモリ	64GB
プログラミング言語	Python 3.6.8
ライブラリ	Flask 1.1.1
モジュール	mod_wsgi 4.5.11, subprocess
オンラインストレージ	ownCloud 9
Web サーバ	Apache 2.4.41

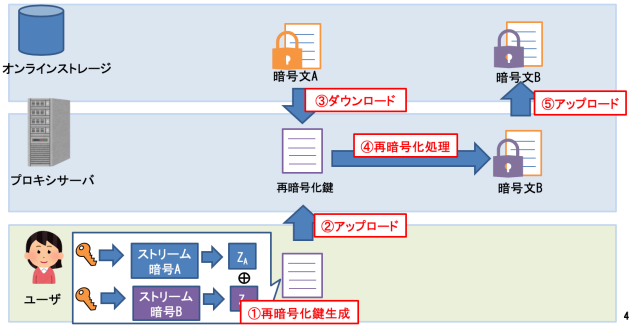

図 5　提案システム

表 4　従来方式における各区間での処理時間 [sec]

	ダウンロード	復号	暗号化	アップロード	合計
1KB	0.4241	0.0020	0.0020	0.5062	0.9343
10KB	0.4366	0.0024	0.0020	0.5473	0.9883
100KB	0.4628	0.0060	0.0026	0.5936	1.0651
1MB	0.6001	0.0384	0.0053	0.8339	1.4778
10MB	1.8784	0.2316	0.0211	2.7740	4.9051

表 5　提案方式における各区間での処理時間（プロキシサーバをユーザと同一 LAN 内に設置）[sec]

	再暗号化鍵生成	アップロード	ダウンロード	再暗号化処理	アップロード	合計
1KB	0.0016	0.0251	0.4904	0.0034	0.5809	1.1014
10KB	0.0018	0.0256	0.4796	0.0036	0.6121	1.1227
100KB	0.0041	0.0343	0.5446	0.0047	0.6811	1.2688
1MB	0.0229	0.1188	0.9745	0.0125	0.7563	1.8850
10MB	0.1494	0.9443	1.4011	0.0505	1.7638	4.3091

5. 考察

5.1 プロキシサーバの設置位置

実験結果からプロキシサーバをユーザに近いエッジ側に設置することでユーザが必要とする処理時間を従来方式の 1/4 以下まで軽減できることがわかった．この結果はオンラインストレージを設置しているサーバとのネットワークの距離が遠いときにより有効になると考えられる．

プロキシサーバをオンラインストレージと同一リージョンに設置した場合については，プロキシサーバ・オンラインストレージ間の通信が高速になって暗号文の再暗号化が終了するまでの全体の処理時間が劇的に短縮すると予想していたが，予想に反して改善度合いが小さかった．この原因としては，実験で使用したクラウドサーバが国内であることに加えてユーザの機器も大学内の比較的高速な通信環境であったことから，同一リージョン内というメリットが得られにくかったのではないかと予想している．さらに，オンラインストレージを実現するためのソフトウェア ownCloud のレスポンスタイム等も通信時間に含まれているため，通信時間の影響に関してはさらに精査する必要がある．しかしながら，評価実験では観測できないが，メリットの一つであるクラウドサービスのリージョン外での通信量自体の削減については効果があると思われるため，プロキシサーバをオンラインストレージと同一リージョンに設置する使用方法に関しても現時点でも一定の意味があると考えられる．

5.2 メッセージ認証

認証暗号を実現するために，本稿では MAC-then-Encrypt と Encrypt-then-MAC の 2 種類の構成法について考察する．MAC-then-Encrypt の場合，平文から計算した MAC 値を平文と連結してから暗号化を行うため，プロキシ再暗号化に影響することなく適用することができる．しかし，Encrypt-then-MAC の場合は MAC 値を暗号文から作成するため，プロキシ再暗号化によって暗号文が変化してしまうと MAC 値が変化するためユーザによる MAC 値の再計算が必要となる．したがって，単純に Encrypt-then-MAC に適用すると MAC 値を計算するためにユーザは暗号文を受信しなければならず，プロキシ再暗号化によるメリットが失われてしまう．この問題は，MAC 値を暗号文本体ではなく暗号文のハッシュ値から計算する方法で解決できると考えている．具体的には，サーバは暗号文のハッシュ値をユーザに送信し，ユーザはハッシュ値から MAC 値を計算してサーバに返すことで暗号文のサイズが大きい場合でも対応が可能となる．この解決法では semi honest model（サーバは暗号文から平文を得ようとするが，プロトコルを正しく実行する）を仮定している．

5.3 Intel SGX を用いた方式

Intel Software Guard Extensions（Intel SGX）は Intel が提供しているハードウェアサポートされたセキュアな実行環境である．アプリケーション上に Enclave と呼ばれる暗号化されたメモリ領域を生成し，Enclave 内のデータを保護しながらプログラムを実行することができる．複数の Enclave 間で安全にデータの共有を行うために Intel SGX の機能の一つとして Attestation がある．同一機器内での異なる Enclave 間の通信を想定する場合には Local Attestation を利用する．遠隔にある Enclave を信用するために Remote Attestation を用いる．

Fisch ら [12] は Intel SGX をモデル化したセキュアハードウェアスキームと公開鍵暗号，デジタル署名を組み合わせることでシングルインプット関数型暗号（Iron）を提案した．関数型暗号は関数情報を含んだ復号鍵を用いて暗号文を復号した場合，平文ではなく平文を入力した関数の演

表 6　提案方式における各区間での処理時間（オンラインストレージのリージョン内） [sec]

	再暗号化鍵生成	アップロード	ダウンロード	再暗号化処理	アップロード	合計
1KB	0.0016	0.1265	0.4613	0.0093	0.5514	1.1501
10KB	0.0018	0.1583	0.4740	0.0096	0.5630	1.2067
100KB	0.0044	0.2006	0.4747	0.0104	0.5617	1.2518
1MB	0.0199	0.3455	0.4677	0.0214	0.5810	1.4356
10MB	0.1419	1.1364	1.3136	0.1056	1.5038	4.2014

算結果を出力する方式である．さらに，シングルインプット関数型暗号は一入力の関数型暗号である．

Iron では，はじめに Intel の認証機関が公開鍵暗号の暗号化鍵と復号鍵，デジタル署名の署名鍵と検証鍵を Key Manager Enclave（KME）内にて作成する．このとき，暗号化鍵と検証鍵を公開パラメータとして Intel の認証機関からユーザへ渡す．オンラインストレージ内の Decryption Enclave(DE) は暗号文を復号するために KME との Remote Attestation を行い，KME から復号鍵を受け取る．さらに，ユーザは任意の関数に対応した秘密鍵の生成を KME に要求する．KME は任意の関数のデータと署名鍵によって秘密鍵を生成する．ユーザはオンラインストレージへ KME からもらった秘密鍵を渡す．ただ，オンラインストレージにある DE では暗号文の復号しかできないため，Function Enclave(FE) を新たに生成し，FE に秘密鍵を保存する．その後，ユーザは暗号化鍵を用いてデータを暗号化し，生成した暗号文をオンラインストレージ内の FE へ送る．FE は秘密鍵を Local Attestation で DE に送り，検証鍵で検証を行う．最終的に，DE から FE へ Local Attestation で復号鍵を送ることによって，FE にて暗号文の復号を介して関数値を出力できる．

岡部らの方式は Iron における関数記述を $F() = f() \oplus f'()$ と定義する事で実現可能と考える．Intel SGX を用いた方式についての概略を図 6 に示す．具体的には， Iron は Intel SGX の Enclave 内の関数 $F()$ をプログラムすることが可能であるため，オンラインストレージの Enclave 内の関数 $F() = f() \oplus f'()$ と定義する．ユーザは Intel の認証機関から公開されている暗号化鍵で共通鍵 K を暗号化 $C = Enc(K)$ する．ユーザは Remote Attestation を用いてオンラインストレージに関数計算 Enclave にて暗号文 C を送る．オンラインストレージは事前に Intel の認証機関から Remote Attestation によって送られている復号鍵を用いて暗号文 C を復号し，共通鍵 K を得る．この共通鍵を定義した関数 $F() = f() \oplus f'()$ に入力することで再暗号化鍵 $F(K) = f(K) \oplus f'(K) = Z_A \oplus Z'_A$ を生成する．オンラインストレージは再暗号化鍵を用いて安全な暗号文 B に変換することができる．

岡部らの方式は再暗号化鍵が暗号文のサイズに依存していたため，巨大なサイズの暗号文を再暗号化する際には鍵送信にかかる負担が膨大になってしまう．一方で，Intel

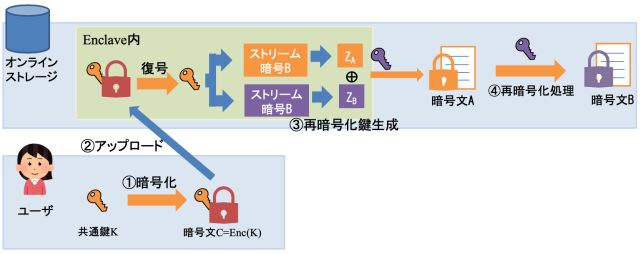

図 6　Intel SGX を利用した方式

SGX を用いた方式ではオンラインストレージ内で再暗号化鍵を生成するため，ユーザが送信する暗号文サイズは共通鍵の鍵サイズ程度に軽減できる． ただし，Intel SGX を用いるプロキシ再暗号化は，オンラインストレージに対しているサーバに Intel SGX があること，オンラインストレージのサーバに機能を追加する必要があるという制限がある．

6. まとめ

本研究では暗号アルゴリズムの更新に対応可能な共通鍵型のプロキシ再暗号化を使用したオンラインストレージシステムの提案および実装・評価を行った．ユーザとオンラインストレージ間にプロキシサーバを導入することにより，Dropbox などの既存のオンラインストレージサービスを利用し，機能を追加することなく暗号アルゴリズムの更新を可能とするシステムを実現した．提案システムの実装・評価を行い，ユーザにかかる処理時間を従来方式から約 1/4 まで削減することに成功し提案方式の有効性を示した．また，提案方式におけるプロキシサーバの設置位置を工夫することで，より効率的に再暗号化を行う方法について評価した．さらに，提案方式への認証暗号の適用方法および再暗号化鍵のサイズを暗号文サイズから，共通鍵の鍵サイズまで削減可能な Intel SGX を利用した方式についての考察を行った．今後の課題としては Intel SGX を用いたプロキシ再暗号化の実装・評価を行うことがあげられる．

謝辞

本研究の一部は JSPS 科研費 19K11971 の助成，MIC/SCOPE (課題番号 162108102) の委託を受けたものである．

参考文献

[1] Fluhrer, S. R., Mantin, I. and Shamir, A.: Weaknesses in the Key Scheduling Algorithm of RC4, *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers*, pp. 1–24 (online), DOI: 10.1007/3-540-45537-X_1 (2001).

[2] Tews, E., Weinmann, R. and Pyshkin, A.: Breaking 104 Bit WEP in Less Than 60 Seconds, *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, pp. 188–202 (online), DOI: 10.1007/978-3-540-77535-5_14 (2007).

[3] Teramura, R., Asakura, Y., Ohigashi, T., Kuwakado, H. and Morii, M.: Fast WEP-Key Recovery Attack Using Only Encrypted IP Packets, *IEICE Transactions*, Vol. 93-A, No. 1, pp. 164–171 (online), DOI: 10.1587/transfun.E93.A.164 (2010).

[4] Sasaki, Y., Wang, L., Ohta, K. and Kunihiro, N.: Security of MD5 Challenge and Response: Extension of APOP Password Recovery Attack, *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, pp. 1–18 (online), DOI: 10.1007/978-3-540-79263-5_1 (2008).

[5] Watanabe, D., Sakazaki, H. and Miyazaki, K.: Representative System and Security Message Transmission using Re-encryption Scheme Based on Symmetric-key Cryptography, *JIP*, Vol. 25, pp. 67–74 (online), DOI: 10.2197/ipsjjip.25.67 (2017).

[6] Blaze, M., Bleumer, G. and Strauss, M.: Divertible Protocols and Atomic Proxy Cryptography, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pp. 127–144 (online), DOI: 10.1007/BFb0054122 (1998).

[7] Everspaugh, A., Paterson, K. G., Ristenpart, T. and Scott, S.: Key Rotation for Authenticated Encryption, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, pp. 98–129 (online), DOI: 10.1007/978-3-319-63697-9_4 (2017).

[8] 岡部大地，石橋拓哉，木村隼人，渡邉英伸，大東俊博：暗号の危殆化に対応可能なオンラインストレージシステムに関する検討，インターネットと運用技術シンポジウム論文集，Vol. 2018, p. 41 (2018).

[9] Ateniese, G., Fu, K., Green, M. and Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage, *ACM Trans. Inf. Syst. Secur.*, Vol. 9, No. 1, pp. 1–30 (online), DOI: 10.1145/1127345.1127346 (2006).

[10] 柳宏之，岡部大地，石橋拓哉，木村隼人，渡邉英伸，大東俊博：暗号の危殆化に対応可能なオンラインストレージシステムの実装・評価，2019 年電子情報通信学会ソサエティ大会，Vol. 2018, p. 42 (2018).

[11] Boneh, D., Lewi, K., Montgomery, H. W. and Raghunathan, A.: Key Homomorphic PRFs and Their Applications, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pp. 410–428 (online), DOI: 10.1007/978-3-642-40041-4_23 (2013).

[12] Fisch, B., Vinayagamurthy, D., Boneh, D. and Gorbunov, S.: IRON: Functional Encryption using Intel SGX, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pp. 765–782 (online), DOI: 10.1145/3133956.3134106 (2017).