情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-SE-164 No.18
Vol.2009-EMB-13 No.18
Vol.2009-CSEC-45 No.18
2009/5/29

木構造を用いた鍵管理プロトコルについて

吉田勝彦[†1]　宮地充子[†1]

鍵管理は暗号の研究における主要な課題である．ユーザの集合が共通の鍵を共有したい場合，多数のスキームが提案されている．それらは全ユーザを階層的に構成するものが多い．階層構造に適し，信頼できる第三者機関を必要としない方式が Canard らによって提案された．この方式は階層構造をグラフで表して鍵管理を行う方式であり，HMAC と階層構造ではないグループ鍵共有プロトコルを用いている．この方式には R&R GKA と呼ばれる鍵共有法が必要であり，任意のユーザが鍵を効率的に更新することが求められる．しかし，提案されている鍵共有方式では，ユーザの追加及び削除のアルゴリズムは提案されているが，複数ユーザが一度に鍵更新を行う方式については述べられていない．本研究では，既存方式の問題点を指摘し，複数ユーザが一度に鍵更新を行う方式について考察する．

On a Tree-Based Group Key Management

Katsuhiko Yoshida[†1] and Atsuko Miyaji[†1]

Key Management is one of the main problems in cryptographic reseach. Many schemes to share a common key have been proposed. Most of schemes group users with hierarchical structures. Canard et al. proposed a scheme suitable for a hierarchical structure without central authority in 2008. This scheme manages a key with a hierarchical structure representing a graph. This uses HMAC and non-hierarchical Group Key Agreement. This scheme needs that any users can refresh their own secret key efficiently with a Group Key Agreement called R&R GKA. However there are no GKA schemes to refresh keys at once. can refresh their key in small rounds.

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

1. はじめに

鍵管理は暗号の研究における主要な課題である．ユーザの集合が共通の鍵を共有したい場合について，多数のスキームが提案されている．それらは全ユーザを階層的に構成するものが多い．階層構造に適した信頼できる第三者機関の必要でない方式が Canard らによって提案された．この方式は階層構造を有向グラフで表して鍵管理を行う方式であり，HMAC と階層構造ではないグループ鍵共有プロトコルを用いている．

階層構造のある企業などの組織において，上司は部下の鍵を全て計算できるが，部下は自分の所管の部下の鍵のみしか生成できないようにしてアクセス管理を行う場合がある[CJ08]．また，全ての鍵を鍵管理センタのような 1 つのエンティティが生成するのではなく，各結節に鍵生成の権限を委任することができる．

この方式には R&R-GKA と呼ばれる鍵共有法が必要であり，任意のユーザが鍵を更新することが求められる．しかし，提案されている鍵共有方式では，1 ユーザの追加，1 ユーザの削除のスキームは提案されており，鍵更新もその応用とされているが，複数ユーザが一度に鍵更新を行う方式については述べられていない．本研究では，複数ユーザが一度に鍵更新を行う方式を提案し，効率性の観点から評価する．

2. グループ鍵管理

グループ鍵管理にはいくつかの方式がある．集中型鍵配布は 1 つのエンティティ（鍵サーバ）が鍵を生成しそれをグループのメンバに配布する方式である．実際の鍵配布には，鍵サーバは各ユーザと長期秘密鍵を共有しなければならない．非集中型鍵配布は動的に選ばれるグループのメンバが鍵を生成し他のグループメンバに配布する方式である．この方式はグループの分割などが発生しても，他の鍵サーバを選ぶことで運用を継続できる．欠点は集中型と同じく，鍵サーバがグループ鍵を配布するための安全なチャネルを必要とすることと，1 つのエンティティが鍵生成を行うということの信頼性についての問題である．分担グループ鍵管理は，各メンバの秘密鍵の関数としてグループ鍵を計算するもので，集中型の鍵サーバの信頼性の問題や一部の故障に関する問題を避けることができる．さらに，グループメンバとの安全なチャネルを必要としない方式もあり，実用的である．このようなグループ鍵共有プロトコルの大きな問題点は実行間のユーザの故障やメンバの変更の発生である．

グループ鍵共有は多数提案されており，多くは Diffie-Hellman 鍵共有を複数のユーザに拡張したものである．特に Desmedt らは BD スキームと呼ばれるグループ鍵共有方式を

Vol.2009-SE-164 No.18
Vol.2009-EMB-13 No.18
Vol.2009-CSEC-45 No.18
2009/5/29

提案した[BD94]．またそれらを拡張して木構造を用いたもの[DLB07]，ペアリングを用いたもの[DL08] なども提案されている．

Canard らはこのようなグループ鍵共有方式を用いたグラフ鍵管理スキームについて提案した[CJ08]．グラフ鍵管理の定義は以下のとおりである．グラフを G とする．l 人のプレーヤ $\mathcal{P}_1, \ldots, \mathcal{P}_l$ の集合を $\mathcal{P}$ と表す．各プレーヤ $\mathcal{P}_i$ はグラフにおけるノード i に対応する．グラフ G の構成は全プレーヤが既知とする．

安全性の概念について導入する．グラフ鍵管理スキームは鍵リカバリ安全性 (Key Recovery security property) が必要であり，以下の試行 (Experiment) に対応する．

Experiment $Exp^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}$:

(1) チャレンジャ$\mathcal{C}$ はシステムを初期化しグラフを攻撃者 $\mathcal{A}$ に送る．

(2) $\mathcal{A}$ はシステムと対話し，鍵を更新，プレーヤ・鍵をコラプトする．試行の間コラプトされていない鍵が少なくとも 1 つ存在するとする．

(3) $\mathcal{A}$ は最終的にグラフ鍵管理の ID π，ノード i，インスタンス ρ，コラプトしていない鍵 k を出力する．

この試行における攻撃者 $\mathcal{A}$ の成功確率を

$$Succ^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}(\tau) = \Pr[k = k_i[\pi, \rho]]$$

と定義する．

定義 1 (鍵リカバリ) GKM スキームが鍵リカバリ安全性を満たすとは，$Succ^{\mathsf{keyrecovery}}_{GKM,\mathcal{A}}(\tau)$ が無視できるほど小さいことである．

3. Canard らの鍵管理方式

Canard らはグラフ鍵管理方式として，HMAC 関数と R&R GKA を利用した方式を提案した[CJ08]．グラフにおいて親が 1 ノードである場合は HMAC を用いて子の鍵を生成し，親が複数の場合は R&R GKA を用いて子の鍵を生成する．

3.1 HMAC 関数

暗号メッセージ認証子 (MAC :message authentication code) は共通鍵暗号に属するメッセージ認証の手段である．MAC アルゴリズムは鍵生成アルゴリズム KeyGen(MAC 鍵 K を生成)，認証子生成アルゴリズム MAC(入力:$K, m \in \{0,1\}^*$, 出力:Σ)，検証アルゴリズム VerMAC(入力:m, Σ，出力:1/0) からなる．MAC スキームは選択メッセージ攻撃における存在的偽造に対して耐性をもたなくてはならない．攻撃者 $\mathcal{A}$ が選んだメッセージに対する MAC を生成するオラクルにアクセスできたとしても，$\mathcal{A}$ はオラクルに聞いたことのないメッセージの MAC を推測することができない．すなわち $\mathcal{A}$ は任意の長さのデータ d^* を選び，それを認証子生成オラクルに送る．認証子生成オラクルは，ランダムに鍵 k を選び，d^* を鍵 k を用いて $t^* = \mathsf{MAC}(k, d^*)$ を計算し，それを $\mathcal{A}$ に送る．最終的に，攻撃者は $\hat{d} \neq d^*$ であるような $(\hat{d}, \hat{t})$ を出力する．$\Pr[\hat{t} = MAC(k, \hat{d})]$ を攻撃者のアドバンテージ $\mathsf{Adv}_{MAC}(\mathcal{A})$ と定義し，これが小さいとき MAC は安全であるとする．

今回のスキームではこれに加えて擬似乱数性も必要である．そのため，Bellare により疑似乱数関数と同等と証明されている HMAC を使用する．

3.2 Refreshable and Replayable Group Key Agreement

グループ鍵共有 (GKA) スキームは公衆網で参加者のグループがそれぞれの貢献で暗号鍵を共有するものであり，いくつかの GKA スキームが提案されている．今回のスキームで必要とする追加の性質であるリフレッシュ可能・再生可能グループ鍵共有 (Refreshable and Replayable Group Key Agreement(R&R-GKA)) の概念を導入する．$\mathcal{P}$ は GKA の潜在的参加者であり，$\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_l\}$ である．

定義 2 (R&R-GKA)

R&R-GKA スキームは以下のアルゴリズムからなる．

- Setup セットアップアルゴリズム．
  入力：τ：セキュリティパラメータ
  出力：Γ：システムパラメータ，システムパラメータにセキュリティパラメータを追加．
- UserSetup ユーザセットアップアルゴリズム．
  入力：Γ：システムパラメータ
  出力：(sk_i, pk_i)：$\mathcal{P}$ の各プレーヤに対する長期鍵ペア．システムパラメータに全プレーヤの公開鍵を追加．
- KeyGeneration 鍵生成アルゴリズム．
  入力：$\mathcal{I} \subset \mathcal{P}$：プレーヤの集合，$\Gamma$：システムパラメータ，$(sk_i, pk_i)$：ユーザの長期鍵ペア．
  出力：$K[\mathcal{I}, 0]$：共有鍵の初期インスタンス，$PE[\mathcal{I}, 0]$：公開情報の初期インスタンス．
- KeyRefresh 鍵更新アルゴリズム．
  入力：$\mathcal{I} \subset \mathcal{P}, \mathcal{J} \subset \mathcal{I}$：鍵更新が必要なユーザの集合
  出力：$K[\mathcal{I}, \rho + 1]$：共有鍵の新インスタンス，$PE[\mathcal{I}, \rho + 1]$：公開情報の新インスタンス

Vol.2009-SE-164 No.18
Vol.2009-EMB-13 No.18
Vol.2009-CSEC-45 No.18
2009/5/29

- KeyRetrieve 鍵検索アルゴリズム．
  入力：$\mathcal{I} \subset \mathcal{P}, \mathcal{P}_i \in \mathcal{I}$：プレーヤ，$\rho$：インスタンス
  出力：$K[\mathcal{I}, \rho]$：共有鍵の ρ 番目のインスタンス

3.3 Canard らの方式[CJ08)]

Canard らは任意の有向グラフに配置された 2 者が安全に情報を交換できるようなスキームを提案した[CJ08)]．HMAC とバーチャルノードを使用することで，任意の階層構造で上位ノードは下位ノードの鍵を生成できるが，逆はできないという鍵管理を行うスキームである．複数の親が子ノードを共有している場合は，子と親の間にバーチャルノードを挿入し，ノード鍵を割り当てることで下位ノードの鍵を生成する．

- Setup セットアップアルゴリズム．
  入力：τ：セキュリティパラメータ
  出力：Γ：システムパラメータ，システムパラメータにセキュリティパラメータを追加．
- UserSetup ユーザセットアップアルゴリズム．
  入力：Γ：システムパラメータ
  出力：(sk_i, pk_i)：$\mathcal{P}$ の各プレーヤに対する長期鍵ペア．システムパラメータに全プレーヤの公開鍵を追加．
- KeyGeneration 鍵生成アルゴリズム．各ノード i によって 3 つのケースがある．
  - i がグラフ中に親をもたない場合，ノード鍵 k_i は $\{0,1\}^\tau$ からランダムに選ばれる．
  - i が 1 つの親 f をもつ場合，s_f を f の子の鍵を計算された数とすると，
    $$k_i[0] = M.\mathsf{MAC}(k_f[0], C||s_f + 1)$$
    公開情報は $pk_i = s_f + 1||i$
  - i が F_i 個の親 $(f_1, \ldots, f_{F_i})$ をもつとき，R&GKA の KeyGeneration を入力 $\mathcal{I} = \{f_1, \ldots, f_{F_i}\}$ として実行する．
- KeyDerivation 鍵導出アルゴリズム．
  - i が 1 つの親 f をもつ場合，$s_v||v$ を v の公開情報とすると，
    $$k_v[\rho] = M.\mathsf{MAC}(k_f[\rho], C||s_v)$$
  - i が F_v 個の親 $(f_1, \ldots, f_{F_v})$ をもつとき，公開情報を $pk_v = PE[\mathcal{V}, \rho]||v$ とすると，GKA.KeyRetrieve を入力 $\mathcal{V} = \{f_1, \ldots, f_{F_i}\}, f, \rho$ として実行する．
- Key Refresh 鍵更新アルゴリズム．
  バーチャルノードを用いているので，更新するノードは必ず 1 つの親をもつ．現在のインスタンスを ρ とするとき，$k_f[\rho+1] = k_f[\rho]$ とする．この親が子に対して鍵を計算した回数を s_f とすると，
  $$k_j[\rho+1] = M.\mathsf{MAC}(k_f[\rho+1], C||s_f + 1)$$
  となり，対応する公開情報は $pk_j = s_f + 1||j$ である．

この方式では，グループ鍵共有方式をサブルーチンとして呼び出しているのみで，具体的な方式については提案されていない．

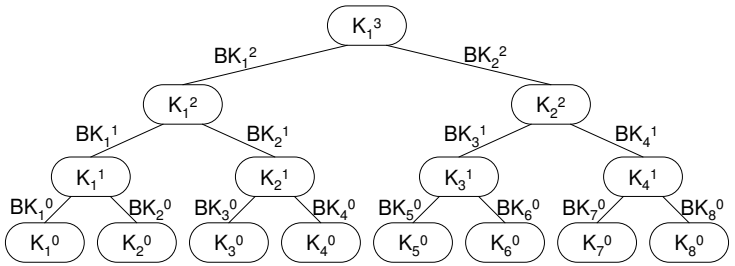

図 1 鍵ツリーの例

4. 鍵ツリー

鍵ツリー構造は従来中央集中型グループ鍵分配方式として提案されたものである．これを完全分散型鍵共有として用いた方式が提案されており，TGDH(Tree-based Group Diffie-Hellman) と呼ばれている[KPT04)]．$n \in \mathbb{N}$，$X = (N_1, N_2, \ldots, N_n)$，$N_i \in G = \langle \alpha \rangle$ とし，鍵ツリー T は n 個の葉ノードをもつ．図 1 に $n = 8$ のときの鍵ツリーの例を示す．表中の K_j^i は i 番目のレベルにおける j 番目の鍵を表し，各葉ノードにはセッションごとランダムな値 $K_j^0 = N_k (k \in [1, n])$ が割り当てられている．BK_j^i は i 番目のレベルにおける j 番目の公開鍵を表し，$\alpha^{K_j^i}$ である．K_j^i は再帰的に以下のように定義される．$K_j^i = \alpha^{K_{2j-1}^{i-1} K_{2j}^{i-1}} = (BK_{2j-1}^{i-1})^{K_{2j}^{i-1}} = (BK_{2j}^{i-1})^{K_{2j-1}^{i-1}}$．

4.1 TGDH のメンバイベント

TGDH においては次のメンバ変更イベントについて考慮されている．

- Join（ユーザの追加）
- Leave（ユーザの削除）
- Mearge（グループの合併）
- Partition（グループの分離）

Vol.2009-SE-164 No.18
Vol.2009-EMB-13 No.18
Vol.2009-CSEC-45 No.18
2009/5/29

- Key refresh（グループ鍵の更新）

グループ鍵の更新 (Key refresh) については，Leave プロトコルにおいて実際に削除されない特別のケースとみなしている．

4.2 ユーザ追加のアルゴリズム

グループは n メンバ $\{M_1, \ldots, M_n\}$ で構成されていると仮定する．

(1) JoinRequest：新メンバ M_{n+1} は自身の公開鍵 $BK_{\langle 0,0 \rangle}$ を含む追加リクエストをブロードキャストする．

(2) DetermineInsertionPoint：木における新メンバの挿入ポイントを決定する．挿入ポイントは最も浅いレベルにある最も右のノード（この場合，木の高さは変わらない），又は木が完全バランス木である場合はルートノードである．

(3) SponsorSelect：スポンサとして，挿入ノードをルートとする部分木の一番右の葉ノードを選ぶ．

(4) MakeNodes：中間ノードと新メンバノードを作成する．中間ノードは挿入ノードと新メンバノードの親になるようにする．

(5) GenerateKeys：スポンサは新しい木と全ての公開鍵をブロードキャストする．

(6) ComputeKey：各メンバは新しいグループ鍵を計算する．

例として 3 ユーザに 1 ユーザを追加する場合を図 2 に示す．$\{M_1, M_2, M_3\}$ というグループに M_4 を追加している．追加操作には 2 ラウンドと 3 メッセージを必要とし，計算量は追加される場所にもよるが，6 から $3h-3$（h は実行後の鍵ツリーの高さ）のモジュロ指数演算が必要である．最も計算量を必要とするのはルートノードへのメンバ追加である．最も少ない場合は，図のようなルートが 1 つのノードをもつ場合である．

4.3 ユーザ削除のアルゴリズム

グループには n メンバ $\{M_1, \ldots, M_n\}$ がいると仮定する．メンバ M_d がグループから削除されるとする．

(1) SponsorSelect：スポンサとして，削除されるノードの兄弟ノードをルートとする部分木の一番右の葉ノードを選ぶ．

(2) UpdateKeyTree：M_d に対応する葉ノードを削除して鍵ツリーを更新する．M_d の元兄弟は親ノードと置き換える．

(3) GenerateKeys：スポンサは自分のノードからルートまでのパス上の秘密鍵・公開鍵の新しい組 $[key, bkey]$ を計算し，全ての公開鍵をブロードキャストする．

(4) ComputeKey：各メンバは新しいグループ鍵を計算する．

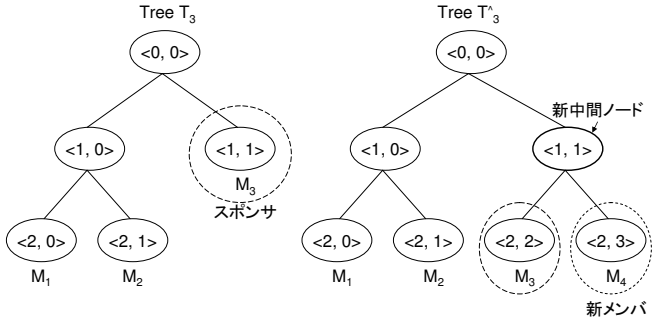

図 2 追加アルゴリズム

例として 5 ユーザから 1 ユーザを削除する場合を図 3 に示す．

図 3 では，メンバ M_3 を削除する場合を示している．残りのメンバはノード $\langle 1,1 \rangle$ と $\langle 2,2 \rangle$ を取り除く．その後，スポンサの M_5 は新しい鍵 $K_{\langle 2,3 \rangle}$ を選び，$K_{\langle 1,1 \rangle}, K_{\langle 0,0 \rangle}, BK_{\langle 2,3 \rangle}, BK_{\langle 1,1 \rangle}$ を計算し，新しい木に公開鍵をブロードキャストする．このブロードキャストメッセージを受信したメンバはグループ鍵を計算する．しかし，削除された M_3 は自分の秘密鍵がグループ鍵に含まれていないため，共有鍵の計算は不可能である．

削除プロトコルの実行には 1 ラウンドと 1 メッセージ（削除ユーザを除いた木を構成し，対応する公開鍵のブロードキャスト）を必要とする．計算量は去るメンバの位置と木構造によるが，その上界は $3h-3$ 回のモジュロ指数演算である．

4.4 複数ノードの削除（分割プロトコル）

n メンバの分割があった場合，残りの各メンバから見ると複数メンバの同時削除とみなされる．よって，分割プロトコルは複数ラウンドを必要とし，新しいグループ鍵を生成できるまで繰り返される．

(1) UpdateKeyTree：残りの各メンバは分割されたメンバとその親のノードを一番深い

Vol.2009-SE-164 No.18
Vol.2009-EMB-13 No.18
Vol.2009-CSEC-45 No.18
2009/5/29

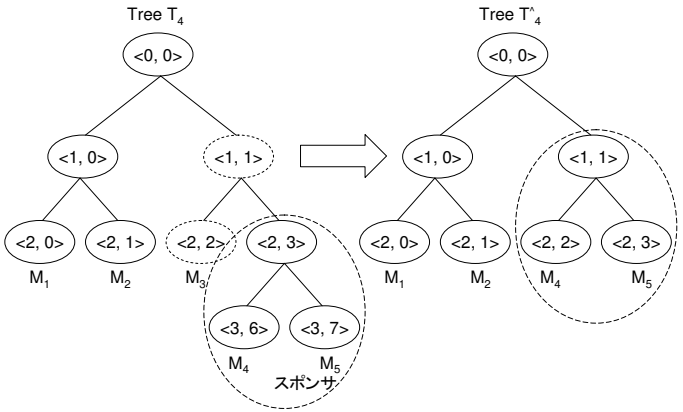

図 3 削除アルゴリズム

位置にあるノードから順番に削除して鍵ツリーを更新する．

(2) SponsorSelect：スポンサとして，削除されるノードの兄弟ノードをルートとする部分木の一番右の葉ノードを選ぶ．

(3) GenerateKeys：スポンサは自分のノードからルートまでのパス上の秘密鍵・公開鍵の新しい組 $[key, bkey]$ を計算し，全ての公開鍵をブロードキャストする．

(4) ComputeKey：各メンバは新しいグループ鍵を計算する．

分割プロトコルに必要な計算量は，削除プロトコルと同様のアルゴリズムであるので，同等である．n メンバから r メンバが削除されるとする．削除されるノードは異なる位置にあるため，それぞれの削除は並列に実行される．このとき，残りの各メンバは r 個の削除ノードとその親をそれぞれ第 1 ラウンドで削除して木構造を更新するため，各鍵ツリーは公開鍵をもたないパスを最大 r 個もつことになる．これらの公開鍵を埋めるためには r 個のパスに対してのなるべく高い位置で公開鍵を計算するか，高々木の高さ h 回のラウンドとなるため，$\min(\lceil \log_2 r \rceil + 1, h)$ ラウンドを必要とする．また通信量は，メッセージ数 $\min(2r, \lceil \frac{n}{2} \rceil)$ である．

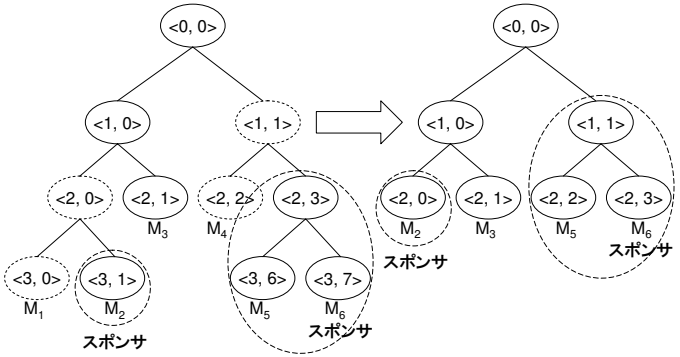

図 4 分割アルゴリズム

4.5 合併プロトコル

k 個のグループが合併する場合については以下のとおりである．

(1) SponsorSelect：スポンサとして各グループの一番右のメンバを選ぶ．

(2) GenerateKeys：各スポンサは全ての公開鍵を他のグループへ送る．

(3) ComputeKey：各メンバは新しいグループ鍵を計算する．

合併プロトコルには最大 $\lceil \log_2 k \rceil + 1$ ラウンドが必要である．メッセージの最大量は $2k$ であり，計算量は追加プロトコルと同じく $3h - 3$ である．

5. 複数ノードの鍵更新についての考察

5.1 鍵 更 新

Canard らの方式[CJ08] は HMAC と R&R GKA を必要とする．その中で R&R GKA として紹介されている鍵共有方式の 1 つである鍵ツリーを用いた方式を適用しようとすると，鍵更新アルゴリズムは削除アルゴリズムの応用として実行されるため，そのまま利用するのでは効率が悪い．既存の鍵共有プロトコルの鍵更新アルゴリズムを効率的にすることで，Canard らの方式をより改善できると考える．

5.2 既存方式の問題点

Canard らのグラフ鍵管理方式において必要とされる R&R GKA スキームは既存の鍵ツリーを用いた方式[KPT04] を用いて実現される．提案されている鍵ツリーの方式における，それぞれのユーザ追加・削除の効率を表 1 に示す．

Vol.2009-SE-164 No.18
Vol.2009-EMB-13 No.18
Vol.2009-CSEC-45 No.18
2009/5/29

	ラウンド数	計算量	通信量
1 ユーザ削除	1	$3h-3$	1
1 ユーザ追加	2	$3h-3$	3
複数ユーザ削除	$\min(\lceil \log_2 r \rceil + 1, h)$	$3h-3$	$\min(2r, \lceil \frac{n}{2} \rceil)$
複数ユーザ追加	$\lceil \log_2 k \rceil + 1$	$3h-3$	2k

表 1 各アルゴリズムの効率の比較

ユーザの追加アルゴリズムに必要な情報は，追加されるノードの公開鍵，新しい木の構造，新しい公開鍵であり，それぞれを全ノードにブロードキャストしなければならない．合計 2 ラウンドと 3 メッセージのブロードキャストが必要とされている．しかし，ユーザ追加については，ラウンド数は 2 回必要であるが，実際に必要なデータは，追加ノードの公開鍵のブロードキャストと，追加ノードが必要とするその兄弟ノードの公開鍵のユニキャスト（兄弟ノードから追加ノード）のみであり，通信量の削減は可能であると考える．

ユーザの削除アルゴリズムは 1 ラウンドを要し，新しい木構造と新しい公開鍵全てのブロードキャストが必要である．各ユーザは自分からルートまでのパスの鍵を全て再計算する必要があるため，計算量は $O(\log n)$ である．このアルゴリズムはスポンサノードが新しい秘密鍵，公開鍵を作成してブロードキャストを行うように構成されている．しかし，削除されるノードの除去という操作により新たな木構造を構成しているため，以前に公開した公開鍵を用いても，新たな共有鍵を計算できると考える．そうすれば，スポンサノードは高々 $\log n - 1$ 回のモジュロ指数演算のみで新たな公開鍵を作成できる．

複数ユーザの削除は，1 ユーザ削除アルゴリズムを複数ラウンドを実行するものであり，深い位置にあるユーザを逐次削除していくというアルゴリズムである．各削除ノードのスポンサも 1 ノード削除のプロトコルと同じようにして決定される．また，鍵の独立性のために，残ったノードのうち 1 つは新しい秘密鍵への更新が必要とされ，1 回目のラウンドでその更新を実行する．今回考える鍵更新では，この独立性については考慮しなくてもよいと考える．

全ての追加・削除アルゴリズムにおいて計算量は実行後の木の高さ h に依存している．また，複数ユーザの削除の通信量は削除されるユーザ数の 2 倍か，全ユーザ数の半分となっている．

ここで，鍵更新のプロトコルはユーザ削除プロトコルを応用したものと提案されている．複数ユーザの削除も実際は 1 ユーザごとの削除を複数回行うものである．Canard らの方式[CJ08] はノードの子に対して HMAC で鍵を導出して高速化を図っているが，複数の親が子を共有している場合に R& RGKA を使うと，効率が落ちてしまう．特に複数の親ノードに鍵更新が行われた場合，グループメンバは変わらないので，高速に変更した鍵を共有する必要がある．そのため，鍵ツリーを用いたグループ鍵共有における複数ノードの鍵更新について考察する．

鍵ツリーのスキームで用いられている木構造は二分木なので，ある葉ノードの鍵更新は親ノードの鍵の変更を必ず伴う．また，上位ノードの鍵も下位ノードで共有した鍵に基づき生成されるため，結果として新たな鍵計算が $\log n = h$（n は全体のユーザ数）回必要となる．

鍵更新はユーザの削除及び新ユーザの追加として実現できるが，実際はユーザが新しい公開鍵をブロードキャストし，それを用いて各ユーザが新しい共有鍵を生成することができる．しかし，この場合各ノード鍵も更新しなければならないため，そのノードからルートまでのパスの鍵計算のため，$O(\log n)$ の計算量が必要である．複数ユーザの鍵更新では，ラウンド数がユーザ数と同じだけ必要となってしまい，効率が悪い．

5.3 既存方式の考察

上記の問題点を踏まえて，既存方式について以下のとおり考察する．

- 既存方式においては，鍵独立性の問題を回避するために更新ノードのスポンサの鍵も更新しており，ブロードキャストデータが大きくなっている．
- 以前に公開鍵としてブロードキャストしたデータは再度ブロードキャストを行わなくてもメンバに既知のものとして取り扱うことができる．
- 複数ユーザの削除のアルゴリズムは削除ユーザの log オーダ又は木の高さのオーダのラウンド数を必要としているが，一定ラウンドとすることが望ましい．数ラウンドで鍵を共有できる鍵共有プロトコルは多く提案されており，鍵更新のために効率が落ちることは避けなければならない．

6. おわりに

任意の階層構造の鍵管理に必要なリフレッシュ&再実行可能なグループ鍵共有方式についての考察を述べた．また，新プロトコルの構築，安全性の厳密な証明については今後の課題である．同様の木構造を使ってペアリングを用いた 3 者鍵共有を基にしたプロトコル[DB05] に対しても鍵更新のアルゴリズムを考えることができるか検討中である．

Vol.2009-SE-164 No.18
Vol.2009-EMB-13 No.18
Vol.2009-CSEC-45 No.18
2009/5/29

参 考 文 献

[CJ08] Sebastien Canard, Amandine Jambert, “Group Key Management: From a Non-hierarchical to a Hierarchical Structure”, In INDOCRYPT 2008, LNCS 5365, pp.213-225, 2008.

[KPT04] Yongdae Kim, Adrian Perrig, Gene Tsudik, “Tree-Based Group Key Agreement”, ACM Transactions on Information and System Security (TISSEC), Volume 7, Issue 1 (February 2004)pp.60 - 96, 2004.

[DB05] Ratna Dutta, Rana Barua, “Dynamic Group Key Agreement in Tree-Based Setting”, In ACISP 2005, LNCS 3574, pp.101-112, 2005.

[BD94] M. Burmester and Y. Desmedt., “A secure and efficient conference key distribution system, ” In Advances in Cryptology - EUROCRYPT ’94, 1995.

[DL08] Y. Desmedt, T. Lange, “Revisiting Pairing Based Group Key Exchange, ” In FC 2008. LNCS, vol. 5143, pp. 53-68. Springer, Heidelberg (2008).

[DLB07] Yvo Desmedt, Tanja Lange, Mike Burmester, “Scalable Authenticated Tree Based Group Key Exchange for Ad-Hoc Groups” In FC 2007 and USEC 2007, LNCS 4886, pp. 104-118, 2007.