SCIS 2018 2018 Symposium on
Cryptography and Information Security
Niigata, Japan, Jan. 23 - 26, 2018
The Institute of Electronics,
Information and Communication Engineers

KEM を用いた動的多者鍵配布プロトコル
Dynamic Multi-Cast Key Distribution with KEM

金城 皓羽 * 岡野 裕樹 * 齋藤 恆和 * 草川 恵太 * 小林 鉄太郎 *
Koha Kinjo Yuki Okano Tsunekazu Saito Keita Xagawa Tetsutaro Kobayashi
星野 文学 *
Fumitaka Hoshino

あらまし RSA 暗号や楕円曲線暗号は SSL/TLS による暗号化通信などに使用されている．これらは量子コンピュータの解読に耐性がないことが証明されているため, 量子コンピュータの解読に耐性のある暗号 (ポスト量子暗号) の研究が盛んに行われている．また, NIST はポスト量子暗号の標準化に向けて, 2017 年 11 月までこれの公募を行い, 今後これの選定を行っていく計画を出している．一方で, ユーザエンドでの End-to-End 暗号化 (E2EE) への関心が高まっており, コンシューマ向け, ビジネス向け問わず広く取り入れられている．Yoneyama らは, 多人数での通信における E2EE を実現するために, 通信効率の良い動的多者鍵配布 (DMKD) プロトコルの具体的な方式を提案している．しかし, この DMKD プロトコルは量子コンピュータの解読に耐性のない Diffie-Hellman(DH) 鍵交換を利用している．本研究では量子コンピュータの実用化に備え, Yoneyama らの DMKD プロトコルをモデルとした, これに耐性のあるプロトコルの構築を目的とした．ポスト量子暗号の一つに格子暗号があるが, これによる鍵交換は鍵カプセル化メカニズム (KEM) により実現されている．本稿では二者間鍵交換を KEM で記述し, 格子暗号をはじめとしたポスト量子暗号が適用可能な DMKD プロトコルの一般的な構成を提案する．

キーワード 鍵配布, End-to-End 暗号化, 鍵漏洩耐性

1 はじめに

近年量子コンピュータに関する研究が盛んに行われており, 例えば IBM は 2017 年に 50 量子ビットのシステムの構築に成功している [7]．これの研究が進み, 大規模なものが構築された場合, 現在使用されている公開鍵暗号システムの多くは破壊されてしまう [14]．例えば, SSL/TLS やデジタル署名などに使用されている RSA や楕円曲線暗号 [20] は, 量子コンピュータの解読に耐性がない [19]．アメリカ国立標準技術研究所 (NIST) は量子コンピュータ時代の到来の正確な時期を予測できるかに依らず, これの解読に耐性のある暗号 (ポスト量子暗号) インフラを整備する必要があるとしており [14], ポスト量子暗号標準化の計画を進めている．

一方, Snowden が NSA の情報収集に関する手口を告発した [21] ことを受け, セキュリティにへの関心が高まっている [18]．ユーザエンドでの End-to-End 暗号化 (E2EE) もその一つで, チャットアプリケーションや電話会議など, コンシューマ向けのものからビジネス向けのものまで幅広くこれを取り入れている．E2EE を実現する鍵交換技術は大きく二者間での鍵交換と多人数での鍵交換に区分することができる．二者間での鍵交換は暗号技術の基本要素であり, ポスト量子暗号標準化では二者間での鍵交換は対象に含まれているが, 多人数での鍵交換は対象でない [14]．そのためこの標準化を踏まえた, 多人数での鍵交換プロトコルの検討はきわめて重要な課題になる．

この検討として本稿では, ユーザの出入りによって鍵の再配布を必要としない動的多者鍵配布 (DMKD) に着目した．これはユーザの出入りの多いチャットアプリケーション等に有効で, ユーザエンドの E2EE を実現するものには Yoneyama らの提案しているプロトコル [10] がある．Yoneyama らの DMKD プロトコルは, DH 鍵交換を用いた効率的な鍵交換を実現しているが, DH 鍵交換は量子コンピュータの解読に耐性がない [19]．そこで本研究では, Yoneyama らの DMKD プロトコルをモデルとし, 量子コンピュータの解読に耐性のある DMKD プロトコルの構築を検討した．

* NTT セキュアプラットフォーム研究所, 180-8585, 東京都武蔵野市緑町 3-9-11, NTT Secure Platform Laboratories, 3-9-11, Midori-Cho Musashino-Shi, Tokyo, 180-8585.

1.1 研究成果

ポスト量子暗号の代表的なものには, 格子暗号や, 符号ベース暗号がある. これらは鍵カプセル化メカニズム (KEM) で鍵交換が構成されており (例えば [2][15]), DH 鍵交換のような構成は難しい. そこで, 本稿では KEM を用いた一般的な構成での DMKD プロトコルを考案した. これにより, Yoneyama らの定義する DMKD プロトコルへの安全性 (DMKD 安全性 [10]) を保持しつつ, ポスト量子暗号の適用が可能なプロトコルとなった.

1.2 関連研究

NIST のポスト量子暗号標準化に応募されたものは, 格子暗号, 符号ベース暗号で多数を占めている. 格子暗号は Ajtai[13] を, 符号ベース暗号は McEliece[17] をはじめとした研究が広くなされている [11][8][16][3]. 多人数での鍵交換には大きく Group Key Exchange(GKE) と Multi-cast Key Distribution(MKD) がある. GKE はメッシュ型, MKD はツリー型ないしはスター型の鍵交換になる. GKE は Bresson[5] をはじめとして研究が行われてきたが, Suzuki らの定義している安全性 [12] を担保すると, 鍵交換に参加人数に依存した通信コストがかかる. MKD では Ballardie[1] をはじめとして研究が行われており, サーバを中心においたスター型のものは参加人数に依存しない鍵交換を可能にするが, Micciancio らの定義している安全性 [4] ではサーバに共有鍵の取得を許すことになる. そこで Yoneyama らはサーバへの共有鍵の秘匿を要件に加えた DMKD 安全性を定義し, これを満たすプロトコルを提案している [10]. 本研究の位置づけは Yoneyama らの DMKD プロトコルをモデルとした, 量子コンピュータの解読に耐性のあるプロトコルの構成で, 本稿ではポスト量子暗号が適用可能なこれの構築を行っている.

2 準備

本稿では, 確率的多項式時間アルゴリズム ALG に対して, $y \leftarrow ALG(x)$ はアルゴリズムに x を入力すると y を出力すること表す. 特に, 乱数 r を固定した出力 $y \leftarrow ALG(x;r)$ はアルゴリズムに x を入力とし, 確定的に y を出力することを表すものとする. 集合 X に対して $y \xleftarrow{r} X$ は X から一様ランダムに y を出力すること表す.

2.1 使用アルゴリズムと安全性の定義

本稿プロトコルで使用するアルゴリズムの定義をする. κ をセキュリティパラメタ, $F = \{F_\kappa : Dom_\kappa \times Kspace_\kappa \rightarrow Rng_\kappa\}_\kappa$ を関数族, $\{Dom_\kappa\}_\kappa$ を関数族の定義域, $\{Kspace_\kappa\}_\kappa$ を鍵空間, $\{Rng_\kappa\}_\kappa$ を値域とする.

定義 2.1 (擬似ランダム関数)
関数族 $F = \{F_\kappa\}_\kappa$ が擬似ランダム関数であるとは, 次を満たすことである. 関数 $F = \{F_\kappa\}_\kappa$ は任意の確率的多項式時間アルゴリズム $\mathcal{D}$ に対し, $|Pr[1 \leftarrow \mathcal{D}^{F_\kappa(\cdot)}] - Pr[1 \leftarrow \mathcal{D}^{RF_\kappa(\cdot)}]| < negl$ (RF は真のランダム関数).

ここで擬似ランダム関数 F_κ を利用して, ねじれ疑似ランダム関数 $tPRF : \{0,1\}^\kappa \times Kspace_\kappa \times \{0,1\}^\kappa \times Kspace_\kappa$ を次で定義する, $tPRF(a,a',b,b') := F_\kappa(a,a') \oplus F_\kappa(b,b')$. また, これに関して以下が成り立つ.

補題 2.1
F_κ が疑似ランダム関数ならば,
$[(a,a'), tPRF(a,a',b,b')]$ は $[(a,a'),R]$(R は Rng_κ 上の一様乱数) と区別される確率が negligible である.
$[(b,b'), tPRF(a,a',b,b')]$ は $[(b,b'),R]$(R は Rng_κ 上の一様乱数) と区別される確率が negligible である.

定義 2.2 (衝突困難関数)
関数 $TCR : Dom_\kappa \rightarrow Rng_\kappa$ が衝突困難関数であるとは, 次を満たすことである. 関数 TCR は任意の確率的多項式時間アルゴリズム $\mathcal{A}$ に対し, $Pr[x \xleftarrow{r} Dom_\kappa; x' \leftarrow \mathcal{A}(x) \ \ s.t. \ \ x \neq x' \wedge TCR(x) \neq TCR(x')] < negl$

定義 2.3 (公開鍵暗号)
公開鍵暗号アルゴリズムは以下の 3 つ組のアルゴリズム

- PGen: 1^κ を入力として, 公開鍵と秘密鍵の組 (pk, sk) を出力する.
- PEnc: 公開鍵 pk, 平文 m を入力として, 暗号文 c を出力する.
- PDec: 暗号文 c, 秘密鍵 sk を入力として, 平文 m もしくは復号不可を意味する $\perp$ を出力する.

からなるアルゴリズムで, かつ暗号文が平文に正しく復号されるアルゴリズム.

定義 2.4 (CCA 安全な公開鍵暗号)
公開鍵暗号アルゴリズムが CCA 安全であるとは以下を満たすことである. 任意の確率的多項式時間アルゴリズム $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ に対して, $|\Pr[(pk,sk) \leftarrow \mathrm{Gen}(1^\kappa); (m_0,m_1,s) \leftarrow \mathcal{A}_1^{\mathcal{DO}(sk,\cdot)}(pk); b \leftarrow \{0,1\}; CT^* \leftarrow \mathrm{Enc}(pk,m_b); b' \leftarrow \mathcal{A}_2^{\mathcal{DO}(sk,\cdot)}(pk, CT^*, s); b' = b] - \frac{1}{2}| \leq negl$

定義 2.5 (Ciphertext-Policy 属性ベース暗号)
Ciphertext-Policy 属性ベース暗号アルゴリズム (CP-ABE) は以下の 4 つ組のアルゴリズム

- Setup: 1^κ, att(属性区分) を入力として, 公開パラメタ $Params$ とマスタ秘密鍵 msk を出力する.

- Der: 公開パラメタ $Params$, マスタ秘密鍵 msk, 属性 A 入力として, ユーザ秘密鍵 usk_A を出力する.
- AEnc: 公開パラメタ $Params$, アクセス情報 P と平文 m 入力として, 暗号文 c を出力する.
- ADec: 公開パラメタ $Params$, ユーザ秘密鍵 usk_A と暗号文 c 入力として, 平文 m を出力する.

からなるアルゴリズムで, かつ暗号文がすべての属性に対して平文に正しく復号されるアルゴリズム.

定義 2.6 (CCA 安全な CP-ABE)
Ciphertext-Policy 属性ベース暗号が CCA 安全であるとは以下を満たすことである. 任意の確率的多項式時間アルゴリズム $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ に対して, $|\Pr[(Params, sk) \leftarrow \mathrm{Setup}(1^\kappa, att); (m_0, m_1, P^*, s) \leftarrow \mathcal{A}_1^{\mathcal{EO}(Params,msk,\cdot),\mathcal{DO}(Params,usk,\cdot)}(Params); b \leftarrow \{0,1\}; CT^* \leftarrow \mathrm{AEnc}(Params, P^*, m_b); b' \leftarrow \mathcal{A}_2^{\mathcal{EO}(Params,msk,\cdot),\mathcal{DO}(Params,usk,\cdot)}(Params, CT^*, s); b' = b] - \frac{1}{2}| \leq negl$. 特に, $\mathcal{A}$ が Setup 前に P^* を決定する条件を付加したとき, これを selective-CCA 安全であるという.

定義 2.7 (メッセージ認証子)
メッセージ認証子は以下の 3 つ組のアルゴリズム

- MGen: 1^κ を入力として, 公開鍵と秘密鍵のペア (pk, sk) を出力する.
- Tag: *MAC* 鍵 mk と平文 m を入力として, タグ σ を出力する.
- MVer: *MAC* 鍵 mk とタグ σ と平文 m を入力として, 承認ならば 1, 否認ならば 0 を出力する.

からなるアルゴリズムで, かつ正しいタグに対しては必ず承認を出すアルゴリズム.

定義 2.8 (UF-CMA なメッセージ認証子)
メッセージ認証子が *UF-CMA* であるとは以下を満たすことである. 任意の確率的多項式時間アルゴリズム $\mathcal{A}$ に対して, $\Pr[(mk) \leftarrow \mathrm{MGen}(1^\kappa); (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{MO}(mk,\cdot)}; 1 \leftarrow \mathrm{MVer}(mk, m^*, \sigma^*)] \leq negl$

定義 2.9 (鍵カプセル化メカニズム)
鍵カプセル化メカニズムは以下の 3 つ組のアルゴリズム

- KGen: 1^κ を入力として, 公開鍵と秘密鍵の組 (pk, sk) を出力する.
- KEnc: 公開鍵 pk を入力とし, 鍵 K とその暗号文 C を出力する.
- KDec: 公開鍵 pk と秘密鍵 sk と暗号文 C を入力とし, 鍵 K または復号不可を意味する $\perp$ を出力する.

からなるアルゴリズムで, かつ正しく鍵を暗号化しているものは正しく復号されるアルゴリズム.

定義 2.10 (CPA 安全な鍵カプセル化アルゴリズム)
鍵カプセル化アルゴリズムが *CPA* 安全であるとは以下を満たすことである. 任意の確率的多項式時間アルゴリズム $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ に対して, $|\Pr[(pk, sk) \leftarrow \mathrm{Gen}(1^\kappa); s \leftarrow \mathcal{A}_1(pk); K_0^* \leftarrow \mathcal{K}(k); (K_1^*, C^*) \leftarrow \mathrm{Enc}(pk); b \leftarrow \{0,1\}; K^* = K_b; b' \leftarrow \mathcal{A}_2(pk, C^*, K^*, s); b' = b] - \frac{1}{2}| \leq negl$

定義 2.11 (コミットメント)
コミットメントは以下の 3 つ組のアルゴリズム

- CGen: 1^κ を入力として, 公開鍵 pk を出力する.
- Com: 公開鍵 pk と平文 m を入力とし, コミットメント com とデコミットメント $dcom$ を出力する.
- CVer: 公開鍵 pk とコミットメント com デコミットメント $dcom$ と平文 m を入力として, 承認ならば 1, 否認ならば 0 を出力する.

からなるアルゴリズム.

定義 2.12 (安全なコミットメント)
コミットメントが安全なコミットメントであるとは以下の 3 つの性質を満たすことである;

[完全性] すべての $m \in \mathcal{M}$ に対し, $\Pr[b = 1 | pk \leftarrow \mathrm{CGen}(1^\kappa), (com, dcom) \leftarrow \mathrm{Com}(m), b := \mathrm{CVer}(pk, com, dcom, m)] = 1$

[秘匿性] 任意の確率的多項式時間アルゴリズム $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2)$ に対して, $|\Pr[(pk, sk) \leftarrow \mathrm{Gen}(1^\kappa); (m_0, m_1, s) \leftarrow \mathcal{D}_1(1^\kappa); b \leftarrow \{0,1\}; (com, dcom) \leftarrow \mathrm{Com}(pk, m_b); b' \leftarrow \mathcal{D}_2(com, s); b' = b] - \frac{1}{2}| \leq negl$

[拘束性] 任意の確率的多項式時間アルゴリズム $\mathcal{A}$ に対して, $|\Pr[(pk, sk) \leftarrow \mathrm{Gen}(1^\kappa); (com, dcom_1, dcom_2, m_1, m_2) \leftarrow \mathcal{A}(1^\kappa); b_1 \leftarrow \mathrm{Ver}(pk, com, dcom_1, m_1); b_2 \leftarrow \mathrm{Ver}(pk, com, dcom_2, m_2); b_1 = b_2 \wedge m_1 \neq m_2] - \frac{1}{2}| \leq negl$

2.2 DMKD の安全性の定義

$\mathcal{U} := \{U_1, \cdots, U_N\}$ をプロトコルの想定ユーザとする. ユーザ U_i を確率的多項式時間アルゴリズムとし, 長期秘密鍵 SSK_i, 長期公開鍵 SPK_i を生成するものし, PKI システムによりユーザは正しく保障されているものとする. ユーザ U_i と認証サーバ S は認証されていないスタートポロジ型の通信でつながっている. また, ユーザ間で直接通信はしない. サーバ S も確率的多項式時間アルゴリズムとし, 長期秘密鍵 SSK_S, 長期公開鍵 SPK_S を生成する.

2.2.1 セッションと state 情報

DMKD プロトコルには (Dist, Join, Leave) の 3 つのフェーズがあり, これの発動をセッションと呼ぶ. 以降, セッションには $\{U_{i_1}, \cdots, U_{i_n}\}$ の $n(2 \geq n \geq N)$ 人含まれるものとする. Π を $\Pi \in \{\text{Dist, Join, Leave}\}$ で与える. ユーザインスタンス $U_{i_l}^{j_l}$ の所有するセッションは $(\Pi, U_{i_l}^{j_l}, \{U_{i_1}^{j_1}, \cdots, U_{i_n}^{j_n})$ によって管理される (ただし, $U_{i_l}^{j_l}$ は U_{i_l} の j_l 番目のインスタンス). ユーザインスタンス $\{U_{i_1}^{j_1}, \cdots, U_{i_{l-1}}^{i_{l-1}}, U_{i_{l+1}}^{i_{l+1}}, \cdots, U_{i_n}^{j_n}\}$ によって張られるセッションを $U_{i_l}^{j^l}$ のマッチングセッションと呼ぶ. 以降, U_{i_l} を U_l と表記する. また, セッションの総数を l_{max} とする. ここで, time frame の概念を取り入れる. 各ユーザとサーバはセッションが time frame 内で開始されるとき, ある state 情報を用いて, 通信をする. 一方, 時間ごとのセッション鍵更新を考慮し, Update は新しい time frame 時のセッション鍵の更新を担う. Dist では, U_i^j は短期秘密鍵 EPK_i^j を生成し, 短期公開鍵 ESK_i^j をサーバに送る. サーバはすべてのユーザから EPK_i^j を受け取り, 各ユーザにメッセージを返答する. ユーザとサーバは round を繰り返し, 最終的にセッション鍵を共有する. また, セッション鍵共有後には, Update および, Join, Leave に必要な state 情報を更新する. Update, Join, Leave においても, セッション鍵共有後に state 情報の更新を行う.

2.2.2 攻撃者

攻撃者 $\mathcal{A}$ を確率的多項式時間アルゴリズムとし, 次のクエリが使用できるものとする.

- Establish(U_i, SPK_i): このクエリにより新たなユーザ (A) を加えることができる. もし $U_i \notin \mathcal{U}$ ならば, U_i に長期公開鍵 SPK_i を付与するものとする. ここで, $\mathcal{A}$ は長期秘密鍵 SSK_i を持っている必要はない. Establish によって生成されたユーザを *dishonest*, それ以外のユーザを *honest* と呼ぶ.
- Send$(U_i^j, message)$: このクエリにより $\mathcal{A}$ はインスタンス U_i^j へ message を送ることができる. message には, $\Pi \in \{\text{Dist, Join, Leave}\}$ を含む.

さらに漏洩している情報を捉えるために, 攻撃者 $\mathcal{A}$ は次のクエリが許される.

- SessionReveal(U_i^j): 攻撃者 $\mathcal{A}$ は完了した U_i^j のセッションに対して, セッション鍵 SK を得る.
- StateReveal(U_i): 攻撃者 $\mathcal{A}$ は U_i の state 情報 $state_i$ を得る.
- ServerReveal: 攻撃者 $\mathcal{A}$ はサーバ S の長期秘密鍵 SSK_S を得る.
- StaticReveal(U_i): 攻撃者 $\mathcal{A}$ は U_i の長期秘密鍵 SSK_i を得る.
- EphemeralReveal(U_i^j): 攻撃者 $\mathcal{A}$ はセッションが終了する前に U_i^j の短期秘密鍵 ESK_i^j を得る.

2.2.3 Freshness

安全性の定義のため, Freshness の概念を導入する.

定義 2.13
$\text{sid}^* = (\Pi, U_i^j, \{U_{i_1}^{j_1}, \cdots, U_{i_n}^{j_n}\})$ を *honest* なユーザ間で完了した U_i^j のセッションとする. $\overline{\text{sid}^*}$ を sid*のマッチングセッションとする. セッション sid が *fresh* であるとは, 以下の条件のいずれも成り立たないことを言う.

- 攻撃者 $\mathcal{A}$ は現在の time frame において, Session Reveal(U_i^j) または, ある $\overline{\text{sid}^*}$ に対する SessionReveal $(U_i^{j'})$ を行う.
- 攻撃者 $\mathcal{A}$ は ServerReveal または StaticReveal(U_i^j) またはある $\overline{\text{sid}^*}$ に対する StaticReveal(U_i^j) を行っていたとき, 過去の time frame において, Session Reveal(U_i^j) または, あるの $\overline{\text{sid}^*}$ に対する Session Reveal$(U_i^{j'})$ を行う.
- 攻撃者 $\mathcal{A}$ は sid^* が完了する前に ServerReveal を行う.
- 攻撃者 $\mathcal{A}$ は現在の $state_i$ に対しての StateReveal(U_i) または, $state_i$ に紐付いた過去の $state_{i'}$ に対しての Session Reveal$(U_{i'})$ を行う.
- 攻撃者 $\mathcal{A}$ は sid もしくはある $\overline{\text{sid}^*}$ が完了する前に, StaticReveal(U_i) を行う.
- 攻撃者 $\mathcal{A}$ は StaticReveal(U_i^j) と EphemeralReveal (U_i^j) を行う.
- 攻撃者 $\mathcal{A}$ は StaticReveal(U_i^j) とある $\overline{\text{sid}^*}$ に対する EphemeralReveal$(U_i^{j'})$ を行う.

2.2.4 安全性の検証

安全性の定義のため, 以下の通りに安全性の検証を行う. 攻撃者 $\mathcal{A}$ には honest ユーザと, 先で定義したクエリを与える. 検証の間, 攻撃者 $\mathcal{A}$ は以下のクエリを出す.

- Test(sid^*): sid^* は fresh でなければならない. $b \in \{0, 1\}$ をランダムに選び, $b = 0$ の場合は, sid^* によってセッション鍵を返し, $b = 1$ の場合は, ランダムな鍵を返す.

この検証は, 攻撃者 $\mathcal{A}$ が b を b' と推測するまで続く. 攻撃者 $\mathcal{A}$ が win する事象をテストセッション sid^* が fresh であって, かつ $b' = b$ であるものとする. 攻撃者 $\mathcal{A}$ の優位性を $\text{Adv}^{\text{dmkd}}(\mathcal{A}) = Pr[\mathcal{A}\ win] - \frac{1}{2}$ と定義する.

定義 2.14 (DMKD 安全)
DMKD プロトコル Π が DMKD モデルで安全であるとは, 以下の条件を満たすことである.

- 2 つの honest なユーザによるマッチングセッションが完了したとき, 2 つのセッション鍵が同一でない確率が negligible である.
- 任意の確率的多項式時間アルゴリズム $\mathcal{A}$ に対して, $\mathrm{Adv}^{\mathrm{dmkd}}(\mathcal{A})$ が negligible である.

3 提案手法

3.1 設計概要

提案手法では, Yoneyama らの DMKD プロトコルと同様, time frame ごとの鍵更新と, ユーザの出入りに伴う鍵更新が走り, それぞれにおいて鍵 K_1, K_2 を共有する. time frame ごとの鍵更新は属性ベース暗号を使用して鍵 K_1 を更新する. 一方, ユーザの出入りに伴う鍵更新では, (番号付けされた) ユーザの隣同士のユーザと二者間鍵交換を KEM で行い, 多人数での鍵交換をその二者間共有鍵を用いて行う. 最終的に, セッション共有鍵 SK を $SK = K_1 \oplus K_2$ の形で発行している (なお, 鍵 K_1 はサーバに秘匿されないため, 鍵 K_2 をサーバに秘匿する設計となる). また, メンバの出入りによって, 鍵の再配布を行わないようにするために state 情報を取り入れている. state 情報は主に二者間鍵交換時の情報を用いて生成され, 保管される. 以降では, 本稿でのアルゴリズムの概要を示す. DMKD プロトコルは 5 つのアルゴリズム DMKD=DMKD.{Setup, Dist, Join, Leave, Update} の使用アルゴリズムからなる.

3.2 DMKD.Setup

Setup アルゴリズムではサーバ, ユーザによる長期鍵発行などを行う (ユーザ, サーバ間の通信は行わない).

3.2.1 サーバ

サーバは以下の動作を行う. κ をセキュリティパラメタとする. $(Params, msk) \leftarrow \mathrm{Setup}(1^\kappa)$ より公開パラメタ $Params$ とマスタ秘密鍵 msk を発行する. 次に演算 $\oplus$ を添加した群 V と演算 $*$ を添加した群 G を置く. 次に標的衝突困難関数 $TCR : \{0,1\}^* \to \{0,1\}^\kappa$, ねじれ擬似ランダム関数 $tPRF : \{0,1\}^\kappa \times Kspace_\kappa \times : \{0,1\}^\kappa \times Kspace_\kappa \to V$, $tPRF' : \{0,1\}^\kappa \times Kspace_\kappa \times : \{0,1\}^\kappa \times Kspace_\kappa \to G$, 擬似ランダム関数 $F : \{0,1\}^\kappa \times G \to V$, $F' : \{0,1\}^\kappa \times V \to Kspace_\kappa$, $F'' : \{0,1\}^\kappa \times Kspace_\kappa \to \{0,1\}^\kappa$, $F : \{0,1\}^\kappa \times Kspace_\kappa \to G$ を用意し, さらに $st_S \xleftarrow{r} \{0,1\}^\kappa$, $st'_S \xleftarrow{r} Kspace_\kappa$ を生成する. 長期秘密鍵, 長期公開鍵をそれぞれ $SSK_S := (msk, st_S, st'_S), SPK_S := (Params, V, G, TCR, tPRF, tPRF', F, F', F'', F''')$ とする.

3.2.2 端末

端末 U_i は以下の動作を行う. $(pk_i, sk_i) \leftarrow \mathrm{PGen}(1^\kappa)$ より, 公開鍵, 秘密鍵を生成し, $st_i \xleftarrow{r} \{0,1\}^\kappa$, $st'_i \xleftarrow{r} Kspace_\kappa$ を生成する. $pk'_i \leftarrow \mathrm{CGen}(1^\kappa)$ よりコミットメント公開鍵を生成する. 長期秘密鍵, 長期公開鍵をそれぞれ $SSK_i := (sk_i, st_i, st'_i), SPK_i := (pk_i, pk'_i)$ とする.

3.3 DMKD.Dist

Dist アルゴリズムではユーザに鍵配布を行う.

3.3.1 サーバ (New Time Frame)

サーバは Time Frame が変わるたびに, 以下の動作を最初に行う. $usk_i \leftarrow \mathrm{Der}(Params, msk, A_i)$, ($A_i$ は端末と時間帯の認証情報), $mk_i \leftarrow \mathrm{MGen}(1^\kappa)$ を行い, 端末 U_i の端末秘密鍵 usk_i, 認証鍵 mk_i を生成する. $CT_i \leftarrow \mathrm{PEnc}(pk_i, (usk_i, mk_i))$ を行い, (msk_i, usk_i) を暗号化し, その暗号文 CT_i を端末 U_i に送る.

3.3.2 端末 (New Time Frame)

端末は Time Frame が変わるたびに, 以下の動作を最初に行う. サーバから CT_i を受け取り, $(usk_i, mk_i) \leftarrow \mathrm{PDec}(sk_i, CT_i)$ より復号する. 端末がメモリ上に保管する情報 $state_i = \{usk_i, mk_i, H_i^{(l)}, H_i^{(r)}, r\}$ のうち, usk_i, mk_i を更新する.

3.3.3 端末 (Round 1)

端末は Dist の開始に伴って最初の Round で以下の動作を行う. $\tilde{r}_i \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{r}'_i \xleftarrow{r} Kspace_\kappa$ を生成し, $r_i := tPRF(\tilde{r}_i, \tilde{r}'_i, st_i, st'_i)$ を計算する. 次に, $(pk''_i, sk''_i) \leftarrow \mathrm{KGen}(1^\kappa; r_i)$ より鍵暗号化鍵, 鍵カプセル復号鍵をそれぞれ生成し, 鍵暗号化鍵 pk''_i をサーバ S に送る.

3.3.4 サーバ (Round 1)

サーバは端末 U_i から pk''_i を受け取り次第, これを U_{i-1} に配布する.

3.3.5 端末 (Round 2)

端末 U_i は Round 2 として, 以下の動作を行う. サーバから pk''_{i+1} を受け取る. 次に $\tilde{s}_i, \tilde{k}_i \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{s}'_i, \tilde{k}'_i \xleftarrow{r} Kspace_\kappa$ を生成し, $s_i := tPRF(\tilde{s}_i, \tilde{s}'_i, st_i, st'_i)$, $k_i := tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ を計算する. 次に $(R_{i,i+1}, C_{i,i+1}) \leftarrow \mathrm{KEnc}(pk''_{i+1}; s_i)$, $(com_i, dom_i) \leftarrow \mathrm{Com}(pk'_i, k_i)$ により, 二者間共有鍵 $R_{i,i+1}$, 鍵カプセル $C_{i,i+1}$, コミットメント com_i, デコミットメント $dcom_i$ を生成し, $(C_{i,i+1}, com_i)$ をサーバに送る. ここで, 端末の短期秘密鍵は $ESK_i := (\tilde{r}_i, \tilde{r}'_i, \tilde{s}_i, \tilde{s}'_i, \tilde{k}_i, \tilde{k}'_i)$ とする.

3.3.6 サーバ (Round 2)

端末は Round 2 として, 以下の動作を行う. 各端末から $(C_{i,i+1}, com_i)$ を受け取る. セッション ID として $sid := TCR(com_1, \cdots, com_n)$ を計算し, 代表端末 (ここでは, U_1 とする) を選ぶ. $(sid, C_{i-1,i})$ を端末 U_i に送り, 端末 U_1 にはさらに代表ユーザであることを伝える.

3.3.7 端末 (Round 3)

端末 $U_i(U_1$ は除く) は Round 3 として, 以下の動作を行う. $R_{i-1,i} \leftarrow$ KEM.Dec$((pk_i'', sk_i''), C_{i-1,i})$ により復号を行い, $K_i^{(l)} = F(sid, R_{i-1,i})$, $K_i^{(r)} = F(sid, R_{i,i+1})$, $T_i := K_i^{(l)} \oplus K_i^{(r)}$ を計算する. さらに, $\sigma_i \xleftarrow{r}$ MAC.Tag($mk_i, (U_i, C_{i,i+1}, com_i, sid, C_{i,i-1}, k_i, dcom_i, T_i)$) により認証子を発行し, $(k_i, dcom_i, T_i, \sigma_i)$ をサーバに送る. 端末 U_1 は Round 3 として, 以下の動作を行う. $R_{n,1} \leftarrow$ KEM.Dec$((pk_1'', sk_1''), C_{n,1})$ により復号を行い, $K_1^{(l)} = F(sid, R_{n,1})$, $K_1^{(r)} = F(sid, R_{1,2})$, $T_1 := K_1^{(l)} \oplus K_1^{(r)}$, $T' := K_1^{(l)} \oplus (k_1 || dcom_1)$ を計算する. さらに, $\sigma_1 \xleftarrow{r}$ MAC.Tag$(mk_1, (U_1, C_{1,2}, com_1, sid, C_{n,1}, T_1, T'))$ により認証子を発行し, (T_1, T', σ_1) をサーバに送る.

3.3.8 サーバ (Round 3)

サーバは Round 3 として, 以下の動作を行う. $k_i, dcom_i, T_i, \sigma_i (i \neq 1)$ と T_1, T', σ_1 を受け取り, MAC.Ver $(mk_i, \sigma_i, (U_i, C_{i,i+1}, com_i, sid, C_{i,i-1}, k_i, dcom_i, T_i))(i \neq 1)$, MAC.Ver$(mk_1, \sigma_1, (U_1, C_{1,2}, com_1, sid, C_{n,1}, T_1, T'))$, COM.Ver$(pk_i', k_i, com_i, dcom_i)(i \neq 1)$ によって認証を行う. 次に $\tilde{k_S}, \tilde{K_1} \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{k_S}', \tilde{K_1}' \xleftarrow{r} Kspace_\kappa$ を生成し, $k_S := tPRF(\tilde{k_S}, \tilde{k_S}', st_S, st_S')$, $K_1 := tPRF(\tilde{K_1}, \tilde{K_1}', st_S, st_S')$ を計算する. 次に, $k' = \bigoplus_{i=2}^{n} k_i \oplus k_S$, $T_i' = \bigoplus_{j=1}^{i-1} T_j$ を計算し, $CT_i' \xleftarrow{r}$ ABE.Enc$(Params, P_i, K_1)$ (P_i は U_i が TimeFrame 内に復号を可能にする構造) により K_1 を暗号化する. さらに, $\sigma_i' \xleftarrow{r}$ MAC.Tag$(mk_i, (U_i, C_{i,i+1}, com_i, sid, C_{i,i-1}, k_i, dcom_i, T_i, com_1, k', T_i', CT_i'))$, $\sigma_1' \xleftarrow{r}$ MAC.Tag$(mk_1, (U_1, C_{1,2}, com_1, sid, C_{n,1}, T_1, T', k', CT_1'))$ により認証子を発行し, (k', CT_1', σ_1') を U_1 に, $(com_1, k', T_i', CT_i', \sigma_i')$ を U_i に送る.

3.3.9 端末 (セッション共有鍵の生成)

端末 $U_i(U_1$ は除く) はセッション共有鍵生成のため, 以下の動作を行う. $(com_1, k', T_i', CT_i', \sigma_i')$ を受け取り, MAC.Ver $(mk_i, (U_i, C_{i,i+1}, com_i, sid, C_{i,i-1}, k_i, dcom_i, T_i, com_1, k', T_i', CT_i'))$ によって認証を行う. 次に $K_1^{(l)} = T_i' \oplus K_i^{(l)}$, $k_1 || dcom_1 = T' \oplus K_1^{(l)}$ を計算し, COM.Ver$(pk_1', k_1, com_1, dcom_1)$ によって認証を行う. 次に, $K_1 \leftarrow$ ABE.Dec(usk_i, CT_i, P_i), $K_2 = F'(sid, k' \oplus k_1)$ を生成し, セッション共有鍵 $SK = F''(sid, K_1) \oplus F"(sid, K_2)$ を発行する. 端末がメモリ上に保管する情報 $state_i = \{usk_i, mk_i, H_i^{(l)}, H_i^{(r)}, r\}$ のうち, $H_i^{(l)}, = R_{i-1,i}, H_i^{(r)} = R_{i,i+1}, r = F'''(sid, K_1) * F'''(sid, K_2)$ を更新する. 端末 U_1 はセッション共有鍵生成のため, 以下の動作を行う. (k', CT_1', σ_1') を受け取り, MAC.Ver $(mk_1, (U_1, C_{1,2}, com_1, sid, C_{n,1}, T_1, T', k', CT_1'))$ によって認証を行う. 次に, $K_1 \leftarrow$ ABE.Dec(usk_1, CT_1, P_1), $K_2 = F'(sid, k' \oplus k_1)$ を生成し, セッション共有鍵 $SK = F''(sid, K_1) \oplus F"(sid, K_2)$ を発行する. 端末がメモリ上に保管する情報 $state_1 = \{usk_1, mk_1, H_1^{(l)}, H_1^{(r)}, r\}$ のうち, $H_1^{(l)}, = R_{n,1}, H_1^{(r)} = R_{1,2}, r = F'''(sid, K_1) * F'''(sid, K_2)$ を更新する.

3.4 DMKD.Join

Join アルゴリズムではユーザの追加に伴う鍵更新を行う. もとのセッション共有鍵を利用して, 追加ユーザを含めた更新を行うため, 追加ユーザの鍵共有が走る. 本稿では Yoneyama らのプロトコル [10] との差分を記す.

3.4.1 端末 (Round 1)

端末 U_1, U_{n+1}(追加端末) は Join の開始に伴って最初の Round で以下の動作を行う. $\tilde{r_i} \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{r_i'} \xleftarrow{r} Kspace_\kappa$ を生成し, $r_i := tPRF(\tilde{r_i}, \tilde{r_i'}, st_i, st_i')$ を計算する. 次に, $(pk_i'', sk_i'') \leftarrow$ KGen$(1^\kappa; r_i)$ より鍵暗号化鍵, 鍵カプセル復号鍵をそれぞれ生成し, 鍵暗号化鍵 pk_i'' をサーバ S に送る. また, $U_i(U_1, U_{n+1}$ は除く) は, このラウンドは省略される.

3.4.2 サーバ (Round 1)

サーバは端末 U_i から pk_i'' を受け取り次第, これを U_{i-1} に配布する.

3.4.3 端末 (Round 2)

端末 U_n, U_{n+1} は Round 2 として, 以下の動作を行う. サーバから pk_{i+1}'' を受け取る. 次に $\tilde{s_i}, \tilde{k_i} \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{s_i'}, \tilde{k_i'} \xleftarrow{r} Kspace_\kappa$ を生成し, $s_i := tPRF(\tilde{s_i}, \tilde{s_i'}, st_i, st_i')$, $k_i := tPRF(\tilde{k_i}, \tilde{k_i'}, st_i, st_i')$ を計算する. 次に, $(R_{i,i+1}, C_{i,i+1}) \leftarrow$ KEnc$(pk_{i+1}''; s_i)$, $(com_i, dom_i) \leftarrow$ Com(pk_i', k_i) により, 二者間共有鍵 $R_{i,i+1}$, 鍵カプセル $C_{i,i+1}$, コミットメント com_i, デコミットメント $dcom_i$ を生成し, $(C_{i,i+1}, com_i)$ をサーバに送る. 端末 $U_i(U_n, U_{n+1}$ は除く) は Round 2 として, $\tilde{k_i} \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{k_i'} \xleftarrow{r} Kspace_\kappa$ を生成し, $k_i := tPRF(\tilde{k_i}, \tilde{k_i'}, st_i, st_i')$ を計算する. 次に, $(com_i, dom_i) \leftarrow$ Com(pk_i', k_i) で, コミットメント com_i, デコミットメント $dcom_i$ を生成し, com_i をサーバに送る.

3.4.4 サーバ (Round 2)

端末は Round 2 として, 以下の動作を行う. 各端末から $(C_{i,i+1}, com_i)$, or, (com_i) を受け取る. セッション ID として $sid := TCR(com_1, \cdots, com_n)$ を計算し, 代表端末 (ここでは, U_1 とする) を選ぶ. $(sid, C_{i-1,i})$ を端

末 $U_i(U_1, U_{n+1})$ に, sid を端末 $U_i(U_1, U_{n+1}$ を除く) に送り, 端末 U_1 にはさらに代表ユーザであることを伝える.

3.5 DMKD.Leave

Leave アルゴリズムではユーザの削除に伴う鍵更新を行う. 二者間共有鍵を利用して, 削除ユーザに秘匿した鍵更新が走る. 本稿では Yoneyama らのプロトコル [10] との差分を記す. 以下, Leave user を U_j とする.

3.5.1 端末 (Round 1)

端末 U_{j+1} は Leave の開始に伴って最初の Round で以下の動作を行う. $\tilde{r}_{j+1} \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{r}'_{j+1} \xleftarrow{r} Kspace_\kappa$ を生成し, $r_{j+1} := tPRF(\tilde{r}_{j+1}, \tilde{r}'_{j+1}, st_{j+1}, st'_{j+1})$ を計算する. 次に, $(pk''_{j+1}, sk''_{j+1}) \leftarrow \mathrm{KGen}(1^\kappa; r_{j+1})$ より鍵暗号化鍵, 鍵カプセル復号鍵をそれぞれ生成し, 鍵暗号化鍵 pk''_{j+1} をサーバ S に送る. また, $U_i(U_j, U_{j+1}$ は除く) は, このラウンドは省略される.

3.5.2 サーバ (Round 1)

サーバは端末 U_{j+1} から pk''_{j+1} を受け取り次第, これを U_{j-1} に配布する.

3.5.3 端末 (Round 2)

端末 U_{j-1} は Round 2 として, 以下の動作を行う. サーバから pk''_{j+1} を受け取る. 次に $\tilde{s}_{j-1}, \tilde{k}_{j-1} \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{s}'_{j-1}, \tilde{k}'_{j-1} \xleftarrow{r} Kspace_\kappa$ を生成し, $s_{j-1} := tPRF(\tilde{s}_{j-1}, \tilde{s}'_{j-1}, st_{j-1}, st'_{j-1})$, $k_{j-1} := tPRF(\tilde{k}_{j-1}, \tilde{k}'_{j-1}, st_{j-1}, st'_{j-1})$ を計算する. 次に, $(R_{j-1,j+1}, C_{j-1,j+1}) \leftarrow \mathrm{KEnc}(pk''_{j+1}; s_{j-1})$, $(com_{j-1}, dom_{j-1}) \leftarrow \mathrm{Com}(pk'_{j-1}, k_{j-1})$ から, 二者間共有鍵 $R_{j-1,j+1}$, 鍵カプセル $C_{j-1,j+1}$, コミットメント com_{j-1}, デコミットメント $dcom_{j-1}$ を生成, $(C_{j-1,j+1}, com_{j-1})$ をサーバに送る. 端末 $U_i(U_j, U_{j-1}$ は除く) は Round 2 として, $\tilde{k}_i \xleftarrow{r} \{0,1\}^\kappa$, $\tilde{k}'_i \xleftarrow{r} Kspace_\kappa$ を生成し, $k_i := tPRF(\tilde{k}_i, \tilde{k}'_i, st_i, st'_i)$ を計算する. 次に, $(com_i, dom_i) \leftarrow \mathrm{Com}(pk'_i, k_i)$ で, コミットメント com_i, デコミットメント $dcom_i$ を生成し, com_i をサーバに送る.

3.5.4 サーバ (Round 2)

端末は Round 2 として, 以下の動作を行う. 各端末から $(C_{j-1,j+1}, com_i)$, $(com_i)(i \neq j-1, j)$ を受け取る. セッション ID として $sid := TCR(com_1, \cdots, com_n)$ を計算し, 代表端末 (ここでは, U_{j-1} とする) を選ぶ. $(sid, C_{j-1,j+1})$ を端末 U_{j+1} に, sid を端末 $U_i(U_j, U_{j+1}$ を除く) に送り, 端末 U_{j-1} にはさらに代表ユーザであることを伝える.

3.6 DMKD.Update

Update アルゴリズムでは time frame ごとの鍵更新を行う. ABE アルゴリズムを用いて共有された K_1 を更新する. 詳細は Yoneyama らのプロトコル [10] を参照.

4 安全性の考察

定理 4.1

TCR が衝突困難であり, $tPRF, tPRF'$ がねじれ擬似ランダム関数であり, F, F', F'', F''' が疑似ランダム関数であり, (PGen, PEnc, PDec) が CCA 安全な公開鍵暗号であり, (Setup, Der, AEnc, ADec) が selective-CCA 安全な CP-ABE であり, (MGen, Tag, MVer) が UF-CMA なメッセージ認証子であり, (KGen, KEnc, KDec) は CPA 安全な鍵カプセル化メカニズムであり, (CGen, Com, CVer) は安全なコミットメントであるならば, この DMKD 方式は DMKD モデルにおいて安全である.

以下, 上記定理の略証を与える.

4.1 DMKD.Dist

以下のハイブリッド実験 $H_0, \cdots, H_7$ を行い. 本プロトコルで攻撃者が正しくビットを推測する確率とランダムに推測する確率の差が negligible であることを示す.

- H_0: 実際の DMKD プロトコルでの実験. すなわち, $\mathrm{Adv}(\mathcal{A}, H_0)$ は実際の実験.
- H_1: 2 つの sid が同じであれば実験を停止する.
- H_2: ユーザインスタンス $\{U_{i_1}^{j_1}, \cdots U_{i_n}^{j_n}\}$ とそのオーナ $\{U_i^j\}$ を前もって選ぶ. 仮に $\mathcal{A}$ が U_i^j の所有する $\{U_{i_1}^{j_1}, \cdots U_{i_n}^{j_n}\}$ 以外のセッションをテストクエリで呼んだ場合は実験を停止する.
- H_3: $\mathcal{A}$ が平文を改ざんした場合は実験を停止する.
- H_4: (r_i, s_i, k_i, k_S) をランダムなものに変更する.
- H_5: $R_{i-1,i}, R_{i,i+1}$ をランダムなものに変更する.
- H_6: $K_i^{(l)}, K_i^{(r)}$ をランダムなものに変更する.
- H_7: K_2 をランダムなものに変更する.
- H_8: K'_2 をランダムなものに変更する.

これらの実験に関して以下の 1-9 が成立し, $\mathcal{A}$ がセッション共有鍵とランダム鍵の区別が困難かつ改ざんが困難である (それぞれの成功確率が negligible) ことから, Dist アルゴリズムの DMKD 安全性が示される.

1. $|\mathrm{Adv}(\mathcal{A}, H_0) - \mathrm{Adv}(\mathcal{A}, H_1)| \leq \epsilon_{\mathrm{TCR}}$
2. $\mathrm{Adv}(\mathcal{A}, H_2) \geq (1/l_{max})\mathrm{Adv}(\mathcal{A}, H_1)$
3. $|\mathrm{Adv}(\mathcal{A}, H_2) - \mathrm{Adv}(\mathcal{A}, H_3)| \leq \epsilon_{\mathrm{MAC,COM}}$
4. $|\mathrm{Adv}(\mathcal{A}, H_3) - \mathrm{Adv}(\mathcal{A}, H_4)| \leq \epsilon_{\mathrm{tPRF}}$
5. $|\mathrm{Adv}(\mathcal{A}, H_4) - \mathrm{Adv}(\mathcal{A}, H_5)| \leq \epsilon_{\mathrm{KEM}}$

6. $|\mathrm{Adv}(\mathcal{A}, H_5) - \mathrm{Adv}(\mathcal{A}, H_6)| \leq \epsilon_{\mathrm{PRF}}$

7. $|\mathrm{Adv}(\mathcal{A}, H_6) - \mathrm{Adv}(\mathcal{A}, H_7)| \leq \epsilon_{\mathrm{PRF}}$

8. $|\mathrm{Adv}(\mathcal{A}, H_7) - \mathrm{Adv}(\mathcal{A}, H_8)| \leq \epsilon_{\mathrm{PRF}}$

9. $\mathrm{Adv}(\mathcal{A}, H_8) = 1/2$

なお, 上記は以下の証明方針から示すことができる.

1. TCR の衝突困難性より, $\mathcal{A}$ は negligible(ϵ_{TCR}) で選択された sid を衝突させることができる.

2. $\mathcal{A}$ は l_{max} 個のセッションから一つ選択する.

3. MAC の UF-CMA, COM の秘匿性より, $\mathcal{A}$ は negligible ($\epsilon_{\mathrm{MAC,COM}}$) で平文を改ざんできる.

4. tPRF の疑似ランダム性より, $\mathcal{A}$ はランダムなものと negligible(ϵ_{COM}) で区別できる.

5. KEM の CPA 安全性より, $\mathcal{A}$ はランダムなものと negligible(ϵ_{KEM}) で区別できる.

6. PRF の疑似ランダム性より, $\mathcal{A}$ はランダムなものと negligible(ϵ_{PRF}) で区別できる.

7. PRF の疑似ランダム性より, $\mathcal{A}$ はランダムなものと negligible(ϵ_{PRF}) で区別できる.

8. PRF の疑似ランダム性より, $\mathcal{A}$ はランダムなものと negligible(ϵ_{PRF}) で区別できる.

9. 真にランダムなものとの置き換え.

4.2 DMKD.{Join, Leave, Update}

Yoneyama らのプロトコル [10] の (6. Security) を参照. なお, PKE の CCA 安全性は Update に寄与する.

5 まとめと今後の課題

本稿では KEM を用いてポスト量子暗号の適用が可能な DMKD プロトコルの考案を行い, 古典攻撃者に対する DMKD 安全性を確認した. 本稿では量子攻撃者を対象とした安全性証明において注意が必要なランダムオラクルモデル, 巻き戻し等の手法 [9] は用いていないため, 量子攻撃者への耐性が期待される (ただし, セキュリティパラメタには注意が必要). 今後は, Song[6] の手法を用いた量子攻撃者に対する安全性証明を行うとともに, 本稿アルゴリズムに適用するポスト量子暗号の検討を行う.

参考文献

[1] A. Ballardie.; Scalable Multicast Key Distribution, RFC Editor 1996.

[2] Chris Peikert.; Lattice Cryptography for the Internet, PQCrypto 2014.

[3] Edoardo Persichetti. Secure and Anonymous Hybrid Encryption from Coding Theory. In Philippe Gaborit, editor, Post-Quantum Cryptography: 5th International Workshop, PQCrypto 2013,

[4] Daniele Micciancio and Saurabh Panjwani. Optimal communication complexity of generic multicast key distribution, EUROCRYPT 2004.

[5] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, Jean-Jacques Quisquater.; Provably Authenticated Group Diffie-Hellman Key Exchange: CSS 2001.

[6] Fang, Song.; A Note on Quantum Security for Post-Quantum Cryptography, PQCrypto 2014.

[7] IBM, IBM Announces Advances to IBM Quantum Systems & Ecosystem, `https://www-03.ibm.com/press/us/en/pressrelease/53374.wss`, (参照 2017-12-08).

[8] Jeffrey Hoffstein , Jill Pipher , Joseph H. Silverman.; NTRU: A Ring-Based Public Key Cryptosystem. Part of the Lecture Notes in Computer Science book series (LNCS, volume 1423).

[9] Jeroen van de Graaf.; Towards a formal definition of security for quantum protocols, PhD thesis, Department d'informatique et de r.o., Universite de Montreal, 1998.

[10] Kazuki Yoneyama, Reo Yoshida, Yuto Kawahara, Tetsutaro Kobayashi, Hitoshi Fuji, Tomohide Yamamoto.: Multi-Cast Key Distribution: Scalable, Dynamic and Provably Secure Construction, ProvSec 2016.

[11] Lyubashevsky V., Peikert C., and Regev O.; On ideal lattices and learning with errors over rings. EUROCRYPT 2010.

[12] Koutarou Suzuki and Kazuki Yoneyama.; Exposure-resilient one-round tripartite key exchange without random oracles, ACNS 2013.

[13] M. Ajtai.; Generating hard instances of lattice problems, STOC '96 Proceedings of the twenty-eighth annual ACM symposium on Theory of computing Pages 99-108.

[14] NIST, Post-Quantum Cryptography, `https://csrc.nist.gov/projects/post-quantum-cryptography` (参照 2017-12-08).

[15] Paulo S. L. M. Barreto, Shay Gueron, Tim Gueneysu, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich.; CAKE: Code-based Algorithm for Key Encapsulation, Published 2017 in IACR Cryptology ePrint Archive.

[16] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto.; MDPC-McEliece : New McEliece Variants from Moderate Density Parity-Check Codes. In Cryptology ePrint Archive, Report 2012/409, 2012

[17] Robert J. McEliece.; A public-key cryptosystem based on algebraic coding theory, Jet Propulsion Laboratory DSN Progress Report 4244, 114116.

[18] PressThink.; The Snowden Effect: definition and examples, `http://pressthink.org/2013/07/the-snowden-effect-definition-and-examples/` (参照 2017-12-08).

[19] Shor, Peter W.; Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer; SIAM journal on computing 26.5 (1997):1484-1509.

[20] SSL247, WHAT IS RSA?, `https://www.ssl247.com/kb/ssl-certificates/generalinformation/what-is-rsa-dsa-ecc` (参照 2017-12-08).

[21] theguardian.; Edward Snowden: the whistleblower behind the NSA surveillance revelations, `https://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance` (参照 2017-12-08).