Vol.2022-DPS-190 No.24
Vol.2022-CSEC-96 No.24
2022/3/10

Forward-Secureなカメレオンハッシュ関数

松原 功樹[1,a)]　Tian Yangguang[1,b)]　宮地 充子[1,2,c)]

概要：カメレオンハッシュ関数 (CH) は秘密鍵と公開鍵のペアを持つハッシュ関数である．CH は秘密鍵を持たないユーザには衝突に強いハッシュ関数となるが，秘密鍵を持っているユーザはハッシュ値の衝突を発見することができる．CH は，Shamir らによるオンライン/オフライン署名や Ateniese らによるブロックチェーンの修正などを始めとする様々な暗号方式で利用されている．しかし，CH において一度秘密鍵が露呈した場合その耐衝突性は失われ，既存の CH ベースの手法の安全性は保証されない．そこで本研究では，新たに Forward-Secure な CH (FS-CH) を提案する．この手法では，秘密鍵を持つユーザは定期的に秘密鍵を更新することができる．これにより，現在の秘密鍵が露呈しても過去に計算されたハッシュ値に関する衝突は発見されず，秘密鍵漏洩によるリスクの軽減を実現した．

キーワード：カメレオンハッシュ関数，Forward-Security，ブロックチェーン

Forward-Secure Chameleon Hash Function

Abstract: Chameleon Hash Function (CH) is a hash function with a secret and public key pair. CH is collision resistant for users without a secret key, while users with a secret key can find collisions in hash values. CH has been used in various cryptographic schemes, including online/offline signatures by Shamir et al. and blockchain modification by Ateniese et al. However, once the secret key is exposed in CH, its collision resistance is lost, and the security of existing CH-based methods cannot be guaranteed. In this paper, we propose a new Forward-Secure CH (FS-CH). In this method, a user with a secret key can update the secret key periodically. In this way, even if the current secret key is exposed, collisions related to hash values calculated in the past will not be found, thereby reducing the risk of secret key leakage.

Keywords: Chameleon Hash Function, Forward-Security, blockchain

1. はじめに

1.1 研究背景

カメレオンハッシュ関数 (CH) [1] は秘密鍵と公開鍵のペアを持つハッシュ関数である．CH は秘密鍵を持たないユーザには衝突耐性を持つハッシュ関数となるが，秘密鍵を持っているユーザはハッシュ値の衝突を発見することができる．CH は様々な暗号方式で利用される．Shamir らによって 2001 年に提案されたオンライン/オフライン署名 [2] では，オフラインで行う署名生成に CH を用いることで，オンラインで行う処理を CH の衝突発見のみしオンラインの計算量を削減することができる．Ateniese らによって 2005 年に提案された Sanitizable 署名 [3] では，過去に署名が生成されたメッセージの指定箇所を秘密鍵の所有者が CH を用いて修正し，修正されたメッセージの新たな署名を生成することができる．Ateniese らによって 2017 年に提案された修正可能なブロックチェーン [4] では CH を用いて，ブロックチェーン上のメッセージを衝突が生じるような新たなメッセージに修正することができる．更にこの手法を拡張した手法として，2017 年に Camenisch らによる，任意のメッセージが修正可能かどうかを予め決定できる Chameleon Hash with Ephemeral Trapdoor (CHET) [5] や，2019 年に Derler らによる，アクセスポリシーを満たすユーザのみに修正を許す Policy-Based Chameleon Hash (PCH) [6] などが存在する．

[1] 大阪大学
Osaka University
[2] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology
[a)] matsubara@cy2sec.comm.eng.osaka-u.ac.jp
[b)] jack@cy2sec.comm.eng.osaka-u.ac.jp
[c)] miyaji@comm.eng.osaka-u.ac.jp

1.2 本研究の目的

CH において一度秘密鍵が露呈した場合その衝突耐性は失われるため，上記の全ての手法の安全性は損なわれる．署名方式においては署名が偽造されることとなり，修正可能なブロックチェーンにおいては悪意のある修正を許すこととなる．そこで本研究では，新たに Forwar-Security を保証した Forward-Secure な CH (FS-CH) を提案する．Forward-Security とは，ある時刻における秘密鍵が露呈した場合でも，過去の秘密鍵及びそれらに関する秘密のデータは露呈しないという安全性である．本提案では [7] における秘密鍵の更新方式及び双線形写像ベースの署名方式を用いて，初めて Forward-Security を保証した CH を非対称な双線形写像を用いて設計した．FS-CH では，現在の秘密鍵が露呈しても過去に計算されたハッシュ値に関する衝突は発見されず，悪意のある署名の偽造やブロックチェーンの修正を防ぐことができる．

1.3 本論文の構成

2 章では本研究に用いる数学的な定義とアルゴリズムの定義について記載する． 3 章では本提案のアルゴリズムの定義及び正当性，安全性の定義について記載する． 4 章では，設計したプロトコルとその正当性及び安全性の証明を記載する． 5 章にて本研究のまとめを記載する．

2. 準備

2.1 数学的準備

本節では，本研究に必要な数学的定義を記述する．

定義 1 双線形写像

$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\tau$ を全て素数位数 q の乗法群とする．写像 $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\tau$ が以下の性質満たす時，$\mathbf{e}$ を双線形写像と呼ぶ．

双線形性 $\forall g_1, g_2 \in \mathbb{G}_1, \forall h_1, h_2 \in \mathbb{G}_2$ に対して，

$$\mathbf{e}(g_1, g_1 \cdot h_2) = \mathbf{e}(g_1, h_1) \cdot \mathbf{e}(g_1, h_2)$$

$$\mathbf{e}(g_1 \cdot g_2, h_1) = \mathbf{e}(g_1, h_1) \cdot \mathbf{e}(g_2, h_1).$$

非退化性 $\mathbf{e}(g, h) \neq 1$ となる $g \in \mathbb{G}_1, h \in \mathbb{G}_2$ が存在する．

計算可能性 $\forall g \in \mathbb{G}_1, \forall h \in \mathbb{G}_2$ に対して，$\mathbf{e}(g, h)$ が多項式時間で計算できる．

ここで，$\mathbb{G}_1 = \mathbb{G}_2$ のとき $\mathbf{e}$ を対称な双線形写像，そうでない時， 非対称な双線形写像と呼ぶ．

定義 2 l-wBDHI 問題

$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\tau$ を全て素数位数 q の乗法群 ($\mathbb{G}_1 \neq \mathbb{G}_2$)，$g, h$ をそれぞれ $\mathbb{G}_1, \mathbb{G}_2$ の生成元，写像 $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\tau$ を非対称な双線形写像とする．入力

$$A_1 = g^\alpha, A_2 = g^{\alpha^2}, ..., A_l = g^{\alpha^l}$$

$$B_1 = h^\alpha, B_2 = h^{\alpha^2}, ..., B_l = h^{\alpha^l}$$

$$C_1 = g^\gamma, C_2 = h^\gamma$$

$$(\alpha, \gamma \in \mathbb{Z}_q)$$

が与えられた時,

$$\mathbf{e}(g, h)^{\gamma \cdot \alpha^{l+1}}$$

を計算する問題を，l-wBDHI (l-Weak Bilinear Diffie-Hellman inversion) 問題という．

また，ここで l-wBDHI 問題を解く敵対者 $\mathcal{A}$ を仮定した時，$\mathcal{A}$ の計算成功確率を $\mathbf{Adv}^{l-\mathrm{wBDHI}}_{\mathbb{G}_1 \times \mathbb{G}_2}(\mathcal{A})$ とする．

2.2 本研究で用いられる方式

本節では，本研究に用いられる方式としてカメレオンハッシュ関数 (CH) のアルゴリズムを定義する．

カメレオンハッシュ関数 (CH) [1] は秘密鍵 sk と公開鍵 pk のペアを持つハッシュ関数である．ハッシュの生成者は，メッセージ m に加えてチェックストリング r を生成し，それらを入力としてハッシュ値 H を計算する．秘密鍵が知られていない場合，CH は衝突耐性を持つハッシュ関数としてふるまう．一方秘密鍵を持つユーザは，元の (m, r) に関して新たなメッセージ m' に対するチェックストリング r' を計算し，ハッシュ値 H の衝突を発見することができる．

定義 3 カメレオンハッシュ関数 (CH)

CH は以下の 5 つのアルゴリズム (Setup,KeyGen,Hash,Verify,Adapt) から成る．

Setup(1^λ):
: セキュリティパラメータ λ を入力とし，公開パラメータ PP を出力する．PP は以下の全てのアルゴリズムで利用され，メッセージ空間 $\mathbf{M}$ を含む．

KeyGen:
: 公開パラメータ PP を入力とし，秘密鍵と公開鍵の鍵ペア (sk, pk) を出力する．

Hash(pk, m):
: 公開鍵 pk 及びメッセージ $m \in \mathbf{M}$ を入力とする．チェックストリング r をランダムに生成し，ハッシュ値 H を計算する．

Adapt(sk, m, m', r, H):
: 秘密鍵 sk，元のメッセージ m と新たなメッセージ m'，チェックストリング r，及びハッシュ値 H を入力とし，新たなチェックストリング r' を計算する．

Verify(pk, m', r', H):
: 公開鍵 pk，メッセージ m'，チェックストリング r'，及びハッシュ値 h を入力とする．ハッシュ値が有効かどうかの判定 $b \in \{0, 1\}$ を出力する．

3. 提案手法

3.1 アルゴリズムの定義

本節では，本研究で提案する Forward-Secure なカメレオンハッシュ関数 (FS-CH) のアルゴリズムを定義する．

FS-CH では，ある時刻 t における秘密鍵 sk_t が知られていない場合，FS-CH は暗号学的ハッシュ関数としてふるまう．すなわち，秘密鍵 sk_t をもたないユーザは時刻 t において計算されたハッシュ値 H_t の衝突を発見することができない．一方秘密鍵を持つユーザは，元の (m, r) に関して新たなメッセージ m' に対するチェックストリング r' を計算し，ハッシュ値 H_t の衝突を発見することができる．

定義 4 Forward-Secure なカメレオンハッシュ関数 (FS-CH)

FS-CH は以下の 5 つのアルゴリズム (Setup,KeyGen,KeyUpdate,Hash,Verify,Adapt) から成る．

Setup$(T, 1^\lambda)$:
: 時刻の最大値 $T \in \mathbb{Z}$ 及びセキュリティパラメータ λ を入力とし，公開パラメータ PP を出力する．PP は以下の全てのアルゴリズムで利用され，メッセージ空間 $\mathbf{M}$ を含む．

KeyGen:
: 公開パラメータ PP を入力とし，時刻 t における秘密鍵と公開鍵の鍵ペア (sk_t, pk) を出力する．

KeyUpdate(sk_t):
: 時刻 t における秘密鍵 sk_t を入力として，その次の時刻 t' における秘密鍵 $sk_{t'}$ を出力する．

Hash(pk, m, t):
: 公開鍵 pk，メッセージ $m \in \mathbf{M}$，及び時刻 t を入力とする．チェックストリング r をランダムに生成し，時刻 t におけるハッシュ値 H_t を計算する．

Adapt(sk_t, m, m', r, H_t):
: 時刻 t における秘密鍵 sk_t，元のメッセージ m と新たなメッセージ m'，チェックストリング r，及び時刻 t におけるハッシュ値 H_t を入力とし，新たなチェックストリング r' を計算する．

Verify(pk, m', r', H_t, t):
: 公開鍵 pk，メッセージ m'，チェックストリング r'，時刻 t におけるハッシュ値 H_t，及び時刻 t を入力とする．時刻 t におけるハッシュ値が有効かどうかの判定 $b \in \{0, 1\}$ を出力する．

3.2 正当性，安全性の定義

本節では，はじめに FS-CH の正当性に関する定義を示す．

定義 5 FS-CH の正当性

Verify アルゴリズムにおいて入力 (pk, m', r', H_t, t) が正しいとき Verify$(pk, m', r', H_t, t) = 1$ となるならば，FS-CH は正当性を満たす．

次に FS-CH の安全性要件である Forward-Secure 衝突耐性について，はじめにその攻撃モデルを定義し，続いて満たされる安全性の定理を示す．

FS-CH が Forward-Secure 衝突耐性を満たす場合，現在の時刻における秘密鍵が露呈しても，それ以前の時刻に計算されたハッシュ値に関する衝突は発見されない．

定義 6 Forward-Secure 衝突耐性

Forward-Secure 衝突耐性は，以下の 3 つのフェーズから成るゲームによって定義される．

(1) Setup フェーズ

敵対者を $\mathcal{A}$，シミュレータを $\mathcal{S}$ とする．$\mathcal{S}$ は時刻の最大値 T 及び公開鍵 pk を $\mathcal{A}$ に与える．

(2) Training フェーズ

$\mathcal{A}$ は，以下に示す 4 つのオラクル (KeyUpdate,Hash,Adapt,Break in) に対して問い合わせを行うことができる．$\mathcal{S}$ は問い合わせに対して各オラクルを実行する．

KeyUpdate オラクル:
: 現在の時刻 t が T より小さい場合，秘密鍵 sk_t を $sk_{t'}$ に更新し，t の値を増加させる．

Hash オラクル:
: $\mathcal{A}$ はメッセージ m，公開鍵 pk，及び時刻 t を入力とする．$\mathcal{S}$ はチェックストリング r と時刻 t におけるハッシュ値 H_t を $\mathcal{A}$ に返す．

Adapt オラクル:
: $\mathcal{A}$ は元のメッセージ m と新たなメッセージ m'，チェックストリング r，時刻 t，時刻 t におけるハッシュ値 H_t を入力とする．$\mathcal{S}$ は新たなチェックストリング r' を $\mathcal{A}$ に返す．

Break in オラクル:
: break-in 時間として $\tilde{t} = t$ を記録し、現在の秘密鍵 $sk_{\tilde{t}}$ を $\mathcal{A}$ に返す．このオラクルはゲームの中で一度だけ問い合わせることができ，問い合わせの後，$\mathcal{A}$ は KeyUpdate オラクルや Adapt オラクルに対して問い合わせを行うことが不可能となる．

(3) Challenging フェーズ

$\mathcal{A}$ はメッセージ m^* と新たなメッセージ $m^{*\prime}$，元のチェックストリング r^* と新たなチェックストリング $r^{*\prime}$，及び時刻 t^* と時刻 t^* におけるハッシュ値 H_{t^*} を出力する．異なる 2 つのメッセージ $m^*, m^{*\prime}$ のハッシュ値 H_{t^*} が過去の時刻 $t^*(< \tilde{t})$ において正しく生成できており，元の m^* が時刻 t^* において Hash オラクルの入力となっている，かつ新たな $m^{*\prime}$ が時刻 t^* において Adapt オラクルの入力となっていない場合，$\mathcal{A}$ の勝利とする．

ここで，敵対者 $\mathcal{A}$ が上記の Forward-Secure 衝突耐性のゲー

ムに勝利する確率を，$\mathbf{Adv}_{FS}^{colres}(\mathcal{A})$ とする．

定義 7　選択的 Forward-Secure 衝突耐性

定義 6 で定義したゲームの Setup フェーズにおいて，敵対者 $\mathcal{A}$ がシミュレータ $\mathcal{S}$ から時刻の最大値 T 及び公開鍵 pk を与えられる前に，Challenging フェーズで出力する時刻 t^* を予め決定しておくようなゲームによって定義される安全性を，選択的 Forward-Secure 衝突耐性とする．

また敵対者 $\mathcal{A}$ が上記の選択的 Forward-Secure 衝突耐性のゲームに勝利する確率を，$\mathbf{Adv}_{FS}^{s-colres}(\mathcal{A})$ とする．

定理 1　時刻の最大値 T に対して Forward-Secure 衝突耐性を攻撃する敵対者 $\mathcal{A}$ 及び l-wBDHI 問題を解く敵対者 $\mathcal{B}$ を仮定した時，以下が成り立つ．

$$\frac{1}{T}\cdot\mathbf{Adv}_{FS}^{colres}(\mathcal{A})\leq\mathbf{Adv}_{\mathbb{G}_1\times\mathbb{G}_2}^{l-\mathrm{wBDHI}}(\mathcal{B})$$

すなわち FS-CH は，l-wBDHI 問題を解くことが計算量的に困難という仮定の下で安全となる．

4. 設計

4.1 二分木を用いた時刻の更新

本節では，[7] の手法を基にした，二分木を用いた時刻の更新方法を示す．図 1 のように，根ノード以外の各ノードには時刻が割り当てられている．また時刻は $t=t_0\parallel t_1\parallel ...\parallel t_l\in\{0,1,2\}^l$ とし，秘密鍵を保持しておくための時刻の集合を Γ_t とする．時刻の更新は以下のように行われる．二分木は T 個のノードを持ち，初めは時刻が根ノードにあるとする．

- 現在の時刻 t が葉ノード以外にある場合，現在の時刻を Γ_t から削除し，その後現在のノードからみて左の子にあたるノードを次の時刻 t' とする．また，そのノードと右の子に割り当てられた時刻を Γ_t に格納する．
- 現在の時刻 t が葉ノードにあり左の子となっている場合，現在の時刻を Γ_t から削除し，現在のノードからみて右の兄弟にあたるノードを次の時刻 t' とする．
- 現在の時刻 t が葉ノードにあり右の子となっている場合，現在の時刻を Γ_t から削除し，現在のノードより深さが 1 だけ小さく，親ノードの右の兄弟にあたるノードを次の時刻 t' とする．

また，秘密鍵 sk_t は Γ_t に含まれる時刻 $w=w_0\parallel w_1\parallel ...\parallel w_l\in\{0,1,2\}^l$ に対して，

$$sk_t=\{\tilde{sk_w}:\forall w\in\Gamma_t,\ f(t)\}$$

で与えられる．$\tilde{sk_w}$ 及び $f(t)$ の構成については 4.2 節に記載する．

図 1 は $l=4, T=2^4-1=15, t=112$ の場合である．$\Gamma_t=200,120,112$ であり，$100,110,111$ に関する秘密鍵は既に Γ_t から削除されている．また，更新が行われた際の次の時刻 t' は，上の 3 つめの場合より $t'=200$ である．

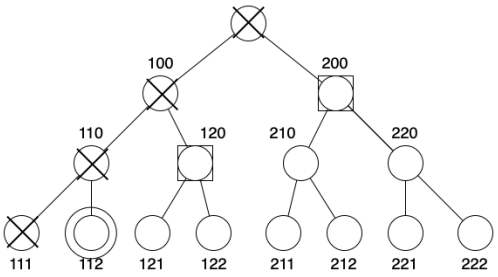

図 1　$t=112$ の時の二分木の様子 $(T=15)$

4.2 プロトコル

本節では，定義 4 で定義した FS-CH について，設計したプロトコルを示す．このプロトコルは，離散対数ベースの CH [1], [8] を基とし [7] で用いられる鍵の更新方式を導入したものである．Hash アルゴリズム及び Verify アルゴリズムには非対称な双線形写像が用いられる．Hash，Verify アルゴリズムは全ユーザが実行できるが，KeyUpdate，Adapt アルゴリズムは秘密鍵の所有者のみが実行できることに注意されたい．

Setup$(T,1^\lambda)$:

$\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_\tau$ を全て素数位数 q の乗法群 $(\mathbb{G}_1\neq\mathbb{G}_2)$，$g,h$ をそれぞれ $\mathbb{G}_1,\mathbb{G}_2$ の生成元，写像 $\mathbf{e}:\mathbb{G}_1\times\mathbb{G}_2\to\mathbb{G}_\tau$ を非対称な双線形写像とする．

最大時刻の値 $T=2^l-1$ $(l\in\mathbb{Z})$ に対し，$z_0,z_1,...,z_l\in\mathbb{Z}_q$ をランダムに選択したのち，$h_0=h^{z_0},h_1=h^{z_1},...,h_l=h^{z_l}$ を計算する．なお，$z_0,z_1,...,z_l$ は秘密鍵の所有者のみが知る値である．また，$t_0,t_1,...,t_l\in\{0,1,2\}$ について時刻 t を $t=t_0\parallel t_1\parallel ...\parallel t_l$ とし，初期値を $t_0,t_1,...,t_l=0$ と設定する．

$\mathbb{Z}_q$ をメッセージ空間とし，以下の 3 つの関数 $F,f,FSCH$ を定義する．

$$F:\{0,1,2\}^l\to\mathbb{G}_2$$
$$t\mapsto\prod_{i=0}^{k}h_i^{t_i}$$

$$f:\{0,1,2\}^l\to\mathbb{Z}_q$$
$$t\mapsto\sum_{i=0}^{k}z_it_i\ \ (mod\ q)$$

なお $f(t)$ は秘密鍵の所有者のみが知る値であり，$F(t)=h^{f(t)}$ を満たす．

$$FSCH:\mathbb{G}_1\times\mathbb{Z}_q\times\mathbb{Z}_q\times\{0,1,2\}^l\to\mathbb{G}_\tau$$
$$pk\times m\times r\times t\mapsto\mathbf{e}(pk\cdot g^m,h)\cdot\mathbf{e}(g^r,F(t))$$

$FSCH$ は Forward-Secure なカメレオンハッシュ関数 (FS-CH) である．

KeyGen:

$x \in \mathbb{Z}_q$ をランダムに選択して長期秘密鍵とし，$s \in \mathbb{Z}_q$ をランダムに選択する．また，$t_0 = 1$ として時刻を $t = 1$ と設定する．時刻 $w = w_0 \parallel w_1 \parallel ... \parallel w_l$, $t = t_0 \parallel t_1 \parallel ... \parallel t_l \in \{0,1,2\}^l$ 及び時刻に割り当てられた二分木の深さ k に対して

$$\begin{aligned} s\tilde{k}_w &= (c, d, e_{k+1}, ...e_l) \\ &= (g^s, h^x \cdot F(t)^s, {h_{k+1}}^s, ...{h_l}^s) \\ &= (g^s, h^x \cdot (\prod_{i=0}^{k} h_i^{w_i})^s, {h_{k+1}}^s, ...{h_l}^s) \\ f(t) &= \sum_{i=0}^{k} z_i t_i \ \ (mod\ q) \end{aligned}$$

とする．すなわちこの時刻 $t = 1$ における秘密鍵 sk_t は

$$\begin{aligned} sk_t &= \{s\tilde{k}_w : \forall w \in \Gamma_t = (1,2),\ f(1)\} \\ &= \{s\tilde{k}_1,\ z_0 t_0\} \\ &= \{(g^s, h^x \cdot {h_0}^s, {h_1}^s, ...{h_l}^s), \\ &\quad (g^s, h^x \cdot {h_0}^{2s}, {h_1}^s, ...{h_l}^s),\ z_0\} \end{aligned}$$

となる．また，公開鍵 $pk = g^x$ を計算し，これを公開する．

KeyUpdate(sk_t)**:**

時刻 t 及び集合 Γ_t を更新し $t', \Gamma_{t'}$ とする．時刻 $t \in \{0,1,2\}^l$ における秘密鍵 sk_t を入力として，その次の時刻 t' における秘密鍵 $sk_{t'} = \{s\tilde{k}_{w'} : \forall w' \in \Gamma_{t'},\ f(t')\}$ を更新する．$s\tilde{k}_{w'} = (c', d', e'_{k+2}, ...e'_l)$ について，はじめに $s' \in \mathbb{Z}_q$ をランダムに選択する．その後 $c', d', e'_{k+2}, ..., e'_l$ を以下のように計算する．

$$\begin{aligned} c' &= c \cdot g^{s'} \\ d' &= d \cdot {e_{k+1}}^{w_{k+1}} \cdot {h_{k+1}}^{w_{k+1} s'} \\ e'_{k+2}, .., e'_l &= e_{k+2} \cdot {h_{k+2}}^{s'}, ..., e_l \cdot {h_l}^{s'} \end{aligned}$$

すなわち，

$$s\tilde{k}_{w'} = (g^{s'}, h^x \cdot (\prod_{i=0}^{k+1} h_i^{w_i})^{s'}, {h_{k+2}}^{s'}, ...{h_l}^{s'})$$

となる．

また $f(t')$ については，時刻 t' における新たな木の深さ k' が元の k と異なる場合，以下のように計算する．

$$f(t) = \sum_{i=0}^{k'} z_i t_i \ \ (mod\ q)$$

Hash(pk, m, t)**:**

公開鍵 $pk \in \mathbb{G}_1$，メッセージ $m \in \mathbb{Z}_q$，及び時刻 $t \in \{0,1,2\}^l$ を入力とする．チェックストリング $r \in \mathbb{Z}_q$ をランダムに生成し，時刻 t におけるハッシュ値 H_t を

$$H_t = FSCH(pk, m, r, t)$$

により計算する．

Adapt(sk_t, m, m', r, H_t)**:**

時刻 $t \in \{0,1,2\}^l$ における秘密鍵 sk_t，元のメッセージ $m \in \mathbb{Z}_q$ と新たなメッセージ $m' \in \mathbb{Z}_q$，チェックストリング $r \in \mathbb{Z}_q$，及び時刻 t におけるハッシュ値 $H_t \in \mathbb{G}_\tau$ を入力とする．まず秘密鍵の所有者は $f(t)$ を用いて，衝突が生じる，すなわち

$$FSCH(pk, m', \hat{r}, t) = FSCH(pk, m, r, t) = H_t$$

となるような $\hat{r} \in \mathbb{Z}_q$ を

$$\hat{r} = f(t)^{-1}(m - m') + r \ \ (mod\ q)$$

により計算する．次に

$$\hat{r} = \hat{r_1} + \hat{r_2} \ \ (mod\ q)$$

なる $\hat{r_1}, \hat{r_2} \in \mathbb{Z}_q$ を計算し，c, d を用いて新たなチェックストリング $r' \in \mathbb{G}_1 \times \mathbb{G}_2$ を以下により計算する．

$$r' = \{r'_1, r'_2\} = \{g^{\hat{r_1}}/c,\ d \cdot F(t)^{\hat{r_2}}\}$$

その後，r' を公開する．

Verify(pk, m', r', H_t, t)**:**

公開鍵 $pk \in \mathbb{G}_1$，メッセージ $m' \in \mathbb{Z}_q$，チェックストリング $r' = \{r'_1, r'_2\} \in \mathbb{G}_1 \times \mathbb{G}_2$，時刻 $t \in \{0,1,2\}^l$ におけるハッシュ値 $H_t \in \mathbb{G}_\tau$，及び時刻 t を入力とする．ハッシュ値が有効かどうかを以下の検証式により検証する．

$$\mathbf{e}(r'_1, F(t)) \cdot \mathbf{e}(g, r'_2) \cdot \mathbf{e}(g^{m'}, h) = H_t$$

検証式が成り立つ場合は $b = 1$ を，成り立たない場合は $b = 0$ を出力する．

4.3 正当性の証明

本節では，4.2 節で示したプロトコルを用いて定義 5 で定義した正当性を満たすことを示す．

証明：

Verify アルゴリズムの出力が 1 となる，すなわち検証式が正しいことを示す．

$$
\begin{aligned}
&\mathbf{e}(r_1', F(t)) \cdot \mathbf{e}(g, r_2') \cdot \mathbf{e}(g^{m'}, h) \\
&= \mathbf{e}(g^{\hat{r_1}}/c, F(t)) \cdot \mathbf{e}(g, d \cdot F(t)^{\hat{r_2}}) \cdot \mathbf{e}(g^{m'}, h) \\
&= \mathbf{e}(g^{\hat{r_1}}, F(t)) \cdot \mathbf{e}(g^{-s}, F(t)) \cdot \mathbf{e}(g, h^x) \cdot \mathbf{e}(g, F(t)^s) \\
&\quad \cdot \mathbf{e}(g, F(t)^{\hat{r_2}}) \cdot \mathbf{e}(g^{m'}, h) \\
&= \mathbf{e}(g^{\hat{r_1}}, F(t)) \cdot \cancel{\mathbf{e}(g^{-s}, F(t))} \cdot \mathbf{e}(g, h^x) \cdot \cancel{\mathbf{e}(g^{s}, F(t))} \\
&\quad \cdot \mathbf{e}(g, F(t)^{\hat{r_2}}) \cdot \mathbf{e}(g^{m'}, h) \\
&= \mathbf{e}(g^{\hat{r_1}}, F(t)) \cdot \mathbf{e}(g, h^x) \cdot \mathbf{e}(g, F(t)^{\hat{r_2}}) \cdot \mathbf{e}(g^{m'}, h) \\
&= \mathbf{e}(g^{\hat{r}}, F(t)) \cdot \mathbf{e}(pk \cdot g^{m'}, h) = H_t
\end{aligned}
$$

このように，入力 (pk, m', r', H_t, t) が正しいとき明らかに検証式は満たされるから，$\mathsf{Verify}(pk, m', r', H_t, t) = 1$．よって，FS-CH は正当性を満たす．

4.4 安全性の証明

本節では， 4.2 節で示したプロトコルを用いて定理 1 が成り立つことを示す．

証明：

証明を 2 つのステップに分けて行う．まずは定義 7 で定義した選択的 Forward-Secure 衝突耐性のゲームを考える．

ステップ 1. 選択的 Forward-Secure 衝突耐性

ステップ 1 では，l-wBDHI 問題を解くことが計算量的に困難という仮定の下で FS-CH が選択的 Forward-Secure 衝突耐性を満たすことを示す．すなわち，選択的 Forward-Secure 衝突耐性を攻撃する敵対者 $\mathcal{A}'$ 及び l-wBDHI 問題を解く敵対者 $\mathcal{B}$ を仮定した時，$\mathcal{B}$ は $\mathcal{A}'$ との間で選択的 Forward-Secure 衝突耐性のゲームを行うことで，l-wBDHI 問題を解くことができることを示す．

(1) Setup フェーズ

はじめに，敵対者 $\mathcal{A}'$ は，Challenging フェーズで出力する時刻 t^* を決定し，これを $\mathcal{B}$ に送信する．続いて $\mathcal{B}$ は，l-wBDHI 問題の入力を用いて公開パラメータを以下のようにシミュレートする．なおこのとき，$w^* = t^*$ なる時刻 $w^* = {w_0}^* \parallel {w_1}^* \parallel ... \parallel {w_l}^* \in \{0,1,2\}^l$ を用いる．

$$
\begin{aligned}
pk &\leftarrow A_1 = g^{\alpha} \\
h &\leftarrow C_2 \cdot B_l = h^{\gamma} \cdot h^{\alpha^l} \\
h_0 &\leftarrow h^{\gamma_0} \cdot \prod_{i=1}^{l-1} B_{l-i+1}^{-{w_i}^*} \cdot {B_1}^{-1} \\
h_i &\leftarrow h^{\gamma_i} \cdot B_{l-i+1}^{w_0} (1 \le i \le l) \\
&\quad (\gamma_0, ..., \gamma_i \in \mathbb{Z}_q)
\end{aligned}
$$

ここで，$pk = g^x$ より $\alpha = x$，また $h^x = h^{\alpha} = (h^{\gamma} \cdot h^{\alpha^l})^{\alpha} = {B_1}^{\gamma} \cdot h^{\alpha^{l+1}}$ となることに注意されたい．

これらの値を設定した後，$\mathcal{B}$ は $\mathcal{A}'$ に時刻の最大値 T 及び公開鍵 pk を与える．

(2) Training フェーズ

$\mathcal{B}$ は $\mathcal{A}'$ の問い合わせに対して各オラクル (KeyUpdate,Hash,Adapt,Break in) をシミュレートし，応答を返す必要がある．各オラクルのシミュレートは以下のように行われる．

KeyUpdate オラクル:

$\mathcal{B}$ は現在の時刻 t の更新を知っておけばよい．またこのオラクルは返り値を返さないため，$\mathcal{B}$ は値のシミュレートを行わない．

Hash オラクル

ハッシュ値 H_t の計算は全てのユーザが行えるので，$\mathcal{B}$ が $\mathcal{A}'$ に H_t を返すことができるのは明らかである．

Adapt オラクル

このオラクルにおける $\mathcal{B}$ のシミュレートは 2 つのケース分けて行われる．まずは，$\mathcal{A}'$ がこのオラクルで入力とした t と Setup フェーズで決定した t^* が異なるケースにおける $\mathcal{B}$ のシミュレートがどのように行われるかについて示す．

(Case1. $t \neq t^*$)

$\mathcal{B}$ は Setup フェーズで設定した値を用いて秘密鍵，すなわち $s\tilde{k}_w$ と $f(t)$ をシミュレートしたのち，r' を計算する．まず $s\tilde{k}_w$ は

$$
\begin{aligned}
s\tilde{k}_w &= (c, d, e_{k+1}, ...e_l) \\
&= (g^s, h^x \cdot (\prod_{i=0}^{k} h_i^{w_i})^s, {h_{k+1}}^s, ...{h_l}^s)
\end{aligned}
$$

である．この c, d, e_i を以下のように変形する．まず d について，

$$
\begin{aligned}
d &= h^x \cdot (\prod_{i=0}^{k} h_i^{w_i})^s \\
&= {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot ({h_0}^{w_0} \cdot \prod_{i=1}^{k} h_i^{w_i})^s \\
&= {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot ({h_0}^{\gamma_0 w_0} \cdot \prod_{i=1}^{l-1} B_{l-i+1}^{-w_0 {w_i}^*} \cdot {B_1}^{-w_0} \\
&\quad \cdot \prod_{i=1}^{k} h^{\gamma_i w_i} \cdot B_{l-i+1}^{w_0 w_i})^s \\
&= {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot (h^{\sum_{i=0}^{k} \gamma_i w_i} \cdot B_{l-i+1}^{w_0 (w_k - {w_k}^*)} \\
&\quad \cdot \prod_{i=k+1}^{l-1} B_{l-i+1}^{-w_0 {w_i}^*} \cdot {B_1}^{-w_0})^s
\end{aligned}
$$

($\because 1 \le i \le k-1$ のとき $w_i = {w_i}^*$ であるから

$$B_{l-i+1}^{-w_0 {w_i}^*} \cdot B_{l-i+1}^{w_0 {w_i}^*} = 1$$

また，$i = k$ のとき $w_i \neq {w_i}^*$ であるから

$$B_{l-k+1}^{-w_0 {w_i}^*} \cdot B_{l-k+1}^{w_0 {w_i}^*} = B_{l-i+1}^{w_0 (w_k - {w_k}^*)})$$

ここで，右辺の () 内全体を F，() 内の · で区切られた要素をそれぞれ F_1, F_2, F_3, F_4 とすると，

$$d = {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot F^{s} \ \ (F = F_1 \cdot F_2 \cdot F_3 \cdot F_4)$$

となる．また，$\mathcal{B}$ は $s' \in \mathbb{Z}_q$ をランダムに選択し s を以下のように設定する．

$$s \leftarrow s' + \frac{\alpha^k}{w_0(w_k^* - w_k)} \ (mod\ q)$$

すなわち，

$$d = {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot F^{s'} \cdot F^{\frac{\alpha^k}{w_0(w_k - {w_k}^*)}}$$

となる．このとき $\mathcal{B}$ は，上式右辺の ${B_1}^{\gamma}$ 及び $F^{s'}$ については計算可能であるが，$h^{\alpha^{l+1}}$ については計算することができない．$F^{\frac{\alpha^k}{w_0(w_k - {w_k}^*)}}$ について，F_1, F_2, F_3, F_4 それぞれに分けて変形すると

$$F_1{}^{\frac{\alpha^k}{w_0(w_k^* - w_k)}} = B_k{}^{\frac{\sum_{i=0}^{k} \gamma_i w_i}{w_0(w_k^* - w_k)}}$$

$$F_2{}^{\frac{\alpha^k}{w_0(w_k^* - w_k)}} = {B_{l-k+1}}^{-\alpha^k} = h^{-\alpha^{l+1}}$$

$$F_3{}^{\frac{\alpha^k}{w_0(w_k^* - w_k)}} = \prod_{i=k+1}^{l-1} B_{l+k-i+1}^{\frac{-{w_i}^*}{{w_k}^* - w_k}} = \prod_{i=0}^{l-k-2} B_{l-1}^{\frac{-{w_i}^* + 2 + i}{{w_k}^* - w_k}}$$

$$F_4{}^{\frac{\alpha^k}{w_0(w_k^* - w_k)}} = B_{k+1}^{\frac{-1}{{w_k}^* - w_k}}$$

となる．ここで $\mathcal{B}$ は $h^{-\alpha^{l+1}}$ 以外の全ての値を計算することができ，$h^{-\alpha^{l+1}}$ については上述の $h^{\alpha^{l+1}}$ とキャンセルされる．以上より，$\mathcal{B}$ は d をシミュレートできる．次に c, e_i については，

$$c = g^s = g^{s'} \cdot g^{\frac{\alpha^k}{w_0(w_k^* - w_k)}} = g^{s'} \cdot {A_k}^{\frac{1}{w_0(w_k^* - w_k)}}$$

$$e_i = {h_i}^{s'} \cdot B_{l+k-i+1}\ (k+1 \le i \le l)$$

$$e_i = {h_{k+i}}^{s'} \cdot B_{l-i}\ (0 \le i \le k-1)$$

として計算可能である．以上より，$\mathcal{B}$ は $s\tilde{k}_w$ をシミュレートできる．

続いて $f(t) = f(w)$ について，$f(w) = \sum_{i=0}^{k} z_i w_i \ (mod\ q)$ であるから，$\mathcal{B}$ は $z_0, z_1, ..., z_k$ を計算する必要がある．$\mathcal{B}$ は

$$h_0 = h^{z_0}, h_1 = h^{z_1}, ..., h_l = h^{z_l}$$

及び Setup で設定した

$$h_0 \leftarrow h^{\gamma_0} \cdot \prod_{i=1}^{l-1} B_{l-i+1}^{-{w_i}^*} \cdot {B_1}^{-1}$$

$$h_i \leftarrow h^{\gamma_i} \cdot B_{l-i+1}^{w_0} (1 \le i \le l)$$

より，$z_0, z_1, ..., z_k$ を

$$z_0 = \gamma_0 - \sum_{i=1}^{l-1} \alpha^{l-i+1} \cdot {w_i}^* - \alpha \ (mod\ q)$$

$$z_i = \gamma_i + \alpha^{l-i+1} \cdot w_0 \ (mod\ q)\ (1 \le i \le l)$$

で計算できる．以上より，$\mathcal{B}$ は $f(t)$ をシミュレートできる．よってこれらの値を用いて，入力に対する正しい r' をシミュレートし $\mathcal{A}'$ に返すことができる．

(Case2. $t = t^*$)

Case1 と異なり，$\mathcal{B}$ は秘密鍵 $s\tilde{k}_w$ の c, d 及び $\hat{r_1}, \hat{r_2}$ を求めることにより，$r' = \{r'_1, r'_2\} = \{g^{\hat{r_1}}/c,\ d \cdot F(t)^{\hat{r_2}}\}$ を直接計算する．まず d について，

$$d = h^x \cdot (\prod_{i=0}^{k} h_i^{w_i})^s$$

$$= {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot (h^{\sum_{i=0}^{k} \gamma_i w_i} \cdot \prod_{i=k+1}^{l-1} B_{l-i+1}^{w_0(w_k - {w_k}^*)} \cdot {B_1}^{-w_0})^s$$

ここで，上式右辺の () 内全体を F，() 内の $\cdot$ で区切られた要素をそれぞれ F_1, F_2, F_3 とすると，

$$d = {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot F^{s} \ \ (F = F_1 \cdot F_2 \cdot F_3)$$

となる．また，$\mathcal{B}$ は $s' \in \mathbb{Z}_q$ をランダムに選択し s を以下のように設定する．

$$s \leftarrow s' + \frac{\alpha^l}{w_0} \ (mod\ q)$$

すなわち，

$$d = {B_1}^{\gamma} \cdot h^{\alpha^{l+1}} \cdot F^{s'} \cdot F^{\frac{\alpha^l}{w_0}}$$

となる．このとき $\mathcal{B}$ は，上式右辺の ${B_1}^{\gamma}$ 及び $F^{s'}$ については計算可能であるが，$h^{\alpha^{l+1}}$ については計算することができない．$F^{-\frac{\alpha^l}{w_0}}$ における $F_3{}^{\frac{\alpha^l}{w_0}}$ を変形すると，

$$F_3{}^{\frac{\alpha^l}{w_0}} = {B_1}^{-\alpha^l} = h^{-\alpha^{l+1}}$$

となるから，上述の $h^{\alpha^{l+1}}$ とキャンセルされる．$F_1{}^{\frac{\alpha^l}{w_0}}, F_2{}^{\frac{\alpha^l}{w_0}}$ については Case1 と同様にして計算できる．次に c については，

$$c = g^s = g^{s'} \cdot g^{\frac{\alpha^l}{w_0}} = g^{s'} \cdot {A_l}^{\frac{1}{w_0}}$$

として計算可能である．また $\hat{r_1}, \hat{r_2}$ については，Case1 と同様に $f(t)$ をシミュレートすることにより計算することができる．以上より，$\mathcal{B}$ は $r' = \{r'_1, r'_2\} = \{g^{\hat{r_1}}/c,\ d \cdot F(t)^{\hat{r_2}}\}$ を c, d 及び $\hat{r_1}, \hat{r_2}$ を求めることによりシミュレートし，$\mathcal{A}'$ に返すことができる．

Break in オラクル

Adapt オラクルのシミュレートより，$\mathcal{B}$ は全ての時刻 w に関する秘密鍵 $s\tilde{k}_w$ をシミュレートできる．よって，$\mathcal{B}$ は現在時刻 $\tilde{t}$ に関する秘密鍵 $sk_{\tilde{t}}$ をシミュレートし，$\mathcal{A}'$ に返すことができる．

(3) Challenging フェーズ

$\mathcal{B}$ は $\mathcal{A}'$ の出力した $(m^*, m^{*\prime}, r^*, r^{*\prime}, t^*, H_{t^*})$ を用いて，l-wBDHI 問題の出力 $\mathbf{e}(g,h)^{\gamma \cdot \alpha^{l+1}}$ を計算する．$r^{*\prime} = \{r_1^{*\prime}, r_2^{*\prime}\} = \{g^{\hat{r_1}^*}/c,\ d \cdot F(t)^{\hat{r_2}^*}\}$ を用いると，以下に示す等式が導かれる．

$$\mathbf{e}(C_1, r_2^{*\prime}) \cdot \mathbf{e}(r_1^{*\prime}, F(t^*)^r)$$
$$= \mathbf{e}(C_1, h^x) \cdot \mathbf{e}(C_1, F(t^*)^s) \cdot \mathbf{e}(C_1, F(t^*)^{\hat{r_2^*}})$$
$$\cdot \mathbf{e}(g^{\hat{r_1^*}}, F(t^*)^r) \cdot e(g^{-s}, F(t^*)^r)$$
$$= \mathbf{e}(C_1, B_1{}^\gamma) \cdot \mathbf{e}(C_1, h^{\alpha^{l+1}}) \cdot \cancel{\mathbf{e}(C_1, F(t^*)^s)} \cdot \mathbf{e}(C_1, F(t^*)^{\hat{r_2^*}})$$
$$\cdot \mathbf{e}(g^\gamma, F(t^*)^{\hat{r_1^*}}) \cdot \cancel{\mathbf{e}(g^\gamma, F(t^*)^{-s})}$$

$$\therefore \mathbf{e}(C_1, r_2^{*\prime}) \cdot \mathbf{e}(r_1^{*\prime}, F(t^*)^r)$$
$$= \mathbf{e}(C_1, B_1{}^\gamma) \cdot \mathbf{e}(C_1, h^{\alpha^{l+1}}) \cdot \mathbf{e}(C_1, F(t^*)^{\hat{r^*}})$$

ここで $\mathcal{B}$ は，左辺及び右辺の $\mathbf{e}(C_1, B_1{}^\gamma)$ については計算を行うことができる．また右辺の $\mathbf{e}(C_1, F(t^*)^{\hat{r^*}})$ についても，$\mathcal{B}$ は $\hat{r^*}$ を知らないため直接計算することができないが，関数 $FSCH$ を用いると以下のように計算できる．すなわち，

$$FSCH(pk, m^{*\prime}, \hat{r^*}, t^*)$$
$$= \mathbf{e}(pk \cdot g^{m^{*\prime}}, h) \cdot \mathbf{e}(g^{\hat{r^*}}, F(t^*)) = H_{t^*}$$

この両辺を γ 乗すると，

$$\mathbf{e}(pk \cdot g^{m^{*\prime}}, h)^\gamma \cdot \mathbf{e}(g^{\hat{r^*}}, F(t^*))^\gamma = H_{t^*}{}^\gamma$$
$$\mathbf{e}(pk \cdot g^{m^{*\prime}}, h)^\gamma \cdot \mathbf{e}(C_1, F(t^*)^{\hat{r^*}}) = H_{t^*}{}^\gamma$$

となる．ここで $\mathcal{B}$ は上式の左辺の $\mathbf{e}(pk \cdot g^{m^{*\prime}}, h)^\gamma$ 及び右辺の $H_{t^*}{}^\gamma$ を計算できるため，$\mathbf{e}(C_1, F(t^*)^{\hat{r^*}})$ を計算することができる．以上より，$\mathcal{B}$ は $\mathbf{e}(C_1, h^{\alpha^{l+1}})$ 以外の全ての値を計算することで，$\mathbf{e}(C_1, h^{\alpha^{l+1}})$ を計算することができる．またこの値は $\mathbf{e}(C_1, h^{\alpha^{l+1}}) = \mathbf{e}(g^\gamma, h^{\alpha^{l+1}}) = \mathbf{e}(g, h)^{\gamma \cdot \alpha^{l+1}}$ より l-wBDHI 問題の出力となる．

よって $\mathcal{A}'$ が正しく $r^{*\prime}$ を計算できている場合，$\mathcal{B}$ は必ず l-wBDHI 問題の出力 $\mathbf{e}(g, h)^{\gamma \cdot \alpha^{l+1}}$ を計算できるので，以下の式 (1) が導かれる．

$$\mathbf{Adv}_{FS}^{s-colres}(\mathcal{A}') \leq \mathbf{Adv}_{\mathbb{G}_1 \times \mathbb{G}_2}^{l-\mathrm{wBDHI}}(\mathcal{B}) \qquad (1)$$

ステップ 2.Forward-Secure 衝突耐性

ステップ 2 では，Forward-Secure 衝突耐性のゲームと選択的 Forward-Secure 衝突耐性のゲームの攻撃成功確率の関係を示す．すなわち，Forward-Secure 衝突耐性を攻撃する敵対者 $\mathcal{A}$ 及び選択的 Forward-Secure 衝突耐性を攻撃する敵対者 $\mathcal{A}'$ を仮定すると，$\mathcal{A}$ は $\mathcal{A}'$ に比べ時刻の最大値 T 倍だけ Challenging フェーズにおける出力の候補を持つから，以下の式 (2) が導かれる．

$$\mathbf{Adv}_{FS}^{colres}(\mathcal{A}) = T \cdot \mathbf{Adv}_{FS}^{s-colres}(\mathcal{A}')$$
$$\therefore \frac{1}{T} \cdot \mathbf{Adv}_{FS}^{colres}(\mathcal{A}) = \mathbf{Adv}_{FS}^{s-colres}(\mathcal{A}') \qquad (2)$$

式 (1)，(2) より，定理 1 で示した以下の式が導かれる．

$$\frac{1}{T} \cdot \mathbf{Adv}_{FS}^{colres}(\mathcal{A}) \leq \mathbf{Adv}_{\mathbb{G}_1 \times \mathbb{G}_2}^{l-\mathrm{wBDHI}}(\mathcal{B})$$

5. おわりに

本研究では，非対称な双線形写像を用いて，Forward-Security を導入した初めてのカメレオンハッシュ関数 (FS-CH) を提案し，その正当性及び安全性の証明を行った．本提案により，CH ベースの手法において秘密鍵が漏洩することによる攻撃のリスクの軽減を実現した．双線形写像の計算は計算コストが大きいため，その削減は今後の課題である．また，本提案に関する実装は未だ行っていないためこちらも今後の課題とする．

謝辞 本研究の一部は文部科学省「Society5.0 に対応した高度技術人材育成事業成長分野を支える情報技術人材の育成拠点の形成 (enPiT)」，文部科学省の平成 30 年度「Society 5.0 実現化研究拠点支援事業」，さらに JSPS 科研費 JP21H034438 の助成を受けています．

参考文献

[1] Hugo Krawczyk and Tal Rabin. Chameleon hashing and signatures. *IACR Cryptol. ePrint Arch.*, 1998:10, 1998.
[2] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.
[3] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS*, volume 3679, pages 159–177. Springer, 2005.
[4] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton R. Andrade. Redactable blockchain - or - rewriting history in bitcoin and friends. In *EuroS&P*, pages 111–126. IEEE, 2017.
[5] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-hashes with ephemeral trapdoors - and applications to invisible sanitizable signatures. In Serge Fehr, editor, *IACR*, volume 10175 of *Lecture Notes in Computer Science*, pages 152–182. Springer, 2017.
[6] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. In *NDSS*. The Internet Society, 2019.
[7] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In Srdjan Capkun and Franziska Roesner, editors, *USENIX*, pages 2093–2110. USENIX Association, 2020.
[8] Xiaofeng Chen, Fangguo Zhang, Haibo Tian, Baodian Wei, and Kwangjo Kim. Key-exposure free chameleon hashing and signatures based on discrete logarithm systems. volume 2009, page 35, 2009.