Web Workersを用いた多変数公開鍵暗号Rainbowの並列実装

鷲見 拓哉† 石黒 司‡ 清本 晋作‡ 三宅 優‡ 小林 透‡† 高木 剛‡‡

† 九州大学大学院数理学府
819-0395 福岡県福岡市西区元岡 744

‡ 株式会社 KDDI 研究所
356-8502 埼玉県ふじみ野市大原 2-1-15

‡† 長崎大学大学院工学研究科
852-8521 長崎県長崎市文教町 1-14

‡‡ 九州大学マス・フォア・インダストリ研究所
819-0395 福岡県福岡市西区元岡 744

あらまし W3C は，HTML5 及び JavaScript 上で並列計算を行うための規格である Web Workers の勧告候補を 2012 年に公開した．Rainbow 署名は，Ding と Schmidt により 2005 年に提案された多変数公開鍵暗号方式のディジタル署名である．Rainbow 署名は有限体上の多変数 2 次多項式の連立方程式に対する求解問題が NP 困難であることを安全性の根拠とし，ポスト量子暗号の一つとして期待されている．本稿では，マルチコア CPU を搭載する汎用 PC 及びタブレット端末上で Rainbow 署名を JavaScript 言語を用いて実装し，ウェブブラウザ上で並列計算を行った際の高速化率等を報告する．

Parallel Implementation of Multivariate Public Key Cryptosystem Rainbow by Using Web Workers

Takuya Sumi† Tsukasa Ishiguro‡ Shinsaku Kiyomoto‡ Yutaka Miyake‡
Toru Kobayashi‡† Tsuyoshi Takagi‡‡

†Graduate School of Mathematics, Kyushu University ‡KDDI R&D Laboratories, Inc.
‡†Graduate School of Engineering, Nagasaki University
‡‡Institute of Mathematics for Industry, Kyushu University

Abstract Web Workers is a specification which defines an API that allows Web application developers to spawn background workers running scripts in parallel. The W3C have announced a candidate recommendation of Web Workers in 2012. Rainbow signature scheme is a one of the multivariate public key cryptosystems proposed by Ding and Schmidt in 2005. In this paper, we implement Rainbow signature schemes in JavaScript with Web Workers on some multi-core CPU devices. We propose a parallel implementation applying to the signature verification processes of Rainbow. We report the speedup factor of parallel implementation by using Web Workers.

1 はじめに

HTML5 [1] は，HTML の次世代標準であり，W3C が策定作業を進めている．W3C は，HTML5 に関する最初の作業草案を 2008 年に公開した．その後，W3C は作業草案の改定を重ね，2012 年に勧告候補を公開した．HTML5 の策定に合わせ，ウェブアプリケーションが並列処理を行うための規格が設けられた．それが，Web Workers [2] である．Web Workers は，ウェブアプリケーションの実行環境にバックグラウンドワーカを提供する．ウェブアプリケーションの開発者は，バックグラウンドワーカを用いて処理を並列化することができる．Web Workers は，当初 HTML5 の一部として規格の策定が進められていたが，現在は HTML5 とは独立した規格として存在している．しかし，HTML5 と Web Workers は密接に関係している．W3C は，Web Workers に関する最初の作業草案を 2009 年に，勧告候補を 2012 年に公開した．HTML5 では，ウェブアプリケーションの

開発に利用できるマルチメディア機能やクライアントのローカルストレージ上にデータベースを構築できる Indexed Database API [3]，Web Storage [4]，ウェブブラウザにローカルストレージへのアクセスを提供する File API [5]，ウェブブラウザとウェブサーバ間の双方向通信を実現する WebSocket [6] 等を規定している．HTML5 を用いることにより，従来の HTML 4 では実現できなかったような高機能なウェブアプリケーションを開発することができる．

現在実用化されている公開鍵暗号には，RSA 暗号や楕円曲線暗号などがある．これらの暗号は，素因数分解問題や楕円曲線上の離散対数問題が困難であることを安全性の根拠とする．しかし，Shor のアルゴリズム [7] によれば，これらの問題は量子コンピュータを用いれば多項式時間で解くことができる．したがって，量子コンピュータを用いた場合においても解くことが困難な公開鍵暗号を構成する必要がある．有限体上の連立多変数非線形方程式において各係数をランダムに選んだ場合，その求解問題は NP 困難であることが知られている [8]．また，有限体上の連立多変数非線形方程式の求解は，量子コンピュータを用いた場合においても困難であると強く信じられている．ゆえに，多変数多項式を用いた公開鍵暗号が活発に研究されている．多変数公開鍵暗号には，Mixed-Field 型と呼ばれる松本今井暗号 [9] や HFE 署名 [10]，Single-Field 型と呼ばれる UOV 署名 [11] などが提案されている．

Rainbow 署名は，Ding と Schmidt により 2005 年に提案された多変数公開鍵暗号方式のディジタル署名である [12]．Rainbow 署名は多変数 2 次多項式の連立方程式に対する求解問題が困難であることを安全性の根拠とし，ポスト量子暗号の一つとして期待されている．Rainbow 署名の署名生成及び署名検証は，位数の小さな有限体上の多項式の演算により実装するため，RSA 署名と比較して処理速度が高速であるという利点を持つ．

本稿では，マルチコア CPU を搭載する汎用 PC 及びタブレット端末上で Rainbow 署名を JavaScript 言語を用いて実装した．Rainbow 署名の実装にあたり，署名検証については Web Workers を用いて並列化を施し，実装した Rainbow 署名をウェブブラウザ上で動作させた場合の署名生成及び署名検証に必要な実行時間を計測した．その上で，並列計算を行う場合の高速化率及び実行効率を算出し，Rainbow 署名をウェブブラウザ上で実装する応用例を示す．

1.1 ウェブアプリケーションの利点

ウェブアプリケーションを構築することの利点を次に示す [13]．

1. 利用者のプラットフォーム（OS，ウェブブラウザ，端末の種類）に依存しない．したがって，企業における製品ラインナップ展開戦略や販売した製品のメンテナンス体制の維持に掛かるコストの削減が期待できる．
2. 一般的なウェブブラウザで動作し，インストール作業は必要ない．ウェブページにアクセスするだけで良い．また，追加のソフトウェアは必要ない．
3. 従来ウェブサーバ上で行っていた処理を，クライアントのウェブブラウザ上で行うことにより，ウェブサーバの CPU 使用時間を削減し，クライアントとウェブサーバ間のデータ送受信に必要な通信トラフィックを削減することができる．これにより，ウェブサーバ設備や通信回線への初期投資やランニングコストの削減が期待できる．

2 Rainbow 署名

本章では，Rainbow 署名 [12, 14] について説明する．

2.1 Rainbow 署名の構築

$\mathbb{K}$ を有限体とする．$n \in \mathbb{N}$ に対し，$V = \{1, 2, \ldots, n\}$ とおく．$t \in \mathbb{N}$ に対し，$v_1, v_2, \ldots, v_{t+1} \in \mathbb{N}$ を

$$0 < v_1 < v_2 < \cdots < v_t < v_{t+1} = n$$

なるものとし，$m = n - v_1$ とおく．$i = 1, 2, \ldots, t+1$ に対し，$V_i = \{1, 2, \ldots, v_i\}$ とおくと，$\#V_i = v_i$ である．$i = 1, 2, \ldots, t$ に対し

$$O_i = V_{i+1} \setminus V_i = \{v_i + 1, v_i + 2, \ldots, v_{i+1}\},$$
$$o_i = v_{i+1} - v_i$$

とおく．$\#O_i = o_i$ である．

Rainbow 署名は，t 個のレイヤから構成される．$h = 1, 2, \ldots, t$ に対し，第 h レイヤでは，次に示す

表 1: 秘密鍵及び公開鍵の大きさ

方式	署名長（ビット）	秘密鍵		公開鍵	
		鍵長	大きさ (KBytes)	鍵長	大きさ (KBytes)
Rainbow($\mathbb{F}_{31}$; 27, 26, 26)	395	113674	69.4	168480	102.9

o_h 個の n 変数多項式を使う．$f_k(x_1, x_2, \ldots, x_n) = f_k(\boldsymbol{x})$ を次に示す 2 次多項式とする．

$$f_k(\boldsymbol{x}) = \sum_{i\in O_h, j\in V_h} \alpha_{i,j}^{(k)} x_i x_j + \sum_{i,j\in V_h, i\le j} \beta_{i,j}^{(k)} x_i x_j + \sum_{i\in V_{h+1}} \gamma_i^{(k)} x_i + \eta^{(k)} \qquad (k \in O_h)$$

ここで，$\alpha_{i,j}^{(k)}, \beta_{i,j}^{(k)}, \gamma_i^{(k)}, \eta^{(k)} \in \mathbb{K}$ である．x_i $(i \in O_h)$ を Oil 変数，x_j $(j \in V_h)$ を Vinegar 変数と呼ぶ．Rainbow 署名の中心写像 $\mathcal{F}\colon \mathbb{K}^n \to \mathbb{K}^m$ を次の通りとする．

$$\mathcal{F}(\boldsymbol{x}) = (\overbrace{f_{v_1+1}(\boldsymbol{x}),\ f_{v_1+2}(\boldsymbol{x}),\ \ldots,\ f_n(\boldsymbol{x})}^{m \text{ 個の } n \text{ 変数多項式}})$$

$f_k(\boldsymbol{x})$ $(k \in O_h)$ は Oil 変数が 1 次線形の形で現れるという特徴を持っており，$f_k(\boldsymbol{x})$ に現れる全ての Vinegar 変数を固定すると，Oil 変数に関する 1 次式になる．

いま，$f_k(\boldsymbol{x})$ と固定された $(b_1, b_2, \ldots, b_{v_h}) \in \mathbb{K}^{v_h}$ に対して，$x_1 = b_1$, $x_2 = b_2$, …, $x_{v_h} = b_{v_h}$ なる代入を行えば，Oil 変数 x_i $(i \in O_h)$ に関する 1 次多項式 $\tilde{f}_k(x_{v_h+1}, x_{v_h+2}, \ldots, x_{v_{h+1}}) = \tilde{f}_k(\tilde{\boldsymbol{x}})$ が作れる．$\tilde{f}_k(\tilde{\boldsymbol{x}})$ と適当な $(a_{v_h+1}, a_{v_h+2}, \ldots, a_{v_{h+1}}) \in \mathbb{K}^{o_h}$ を用いて連立 1 次方程式を解くことにより，中心写像 $\mathcal{F}$ の逆像（の一つ）を簡単に求めることができる．以下に，署名生成において $\boldsymbol{b} = \mathcal{F}^{-1}(\boldsymbol{a})$ を計算する方法を示す．

$\boldsymbol{b} = \mathcal{F}^{-1}(\boldsymbol{a})$ の計算方法

1. $(b_1, b_2, \ldots, b_{v_1}) \in \mathbb{K}^{v_1}$ をランダムに決める．
2. $\boldsymbol{a} = (a_{v_1+1}, a_{v_1+2}, \ldots, a_n) \in \mathbb{K}^m$ とする．$h = 1, 2, \ldots, t$ に対して，Oil 変数 x_i $(i \in O_h)$ に関する連立 1 次方程式

   $$\tilde{f}_k(x_{v_h+1}, x_{v_h+2}, \ldots, x_{v_{h+1}}) = a_k \quad (k \in O_h)$$

   の解 $(b_{v_h+1}, b_{v_h+2}, \ldots, b_{v_{h+1}}) \in \mathbb{K}^{o_h}$ を計算する．解がなければ 1. へ戻る．
3. $(b_1, b_2, \ldots, b_n) \in \mathbb{K}^n$ が中心写像 $\mathcal{F}$ の逆像（の一つ）である．

鍵生成

秘密鍵 中心写像 $\mathcal{F}$ と 2 つのアフィン同型写像

$$\mathcal{S}(\boldsymbol{y}) = S\boldsymbol{y} + \boldsymbol{c}_S : \mathbb{K}^m \to \mathbb{K}^m,$$
$$\mathcal{T}(\boldsymbol{x}) = T\boldsymbol{x} + \boldsymbol{c}_T : \mathbb{K}^n \to \mathbb{K}^n.$$

ここで，$S \in \mathbb{K}^{m\times m}$, $T \in \mathbb{K}^{n\times n}$ は正則行列であり，$\boldsymbol{y}, \boldsymbol{c}_S \in \mathbb{K}^m$, $\boldsymbol{x}, \boldsymbol{c}_T \in \mathbb{K}^n$ である．

公開鍵 合成写像 $\mathcal{P} = \mathcal{S}\circ\mathcal{F}\circ\mathcal{T} : \mathbb{K}^n \to \mathbb{K}^m$.

$$\mathcal{P}(\boldsymbol{x}) = \mathcal{S}\circ\mathcal{F}\circ\mathcal{T}(\boldsymbol{x}) = (p_{v_1+1}(\boldsymbol{x}), p_{v_1+2}(\boldsymbol{x}), \ldots, p_n(\boldsymbol{x}))$$

署名生成 署名対象のメッセージを $\boldsymbol{M} \in \mathbb{K}^m$ とする．$\boldsymbol{a} = \mathcal{S}^{-1}(\boldsymbol{M})$, $\boldsymbol{b} = \mathcal{F}^{-1}(\boldsymbol{a})$, $\boldsymbol{c} = \mathcal{T}^{-1}(\boldsymbol{b})$ の順に $\boldsymbol{a} \in \mathbb{K}^m$, $\boldsymbol{b} \in \mathbb{K}^n$, $\boldsymbol{c} \in \mathbb{K}^n$ を計算する．$\boldsymbol{c}$ が $\boldsymbol{M}$ に対する署名である．

署名検証 $\mathcal{P}(\boldsymbol{c}) = \boldsymbol{M}$ ならば署名は有効である．

この方式を Rainbow($\mathbb{K}; v_1, o_1, o_2, \ldots, o_t$) と表し，$(\mathbb{K}; v_1, o_1, o_2, \ldots, o_t)$ を Rainbow 署名のパラメータと呼ぶ．

2.2 パラメータと鍵長

本稿では，安全性が 128 ビットとなる $\mathbb{F}_{31}$ 上の Rainbow 署名を実装する．Rainbow 署名で用いるパラメータは，$(\mathbb{F}_{31}; 27, 26, 26)$ である [15]．また，秘密鍵及び公開鍵の構成に必要な $\mathbb{K}$ の元の個数を鍵長と呼ぶ．鍵長は次の通りである．

秘密鍵長 $m(m+1) + n(n+1) + \sum_{h=1}^{t} o_h\,(v_h o_h + v_h(v_h+1)/2 + v_{h+1} + 1)$

公開鍵長 $m(n+1)(n+2)/2$

表 1 に，秘密鍵及び公開鍵の大きさを示す．$\mathbb{F}_{31}$ の 1 つの元は，5 ビットで表現できる．表 1 に示す大きさは，元の表現に必要なビット数を基に算出した理論値である．

表 2: 実装・計測に使用した計算機の性能

端末	OS	CPU	コア数	RAM
PC	Windows 7 64 ビット	AMD Phenom II X6 1090T 3.20 GHz	6	4.0 GB
Nexus 7	Android 4.3	NVIDIA Tegra 3 T30L (Cortex-A9) Quad-core 1.3 GHz	4	1.0 GB

表 3: 実装・計測に使用したウェブブラウザとそのバージョン

ウェブブラウザ	JavaScript エンジン	バージョン	
		PC	Nexus 7
Internet Explorer	Chakra	10.0.9200.16660	N/A[1]
Google Chrome	V8	29.0.1547.57	29.0.1547.59
Opera	V8	15.0.1147.153	15.0.1162.61541
Firefox	SpiderMonkey	22.0	22.0

[1] Android 版 Internet Explorer は提供されていない.

3 実装手法

本章では，Web Workers を用いた Rainbow 署名の並列実装手法について説明する．

3.1 有限体の演算と実装環境

有限体 $\mathbb{K}$ は

$$\mathbb{F}_{31} = \{0, 1, \ldots, 30\}$$

とする．$\mathbb{F}_{31}$ 上の加算，減算及び乗算は都度計算，逆元算は $a \in \mathbb{F}_{31}$ に対し，$a^{-1} = a^{29}$ として計算する．

表 2 に，実装・計測に使用した計算機の性能を示す．実装・計測には，表 3 に示すウェブブラウザを使用する．表 3 に示すウェブブラウザのうち，Google Chrome 及び Opera は同一の JavaScript エンジンを搭載する．JavaScript エンジンの Chakra 及び SpiderMonkey は，JavaScript コードをバイトコードへ変換し，インタプリタ方式で実行する．その後，実行中の状況に応じて，バイトコードをコンパイルしネイティブコードを生成する．V8 は，JavaScript コードを最初の実行時にコンパイルし，バイトコードへ変換することなくネイティブコードを生成する．

3.2 並列計算の性能指標

並列計算に関する性能指標について説明する [16]．以降の説明において，ある処理のうち，並列実行できる部分の実行時間の割合を p，使用する CPU のコア数を n とする．

高速化率 S を次の式で定める．

$$S = \frac{\text{逐次実行の所要時間}}{\text{並列実行の所要時間}}$$

高速化率は，逐次実行と比較して，並列実行がどの程度速くなったかを表す．

例 1 ある処理の逐次実行に必要となる時間が 1000 ミリ秒であり，同じ処理を並列実行した際の所要時間が 200 ミリ秒であるとする．このときの高速化率は $S = 5$ である．

実行効率 E を次の式で定める．

$$E = \frac{S}{n}$$

実行効率は，計算機のリソースがどの程度消費されているかを表す．並列数 P の並列計算を行うとき，$P < n$ ならば $E \leq \frac{S}{P} < \frac{S}{n}$ である．

例 2 64 コアの CPU を用いて，53 倍の高速化率が得られたならば，実行効率は $E = 0.828$ である．これは，処理全体を平均して，実行時間の約 17 %は CPU の全コアがアイドル状態になっているという意味である．

また，アムダールの法則を用いることにより，達成可能な高速化率の上限を知ることができる．アムダールの法則によれば

$$S \leq \frac{1}{(1-p) + \frac{p}{n}}$$

である．

3.3 JavaScript と Web Workers

JavaScript は，オブジェクト指向のスクリプト言語である．JavaScript は，国際的な標準化団体である Ecma International により標準化され，ECMAScript（ECMA-262 [17]）としてその仕様が公開されている．各ウェブブラウザは，ECMA-262 に基づき，JavaScript を実装している．HTML がウェブページの構造を定義するのに対し，JavaScript はウェ

ブページに動的な振る舞いを提供する．HTML と JavaScript を組み合わせて用いることにより，ユーザの入力に応じた，動的な振る舞いを行う高度なウェブアプリケーションを開発することができる．

Web Workers の策定目的は，JavaScript の実行がシングルスレッドであることに起因するウェブブラウザのユーザインタフェーススレッド（UI スレッド）の応答性の低下を解消することである．Web Workers 登場以前は，JavaScript の実行環境は常にシングルスレッドであり，JavaScript 上で並列計算を行うことはできなかった．しかし，Web Workers の登場により，JavaScript の実行環境に専用のバックグラウンドワーカを導入できるようになった．ウェブブラウザの UI スレッドとバックグラウンドワーカは，メッセージパッシング方式による通信を行うことができる．この通信を用いて，UI スレッドからバックグラウンドワーカへデータを転送し，計算量の多い問題をバックグラウンドワーカ上で処理することができる．これにより，UI スレッドの応答性低下を解消することができる．また，Web Workers を用いれば，ウェブアプリケーションの開発者はバックグラウンドワーカをいくつでも生成できる．生成された各バックグラウンドワーカは，それぞれが独立したアドレス空間を持つスレッドとして独立に動作する．したがって，計算対象のデータを分割し，分割したデータを各バックグラウンドワーカへ転送することにより，ウェブブラウザ上で並列計算を行うことができる．UI スレッド，バックグラウンドワーカ間の通信には postMessage 関数を使用する．

3.4 署名生成と署名検証

Rainbow 署名の署名生成は，署名者の秘密鍵 $\mathcal{S}$, $\mathcal{F}$, $\mathcal{T}$ を用いて行う．HTML5 登場以前は，セキュリティ上の理由により，JavaScript プログラムからローカルストレージにアクセスすることができなかった．しかし，HTML5 で新たに組み込まれた File API を用いることにより，JavaScript プログラムからローカルストレージへアクセスすることができるようになった．本稿における実装では，JavaScript プログラムが署名者の秘密鍵を読み込むために，File API のうち FileReaderSync 関数をバックグラウンドワーカ上で使用する．

Rainbow 署名の署名検証は，署名者の公開鍵 $\mathcal{P}$ を用いて行う．本稿では，Rainbow 署名の署名検証における $\mathcal{P}(\boldsymbol{c})$ の計算を並列に行う．並列計算は，

アルゴリズム 1 Rainbow 署名の署名検証の実行時間計測アルゴリズム

入力: 公開鍵 $\mathcal{P}$
出力: 署名検証の実行時間
1: $N \leftarrow$ 並列数の最大値
2: $L \leftarrow$ ランダムなメッセージの個数
3: $M \leftarrow L$ 組のランダムなメッセージ
4: $C \leftarrow M$ に対する，L 組の署名
5: **for** $P = 1$ to N **do**
6: バックグラウンドワーカを P 個生成する．
7: $t_0 \leftarrow$ 時刻
8: P 個のバックグラウンドワーカへ $\mathcal{P}, C$ を転送する．
9: **for** $i = 1$ to L **do**
10: P 個のバックグラウンドワーカを用いて，$\mathcal{P}(C_i)$ を計算する．
11: **end for**
12: UI スレッドへ $\mathcal{P}(C_i)$ $(i \in \{1, 2, \ldots, L\})$ を転送する．
13: $t_1 \leftarrow$ 時刻
14: **if** $\mathcal{P}(C_i) = M_i$ $(i \in \{1, 2, \ldots, L\})$ **then**
15: $t \leftarrow t_1 - t_0$
16: 並列数 P，署名の個数 L のときの実行時間として t を出力する．
17: **else**
18: エラーを出力して終了する．
19: **end if**
20: **end for**

JavaScript 言語及び Web Workers を用いて実装する．いま，$\boldsymbol{M} \in \mathbb{K}^m$ を署名対象のメッセージとし，$\boldsymbol{c} \in \mathbb{K}^n$ を $\boldsymbol{M}$ に対する署名とする．Rainbow 署名の署名検証は，署名者の公開鍵 $\mathcal{P}$ に $\boldsymbol{c}$ を代入し，$\mathcal{P}(\boldsymbol{c})$ が $\boldsymbol{M}$ と一致するかどうかを調べることにより行う．$\mathcal{P}(\boldsymbol{c})$ は，m 個の n 変数多項式 $p_k(\boldsymbol{x})$ $(v_1 + 1 \leq k \leq n)$ へそれぞれ $\boldsymbol{c}$ を代入することにより求められる．$p_k(\boldsymbol{x})$ $(v_1 + 1 \leq k \leq n)$ は互いに干渉しないため，$p_k(\boldsymbol{c})$ $(v_1 + 1 \leq k \leq n)$ を独立に計算することができる．バックグラウンドワーカに $p_k(\boldsymbol{c})$ $(v_1 + 1 \leq k \leq n)$ を均等に割り振り，$\mathcal{P}(\boldsymbol{c})$ の計算をバックグラウンドワーカ上で並列に行う．また，Rainbow 署名の署名検証における $\boldsymbol{M}' = \mathcal{P}(\boldsymbol{c})$ を計算するステップと，$\boldsymbol{M}$ と $\boldsymbol{M}'$ が等しいかを判定するステップのうち，後者の判定に必要な時間は，$\boldsymbol{M}'$ の計算時間と比較して十分に小さい．そのため，Rainbow 署名の署名検証においては，$p = 1$ と考えることができる．したがって，アムダールの法則より $S \leq n$ となり，Rainbow 署名の署名検証における高速化率の上限は，使用する CPU のコア数と等しい．

4 実装結果

本章では，Web Workers を用いた Rainbow 署名の並列計算の実装結果について説明する．

4.1 実行時間

表 4 に，Rainbow 署名の署名生成の実行時間を示す．表 5 に，Rainbow 署名の署名検証の実行時間を

表 4: 各ウェブブラウザにおける Rainbow 署名の署名生成 1 回の実行時間（ミリ秒）

端末	Internet Explorer	Google Chrome	Opera	Firefox
PC	11.19	3.83	3.78	8.56
Nexus 7	N/A	20.21	20.58	31.30

表 5: 各ウェブブラウザにおける Rainbow 署名の署名検証 1 回の実行時間（ミリ秒）

端末	並列数 P	Internet Explorer	Google Chrome	Opera	Firefox
PC	1	12.90	4.39	4.88	6.63
	2	6.49	2.23	2.47	3.36
	3	4.47	1.57	1.74	2.35
	4	3.43	1.20	1.32	1.77
	5	3.21	1.09	1.20	1.62
	6	3.08	1.08	1.20	1.61
	7	3.37	1.20	1.05	1.50
	8	3.19	1.18	1.31	1.75
	9	3.17	1.09	1.43	1.90
	10	2.59	0.89	1.05	1.37
	11	3.21	1.34	1.37	1.94
	12	3.24	1.05	1.17	1.48
	13	2.47	0.88	0.96	1.26
	14	3.88	1.44	1.65	2.02
	15	3.71	1.18	1.37	1.80
Nexus 7	1	N/A	21.40	22.77	35.31
	2	N/A	10.77	11.39	17.87
	3	N/A	7.48	7.95	11.97
	4	N/A	5.64	5.83	8.91
	5	N/A	5.78	6.57	9.35
	6	N/A	6.37	6.30	10.71
	7	N/A	6.54	6.61	9.78
	8	N/A	6.72	7.06	10.28
	9	N/A	7.76	8.16	11.73
	10	N/A	6.18	6.41	9.17
	11	N/A	7.88	8.56	9.51
	12	N/A	6.76	7.13	9.95
	13	N/A	5.69	5.91	10.88
	14	N/A	8.65	9.11	11.53
	15	N/A	7.87	8.22	11.85

示す．表 5 に示す実行時間は，アルゴリズム 1 を用いて計測した．アルゴリズム 1 において，$N = 15$, $L = 1000$ として Rainbow 署名の署名検証の実行時間を計測し，その結果を基に，1 回の署名検証に必要な時間を算出した．表 6 に，各ウェブブラウザにおける Rainbow 署名の署名検証の高速化率及び実行効率を示す．

表 5 に示す結果より，Rainbow 署名の署名検証は，PC 上においては並列数 13 のとき，Nexus 7 上においては並列数 4 のとき最も高速に動作することが分かる．特に，PC 上の Google Chrome 及び Opera を使った場合は，1 回の署名検証を 1 ミリ秒未満で完了している．Nexus 7 については，最も高速に署名検証を行うことができたウェブブラウザは Google Chrome であり，その実行時間は 5.64 ミリ秒であった．表 6 に示す高速化率及び実行効率に着目すると，PC 上においては，並列数 13 のとき，およそ 5 倍程度の高速化が行えており，Nexus 7 上においては，並列数 4 のとき，およそ 4 倍程度の高速化が行えていることが分かる．どちらの場合においても，高速化率が最も大きかったウェブブラウザは Firefox であった．特に，Nexus 7 上の Firefox を使った場合の高速化率は 3.96 であり，これは高速化率の上限である 4.00 に極めて近い値である．このときの実行効率は，99.0 %であった．Nexus 7 に搭載されている CPU は 4 コア構成であるから，Nexus 7 については高速化率が最大になるときの並列数と，使用する CPU のコア数が一致する結果となった．

全体を通して見ると，PC 上と Nexus 7 上の両方において，Google Chrome と Opera が同様の挙動を示していることが分かる．これは，両ウェブブラウザの使用する JavaScript エンジンが，同じ V8 であるからだと考えられる．また，Rainbow 署名の署名生成及び署名検証の実行時間は，Google Chrome，Opera，Firefox，Internet Explorer の順に高速であることが分かる．Firefox 及び Internet Explorer に搭載されている JavaScript エンジンの SpiderMonkey と Chakra は，JavaScript コードをバイトコードへ変換しインタプリタ方式で実行する．インタプリタ方式で実行したのちに，実行状況のプロファイリングを行い，必要であればバイトコードをコンパイルし，ネイティブコードを生成する．それに比べて，Google Chrome 及び Opera に搭載されている JavaScript エンジンの V8 は，JavaScript コードを最初の実行時

表 6: 各ウェブブラウザにおける Rainbow 署名の署名検証の高速化率 S 及び実行効率 E

端末	並列数 P	Internet Explorer		Google Chrome		Opera		Firefox	
		S	E	S	E	S	E	S	E
PC	2	1.99	0.332	1.97	0.328	1.98	0.330	1.97	0.328
	3	2.89	0.482	2.80	0.467	2.80	0.467	2.82	0.470
	4	3.76	0.627	3.66	0.610	3.70	0.617	3.75	0.625
	5	4.02	0.670	4.03	0.672	4.07	0.678	4.09	0.682
	6	4.19	0.698	4.06	0.677	4.07	0.678	4.12	0.687
	7	3.83	0.638	3.66	0.610	4.65	0.775	4.42	0.737
	8	4.04	0.673	3.72	0.620	3.73	0.622	3.79	0.632
	9	4.07	0.678	4.03	0.672	3.41	0.568	3.49	0.582
	10	4.98	0.830	4.93	0.822	4.65	0.775	4.84	0.807
	11	4.02	0.670	3.28	0.547	3.56	0.593	3.42	0.570
	12	3.98	0.663	4.18	0.697	4.17	0.695	4.48	0.747
	13	5.22	0.870	4.99	0.832	5.08	0.847	5.26	0.877
	14	3.32	0.553	3.05	0.508	2.96	0.493	3.28	0.547
	15	3.48	0.580	3.72	0.620	3.56	0.593	3.68	0.613
Nexus 7	2	N/A	N/A	1.99	0.498	2.00	0.500	1.98	0.495
	3	N/A	N/A	2.86	0.715	2.86	0.715	2.95	0.738
	4	N/A	N/A	3.79	0.948	3.91	0.978	3.96	0.990
	5	N/A	N/A	3.70	0.925	3.47	0.868	3.78	0.945
	6	N/A	N/A	3.36	0.840	3.61	0.903	3.30	0.825
	7	N/A	N/A	3.27	0.818	3.44	0.860	3.61	0.903
	8	N/A	N/A	3.18	0.795	3.23	0.808	3.43	0.858
	9	N/A	N/A	2.76	0.690	2.79	0.698	3.01	0.753
	10	N/A	N/A	3.46	0.865	3.55	0.888	3.85	0.963
	11	N/A	N/A	2.72	0.680	2.66	0.665	3.71	0.928
	12	N/A	N/A	3.17	0.793	3.19	0.798	3.55	0.888
	13	N/A	N/A	3.76	0.940	3.85	0.963	3.25	0.813
	14	N/A	N/A	2.47	0.618	2.50	0.625	3.06	0.765
	15	N/A	N/A	2.72	0.680	2.77	0.693	2.98	0.745

表 7: ネイティブ実装した Rainbow 署名の実行時間（ミリ秒）

実装方法	PC		Nexus 7	
	署名生成	署名検証[1]	署名生成	署名検証[1]
ネイティブ	1.93	2.14	19.10	17.06
JavaScript	3.78[2]	4.39[3]	20.21[2]	21.40[3]

[1] 並列化せずに行った場合の実行時間である．
[2] Opera 及び Google Chrome を用いた場合の実行時間である（表 4）．
[3] Google Chrome を用いた場合の実行時間である（表 5）．

にコンパイルし，バイトコードへ変換することなくネイティブコードを生成する．したがって，Google Chrome 及び Opera を用いた場合の実行速度が最も高速になったと考えられる．

4.2 ネイティブ実装との比較

表 7 に，Rainbow 署名をネイティブ実装した場合の実行時間を示す．表 7 に示す実行時間は，Rainbow 署名の署名生成及び署名検証を並列化せずに行った場合の実行時間である．ネイティブ実装には，C++言語を用いた．C++コードのコンパイルには，Microsoft Visual Studio Express 2012 for Windows Desktop 及び Android NDK, Revision 9 に含まれるコンパイラを使用した．表 7 に示す全ての場合において，ネイティブ実装した Rainbow 署名の方が，JavaScript 実装の Rainbow 署名より高速であった．

5 応用例

JavaScript を用いた Rainbow 署名の応用例について述べる．JavaScript を用いてウェブブラウザ上に実装した Rainbow 署名は，他のディジタル署名と同様に，メッセージの送信者の認証と，メッセージの改ざん検知機能を提供する．

まず，インターネット選挙運動に関する応用例を挙げる．2013 年 4 月 26 日に公職選挙法の一部を改正する法律が公布され，2013 年 5 月 26 日に施行された．公職選挙法の改正は，インターネット等を利用する方法による選挙運動を解禁することを目的として行われた．インターネット選挙運動解禁に伴う改正公職選挙法第 142 条の 3 第 1 項により，ウェブサイト等を利用する方法による選挙運動用文書図画の頒布が解禁された．これにより，何人も，ウェブサイト等を利用する方法により，選挙運動を行うことが可能となった．ここではウェブサイトを利用して政見放送を行うことを考える．この際，政見放送の送信者を認証し，政見放送の内容が改ざんされていないことを保証する必要がある．政見放送の送信者は，映像に対し，Rainbow 署名を用いて署名を付加する．映像と署名を同時に送信することにより，政見放送の閲覧者が使用するウェブブラウザ上で署名検証を行うことができ，送信者の詐称や内容の改ざんを検知することが可能となる．

次に，Web Workers を用いて Rainbow 署名の署

表 8: 各ウェブブラウザにおける 1 秒間に実行可能な Rainbow 署名の署名検証の回数

ウェブブラウザ	PC			Nexus 7		
	非並列	並列		非並列	並列	
Internet Explorer	77	404	$(P = 13)$	N/A	N/A	
Google Chrome	227	1136	$(P = 13)$	46	177	$(P = 4)$
Opera	204	1041	$(P = 13)$	43	171	$(P = 4)$
Firefox	150	793	$(P = 13)$	28	112	$(P = 4)$

名検証を並列化したとき特有の応用例を挙げる．表 8 に，各ウェブブラウザにおいて 1 秒間に実行可能な Rainbow 署名の署名検証の回数を示す．ここでは監視カメラの映像をウェブブラウザ上でリアルタイムに閲覧できるウェブアプリケーションを考える．一般に，監視カメラは，毎秒 15 フレーム，30 フレーム，60 フレーム等の FPS (Frames Per Second) で映像を撮影する．監視カメラは，撮影した映像の各フレームに対し，Rainbow 署名を用いて署名を付加し，フレームと署名を利用者へ送信する．この場合，利用者のウェブブラウザは，監視カメラの FPS と同じ回数だけ 1 秒間に署名検証を行う必要がある．しかし，表 8 に示す通り，Nexus 7 において，署名検証の並列計算を行わない場合は，Google Chrome 及び Opera を用いたときで毎秒 40 回程度，Firefox を用いたときは毎秒 28 回しか署名検証を行うことができない．これではウェブアプリケーション上で監視カメラの映像をリアルタイムに閲覧することはできない．Web Workers を用いて署名検証を並列に行う場合は，Google Chroem 及び Opera を用いたときで毎秒 170 回程度，Firefox を用いたときにおいても毎秒 112 回の署名検証を行うことができる．したがって，ウェブアプリケーション上においても，監視カメラの映像をリアルタイムに閲覧することが可能となる．

6 まとめ

本稿では，JavaScript 言語を用いて Rainbow 署名をウェブブラウザ上に実装した．Rainbow 署名の署名生成においては，File API を用いることにより，JavaScript プログラムからユーザのローカルストレージ上に保存されている秘密鍵へアクセスすることができるようになった．また，JavaScript 上で並列計算を行うための新しい規格である Web Workers を用いて，Rainbow 署名の署名検証を並列実装し，ウェブブラウザ上で並列計算を行う際の実行時間を計測した．計測は，汎用 PC 及び Nexus 7 上で行った．汎用 PC は 6 つのコアで構成される CPU を搭載し，Nexus 7 は 4 つのコアで構成される CPU を搭載する．計測した実行時間を基に，Rainbow 署名の署名検証についての高速化率及び実行効率を算出した．その結果，汎用 PC 上においては，Rainbow 署名の署名検証は並列数が 13 のとき最も高速に動作し，そのときの高速化率は最大で 5.26 であった．Nexus 7 上においては，並列数が 4 のとき最も高速に動作し，そのときの高速化率は最大で 3.96 であった．

参考文献

[1] HTML5, http://www.w3.org/TR/html5/.

[2] Web Workers, http://www.w3.org/TR/workers/.

[3] Indexed Database API, http://www.w3.org/TR/IndexedDB/.

[4] Web Storage, http://www.w3.org/TR/webstorage/.

[5] File API, http://www.w3.org/TR/file-upload/.

[6] The WebSocket API, http://www.w3.org/TR/websockets/.

[7] Peter W. Shor, “Algorithms for Quantum Computation: Discrete Logarithms and Factoring”, Proceedings, 35th Annual Symposium on Foundations of Computer Science, pp. 124–134, 1994.

[8] Michael R. Garey and David S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W. H. Freeman and Company, 1979.

[9] Tsutomu Matsumoto and Hideki Imai, “Public Quadratic Polynomial-Tuples for Efficient Signature-Verification and Message-Encryption”, EUROCRYPT 1988, LNCS, Vol. 330, pp. 419–453, 1988.

[10] Jacques Patarin, “Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms”, EUROCRYPT 1996, LNCS, Vol. 1070, pp. 33–48, 1996.

[11] Aviad Kipnis, Jacques Patarin, and Louis Goubin, “Unbalanced Oil and Vinegar Signature Schemes”, EUROCRYPT 1999, LNCS, Vol. 1592, pp. 206–222, 1999.

[12] Jintai Ding and Dieter Schmidt, “Rainbow, a New Multivariable Polynomial Signature Scheme”, ACNS 2005, LNCS, Vol. 3531, pp. 164–175, 2005.

[13] 小林透, 瀬古俊一, 川添雄彦, 篠原弘道, “HTML5 によるマルチスクリーン型次世代 Web サービス開発”, 翔泳社, 2013.

[14] Takanori Yasuda, Kouichi Sakurai, and Tsuyoshi Takagi, “Reducing the Key Size of Rainbow Using Non-commutative Rings”, CT-RSA 2012, LNCS, Vol. 7178, pp. 68–83, 2012.

[15] Albrecht Petzoldt, Stanislav Bulygin, and Johannes Buchmann, “Selecting Parameters for the Rainbow Signature Scheme”, PQCrypto 2010, LNCS, Vol. 6061, pp. 218–240, 2010.

[16] Clay Breshears, 千住治郎, “並行コンピューティング技法”, オライリー・ジャパン, 2009.

[17] Standard ECMA-262, http://www.ecma-international.org/publications/standards/Ecma-262.htm.