社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－CSEC－42（7）
2008/7/24

木構造を用いた墨塗り署名方式の効率評価

宮崎 邦彦† 秦野 康生† 本多 義則†

†(株) 日立製作所 システム開発研究所
244-0817 横浜市戸塚区吉田町 292

あらまし 署名後の文書を一部削除（墨塗り）しても，元の署名に基づく真正性が確認可能な，墨塗り署名方式が数多く提案されている．2008年3月には，Haberらによって，木構造を用いて署名サイズを削減可能した墨塗り署名方式が発表された．本稿では，Haberらの提案した署名方式の効率を，理論的および実験的に評価した結果を報告する．
キーワード 電子署名，墨塗り署名，木構造

Efficiency of a Redactable Signature with Tree Construction

Kunihiko MIYAZAKI†, Yasuo HATANO†, and Yoshinori HONDA†

† Systems Development Laboratory, Hitachi Ltd.
292 Yoshida-cho Totsuka-ku, Yokohama-shi, Kanagawa, 244-0817 Japan

Abstract A digital signature does not allow any alteration of the document to which it is attached. Appropriate alteration of some signed documents, however, should be allowed because there are security requirements other than the integrity of the document. Many digital signature schemes, called sanitizable signature or redactable signature, have been proposed for several years as a solution of the problem and Haber et al. proposed one of them on March 2008. Their scheme reduces signature size with tree construction. This article reports theoretical and experimental results about efficiency of the redactable signature scheme with tree construction by Haber et al.
Key words digital signature, redactable signature, tree structure

1. はじめに

電子文書の真正性を保証する技術のひとつとして，電子署名技術が広く用いられている．一般的な電子署名技術を利用した場合，署名付与後の文書を1ビットでも変更すると，署名検証時に，改ざんとして検知される．多くの利用場面において，この性質は望ましいものと考えられているが，場合によっては，かえって問題を引き起こす可能性もある．たとえば，署名が付与された文書に，個人情報等の開示したくない／すべきでない情報が含まれている場合である．個人情報が記載された箇所を削除（墨塗り）するためには，文書を変更しなければならないが，一般的な電子署名技術では，それは改ざんと同様に見なされ，真正性が確認できなくなる．いいかえると，個人情報に関わる箇所「以外」の真正性を示すためであっても，個人情報に関わる箇所も含めて，文書すべてを開示しなければならなくなる．この問題を解決するために，署名後の文書を一部削除（墨塗り）しても，元の署名に基づく真正性が確認可能な，墨塗り署名方式として，Steinfeldらによる Content Extraction Signature [1] をはじめとして，数多くの方式が提案されている．

墨塗り署名方式の一つとして，開示制御可能という特徴を持つ墨塗り署名方式 (SUMI5) が宮崎らによって提案されている [2]．これは，署名付き文書を墨塗りする際に，文書中の各部分の開示条件を，(a) 墨塗りする，(b) 今は墨塗りしないが将来墨塗りできるようにする，または，(c) 今は墨塗りしないし将来的にも墨塗りできないようにするか，のいずれかを設定できる方式である．同様に開示制御可能という特徴を持つ墨塗り署名方式としては，宮崎らによる不可視墨塗り署名方式 [3], [4] や，上山らによる RSA Accumulator を用いた墨塗り署名方式 [5] が知られている．

2008年3月，Haberらによって，開示制御可能な墨塗り署名の一方式が発表された [6]．この方式は，SUMI5 をベースに，木構造を用いて署名サイズを削減した方式である (以下，SUMI5-tree と呼ぶ)．そこで本稿では，SUMI5-tree の効率を，理論的および実験的に評価した結果を報告する．

本稿の構成は，以下の通りである．第2.節において，SUMI5-tree の概略について説明し，第3.節において，理論的な効率の評価結果を示す．また，第4.節において，実験的な評価結果を示す．第5.節において，これら評価結果に基づき SUMI5-tree について考察した後，第6.でまとめを述べる．

2. SUMI5-tree

2.1 アイデア

SUMI5-tree は，SUMI5 をベースに，署名サイズの削減を図った方式である．具体的には，SUMI5 と比較し，次の2つのデータの数を減らしている．

（1） コミット値 (ハッシュ値) のランダム化や開示条件制御のために必要な乱数の個数．

（2） 墨塗り後の署名に含める必要のあるハッシュ値の個数．

SUMI5-tree では，上記 (2) の削減のために，入力長の二倍の長さの出力を与える擬似乱数生成器を用いて，二分木構造に従って，乱数を（根から葉に向かって）順次派生させる．これによって，乱数自体を保持する代わりに，複数の乱数を導出可能なシード値を保持すればよくなるため，個数を削減できる．また上記 (1) の削減のためには，ハッシュ関数を用いて，二分木構造に従って，ハッシュ値を（葉から根に向かって）順次まとめあげていく．これによって，各部分文書のハッシュ値を保持する代わりに，それらをひとまとめにしたハッシュ値を保持すればよくなるため，個数を削減できる．(注1)

2.2 SUMI5 概略

SUMI5-tree のアルゴリズムについて説明する前に，まず元となる SUMI5 のアルゴリズムの概略を説明する．(注2)

2.2.1 署名生成手順

（1） オリジナル文書 M を n 個のブロックに分割する（これを $\{M_0, M_1, \ldots, M_{n-1}\}$ と書く）

（2） 各ブロック M_i に対し，乱数 r_i を生成し，結合したデータ（これを乱数つきブロックと呼び，$R_i = M_i||r_i$ と書く）を生成する

（3） n 個のブロックそれぞれに対し，正当な「墨」をあらわす乱数を生成する（これを墨ブロックと呼び，S_i と書く）

（4） 各ブロック M_i に対し $P_i = h(R_i), Q_i = h(S_i)$ を求める．ここで h はハッシュ関数である．

（5） $P_0||P_1||\cdots||P_{n-1}||Q_0||Q_1||\cdots||Q_{n-1}$ を署名対象データとし，署名者の秘密鍵 sk を用いて，署名 $SIGN$ を生成する．

（6） 生成された署名 $SIGN$，乱数つきブロック $\{R_i\}_{i=0}^{n-1}$，墨ブロック $\{S_i\}_{i=0}^{n-1}$ からなるデータを署名付きオリジナル文書とする．

2.2.2 墨塗り手順

（1） 署名付きオリジナル文書（または他の墨塗り者によってすでに墨塗られた開示文書）に含まれる各ブロック M_i について，「非開示」「開示かつ追加墨塗り許可」「開示かつ追加墨塗り禁止」のいずれかを選択する(注3)．以降，$C = \{i|M_i$は非開示 $\}$，$DA = \{i|M_i$は開示かつ追加墨塗り許可 $\}$，$DB = \{i|M_i$ は開示かつ追加墨塗り禁止 $\}$ と書く

（2） 署名 $SIGN$，乱数つきブロック $\{R_i\}_{i=0}^{n-1} \setminus \{R_i\}_{i\in C}$，乱数つきブロックのハッシュ値 $\{P_i\}_{i\in C}$，墨ブロック $\{S_i\}_{i=0}^{n-1} \setminus \{S_i\}_{i\in DB}$，墨ブロックのハッシュ値 $\{Q_i\}_{i\in DB}$ からなるデータを開示文書とする．

2.2.3 検証手順

（1） すべての i $(0 \leq i \leq n-1)$ に対し，乱数つきブロック R_i または墨ブロック S_i の少なくとも一方が，開示文書に含まれることを確認する．どちらも含まれない場合は，不正な開示文書として却下する．

（2） $0 \leq i \leq n-1$ に対し，乱数つきブロック R_i が開示文書に含まれるときは，$P_i' = h(R_i)$ を求める．また含まれないときは，乱数つきブロックのハッシュ値 P_i が，開示文書に含まれることを確認し，これを P_i' とする．乱数つきブロックと，そのハッシュ値のどちらも含まれない場合は，不正な開示文書として却下する．

（3） $0 \leq i \leq n-1$ に対し，墨ブロック S_i が開示文書に含まれるときは，$Q_i' = h(S_i)$ を求める．また含まれないときは，墨ブロックのハッシュ値 Q_i が，開示文書に含まれることを確認し，これを Q_i' とする．墨ブロックと，そのハッシュ値のどちらも含まれない場合は，不正な開示文書として却下する．

（4） $P_0'||P_1'||\cdots||P_{n-1}'||Q_0'||Q_1'||\cdots||Q_{n-1}'$ を署名検証対象データとし，署名者の公開鍵 pk を用いて，署名 $SIGN$ を検証する

2.3 SUMI5-tree 概略

第 2.1 節で述べたように，SUMI5-tree は，SUMI5 における，乱数 r_i と墨データ S_i，および乱数つきブロックのハッシュ値 P_i と墨データのハッシュ値 Q_i を削減した方式である．以下，簡単のため，$n = 2^L$ と仮定する．

二分木のあるノードを $X_{\{b\}}$ としたとき，その2つの子ノードを $X_{\{b0\}}, X_{\{b1\}}$ と書く．ここで $\{b\}$ は 0,1 からなるビット列であり，$\{b0\}, \{b1\}$ は，それぞれ $\{b\}$ に 0 または 1 を連結したビット列を表す．また特にビット列 $\{b\}$ の長さが L ビットのときは，$\{b\}$ を二進表記と見なした数値と同一視することがある．すなわち $\{00\ldots0\}$ を 0 と見なし，$\{11\ldots1\}$ を $n-1(=2^L-1)$ と見なす．

あるノードの集合 $Z_1 = \{X_{\{b_1\}}, X_{\{b_2\}}, \ldots, X_{\{b_k\}}\}$ に含まれるすべてのノードが，他のノードの集合 $Z_2 = \{Y_{\{b_1\}}, Y_{\{b_2\}}, \ldots, Y_{\{b_l\}}\}$ のいずれかのノードの子孫になり，かつ，Z_2 に含まれるノードの子孫は，必ず Z_1 のいずれかのノードの子孫または先祖となっているときに，Z_2 を Z_1 の被覆と呼び，Z_2 は，Z_1 を覆う，または被覆する，という．また Z_1 を覆うノード集合 Z のうち濃度が最小のものを Z_1 の最小被覆と呼ぶ．なお，二分木においては，与えられたノード集合に対し，その最小被覆は必ず一意に決まり，またそれを具体的に求めることも容易である．

SUMI5-tree では，二分木に入力長の二倍の長さの出力を与える擬似乱数生成器を用いて，二分木構造に従って，乱数を根から葉に向かって順次派生させる．以下，$PRNG()$ を，k ビットの値を入力すると $2k$ ビットの出力する擬似乱数生成器とし，$X_{\{b\}}$

(注1)：これらのデータ削減手法は，Johnson らの redactable signature でも用いられている [7]．

(注2)：なお，ここで述べる方式は [2] などの記述と異なるが，本質的に変わらない．

(注3)：M_i は乱数つきブロック R_i に含まれている

を入力としたときの出力を $X_{\{b0\}}||X_{\{b1\}} = PRNG(X_{\{b\}})$ と書くことがある．なお上述の通り，$\{b\}$ を数値と同一視することがある．

さらに SUMI5-tree では，ハッシュ関数を用いて，二分木構造に従って，ハッシュ値を葉から根に向かって順次まとめあげていく．以下，$hash()$ を，$2k$ ビット入力で，k ビット出力のハッシュ関数とし，$Y_{\{b0\}}, Y_{\{b1\}}$ を入力としたときの出力を $Y_{\{b\}} = hash(Y_{\{b0\}}||Y_{\{b1\}})$ と書くことがある．また $\{b\}$ を数値と同一視することがあるのは，上述の通りである．なお，ハッシュ値は，葉から根に向かって生成されるが，この場合であっても，葉に近い方を子孫，根に近い方の先祖と呼び，被覆の定義もこの関係に従うものとする．

2.3.1 署名生成手順

（1） オリジナル文書 M を $n(=2^L)$ 個のブロックに分割する（これを $\{M_0, M_1, \ldots, M_{n-1}\}$ と書く）

（2） 乱数シード $Seed$ を，乱数として生成する．

（3） $Seed$ から，$r||S = PRNG(Seed)$ を算出する．

（4） r から，$r_{\{0\}}||r_{\{1\}} = PRNG(r)$ を算出し，さらに $r_{\{0\}}, r_{\{1\}}$ から，$r_{\{00\}}||r_{\{01\}} = PRNG(r_{\{0\}})$，$r_{\{10\}}||r_{\{11\}} = PRNG(r_{\{1\}})$ を算出する．以降，これを L 回繰り返し，$r_0, r_1, \ldots, r_{n-1}$ を算出する．

（5） 同様にして，S から，$PRNG$ を用い，n 個のブロックそれぞれに対する正当な「墨」をあらわす乱数である $S_0, S_1, \ldots, S_{n-1}$ を算出する．（これを墨ブロックと呼ぶ）

（6） 各ブロック M_i に対し，生成された乱数 r_i を結合したデータ（これを乱数つきブロックと呼び，$R_i = M_i||r_i$ と書く）を生成する．

（7） 各ブロック M_i に対し $P_i = h(R_i), Q_i = h(S_i)$ を求める．ここで h はハッシュ関数である．

（8） $P_0, P_1, \cdots, P_{n-1}$ から，$hash$ を繰り返し適用して P を算出する．また $Q_0, Q_1, \cdots, Q_{n-1}$ から，同様に $hash$ を繰り返し適用して Q を算出する．$P||Q$ を署名対象データとし，署名者の秘密鍵 sk を用いて，署名 $SIGN$ を生成する．

（9） 生成された署名 $SIGN$，乱数シード $Seed$，オリジナル文書 M からなるデータを署名付きオリジナル文書とする．

2.3.2 墨塗り手順

（1） 署名付きオリジナル文書（または他の墨塗り者によってすでに墨塗られた開示文書）に含まれる各ブロック M_i について，「非開示」「開示かつ追加墨塗り許可」「開示かつ追加墨塗り禁止」のいずれかを選択する．以降，$C = \{i|M_i$は非開示 $\}$，$DA = \{i|M_i$は開示かつ追加墨塗り許可 $\}$，$DB = \{i|M_i$は開示かつ追加墨塗り禁止 $\}$ と書く

（2） 乱数シード $Seed$ から，乱数集合 $\{r_i\}_{i\in C}$ の最小被覆となる，乱数シード集合 $\{r_{\{b_i\}}\}_{i\in RS}$ と，墨ブロック集合 $\{S_i\}_{i\in DB}$ の最小被覆となる，墨ブロックシード集合 $\{S_{\{b_i\}}\}_{i\in SS}$ とを算出する．

（3） 乱数つきブロック R_i から，乱数つきブロックのハッシュ値集合 $\{h(R_i)\}_{i\in C}$ の最小被覆となる，文書ハッシュ集合 $\{P_{\{b_i\}}\}_{i\in PS}$ を算出する．また，墨ブロック $\{S_i\}$ から，墨ブロックのハッシュ値集合 $\{h(S_i)\}_{i\in DB}$ の最小被覆となる，墨ハッシュ集合 $\{Q_{\{b_i\}}\}_{i\in QS}$ を算出する．

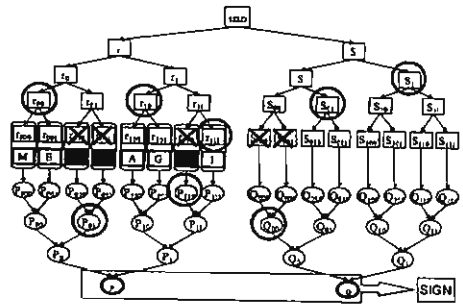

図 1 SUMI5-tree の概要

（4） 署名 $SIGN$，開示するブロック $\{M_i\}_{i=0}^{n-1}\backslash\{R_i\}_{i\in C}$，乱数シード集合 $\{r_{\{b_i\}}\}_{i\in RS}$，墨ブロックシード集合 $\{S_{\{b_i\}}\}_{i\in SS}$，文書ハッシュ集合 $\{P_{\{b_i\}}\}_{i\in PS}$，墨ハッシュ集合 $\{Q_{\{b_i\}}\}_{i\in QS}$ からなるデータを開示文書とする．

2.3.3 検証手順

（1） 開示文書に含まれる，開示するブロック $\{M_i\}_{i=0}^{n-1} \setminus \{R_i\}_{i\in C}$，乱数シード集合 $\{r_{\{b_i\}}\}_{i\in RS}$，墨ブロックシード集合 $\{S_{\{b_i\}}\}_{i\in SS}$ から，$PRNG$ を用いて，算出可能な乱数つきブロック R_i と墨ブロック S_i をすべて算出する．

（2） すべての i $(0 \le i \le n-1)$ に対し，乱数つきブロック R_i または墨ブロック S_i の少なくとも一方が，開示文書に含まれることを確認する．どちらも含まれない場合は，不正な開示文書として却下する．

（3） $0 \le i \le n-1$ に対し，乱数つきブロック R_i が開示文書から算出できるときは，$P'_i = h(R_i)$ を求める．これらと，開示文書に含まれる文書ハッシュ集合 $\{P_{\{b_i\}}\}_{i\in PS}$ とから，$hash$ を繰り返し適用し，P' を算出する．

（4） $0 \le i \le n-1$ に対し，墨ブロック S_i が開示文書から算出できるときは，$Q'_i = h(S_i)$ を算出する．これらと，開示文書に含まれる墨ハッシュ集合 $\{Q_{\{b_i\}}\}_{i\in SS}$ とから，$hash$ を繰り返し適用し，Q' を算出する．

（5） $P'||Q'$ を署名検証対象データとし，署名者の公開鍵 pk を用いて，署名 $SIGN$ を検証する

図 1 は，SUMI5-tree に従って生成された署名を墨塗りしたときに残すノード集合を示したものである．丸で囲んだノードが，墨塗り後に残すノードである．

3. 理論的評価

本節では，SUMI5-tree のデータサイズについて，理論的な評価を行う．以下で，二分木 X とは，葉ノード数が n $(n = 2^L$ はオリジナル文書のブロック数$)$ であるような完全二分木とする．また，二分木 X の根ノードのことも，しばしば X と書く．

3.1 二分木構造の特徴

二分木構造を用いて，データサイズを削減するアイデアは，ワンタイム署名や放送型暗号などで幅広く利用されている．これらの利用場面においては，二分木に含まれる葉ノードに割り当てられるデータは，例えば鍵のように互いに独立したデータ

であるので，どの葉ノードが使われるか（あるいは使われないか）は一様分布に従って選択されることを想定しているケースが多い．

これに対し墨塗り署名においては，葉ノードに割り当てられるデータは，文書ブロックであるので，署名対象文書の性質にも依存するが，墨塗りされる領域はある程度連続すると考えられる．したがって，使われる葉ノードは，いくつかの連続する塊となる．

このような場合の効率評価に関して，次の定理が知られている [6], [8].

[定理 1]（[8] Lemma 2） 二分木 X において，ひとかたまりの連続した葉ノードからなる集合 Z に対し，Z の被覆であって，たかだか $2L-2$ 個のノードからなるものが存在する．

3.2 SUMI5-tree の特徴

まず，SUMI5-tree に従って作成された署名付きオリジナル文書，すなわち署名が付与され，まだどこも墨塗りされていない文書，のサイズを評価する．

上述の署名生成手順に示したとおり，署名付きオリジナル文書は，署名 $SIGN$，乱数シード $Seed$，オリジナル文書 M からなる．すなわち，一般的な署名と比較し，SUMI5-tree の増加分は，乱数シード $Seed$ のみである．たとえば乱数シードを 160 ビット (=20 バイト) とすれば，SUMI5-tree の署名付きオリジナル文書のサイズは，元のオリジナル文書のサイズや，分割ブロック数によらず，20 バイトの増加だけであり，非常に効率的であると言える．

次に，SUMI5-tree に従って作成された開示文書のサイズを評価する．開示文書のサイズは，乱数シード集合，墨ブロックシード集合，文書ハッシュ集合，墨ハッシュ集合の各集合のサイズによってきまる．これらについて，次の定理が成り立つ．

[定理 2] 開示文書に含まれる乱数シード集合を $\{r_{\{b_i\}}\}_{i\in RS}$，文書ハッシュ集合を $\{P_{\{b_i\}}\}_{i\in PS}$ とする．二分木 X において，$\{X_{\{b_i\}}\}_{i\in RS}\cup\{X_{\{b_i\}}\}_{i\in PS}$ は，二分木 X のすべての葉ノードからなる集合に対する被覆となる．

[定理 3] 開示文書に含まれる墨ブロックシード集合を $\{S_{\{b_i\}}\}_{i\in SS}$，墨ハッシュ集合を $\{Q_{\{b_i\}}\}_{i\in QS}$ とする．二分木 X において，$\{X_{\{b_i\}}\}_{i\in SS}\cup\{X_{\{b_i\}}\}_{i\in QS}$ は，二分木 X のすべての葉ノードからなる集合に対する被覆となる．

さらに，墨塗り後の開示文書が，「非開示」または「開示かつ追加墨塗り禁止」のいずれかである場合には，以下の定理が成り立つ．

[定理 4] 開示文書に含まれる乱数シード集合を $\{r_{\{b_i\}}\}_{i\in RS}$，文書ハッシュ集合を $\{P_{\{b_i\}}\}_{i\in PS}$，墨ブロックシード集合を $\{S_{\{b_i\}}\}_{i\in SS}$，墨ハッシュ集合を $\{Q_{\{b_i\}}\}_{i\in QS}$ とする．このとき $RS=QS, PS=SS$ が成り立つ．

証明は，各集合の構成方法から明らかである．

3.3 SUMI5-tree の効率

定理 1,2,3,4 より，SUMI5-tree のデータサイズは，以下のように評価できる．

[定理 5] 開示文書の墨塗り箇所が，いくつかの連続するブロックの，t 個の塊であったとする．このとき，開示文書に含まれる乱数シード集合は，たかだか $(2L-2)t$ 個からなり，文書ハッシュ集合は，たかだか $(2L-2)(t+1)$ 個からなる．さらに，開示箇所がすべて「開示かつ追加墨塗り禁止」に設定されている場合は，開示文書に含まれる墨ブロックシード集合は，たかだか $(2L-2)(t+1)$ 個からなり，墨ハッシュ集合は，たかだか $(2L-2)t$ 個からなる．

したがって，乱数のサイズとハッシュ値のサイズがともに k ビットであるとすると，墨塗り後の開示文書が，「非開示」または「開示かつ追加墨塗り禁止」のいずれかである場合，文書と署名値以外に必要なデータのサイズは，たかだか $(2L-2)(4t+2)k$ ビットである．すなわちデータサイズは，オリジナル文書のブロック数の対数と，連続する墨塗りブロックの塊の個数とに，比例する．(注4)

4. 実験的評価

本節では，SUMI5-tree のデータサイズについて，実験的な評価について述べる．

実験は，自然言語で記述された文書，具体的には電子メール，を対象として行った．またプライバシ保護の観点からの墨塗りを想定し，墨塗り箇所は個人名部分とした．

文書データを抽象化したモデル上で議論を行う理論的評価と異なり，実験的評価においては，以下のような点が，結果に影響を及ぼす可能性がある．

（1） 文書や署名関連データの構造化に関する定義情報

（2） 墨塗り箇所の出現割合，位置

(1) は，理論的な評価においては，文書モデルの抽象化の過程で無視されることが多い項目であるが，実際に墨塗り署名を適用する場合には，署名対象文書を，どのようなブロックの組み合わせとして表現するかに応じて，様々なデータを追加する必要がある．具体的には，ブロックの切れ目を意味するデリミタを挿入したり，あるいは文書の何バイト目から何バイト分を一つのブロックとみなすかを定義する情報を含めたりする必要がある．

(2) は，対象とする文書の種類に応じて決まる項目であり，理論的な評価の際には，具体的な値を決めることが難しい点である．

4.1 実験方法

本稿における実験は，以下のようにして行った．

4.1.1 署名付与

（1） 電子メールデータを入力する

（2） 形態素分析ツールを用い単語単位に分割する

（3） 分析結果を元にブロック分割に関する定義情報を生成する

（4） SUMI5-tree に従い署名を付与する

（5） 署名データ（含む乱数等）を定義情報とともに zip 圧縮して保管する

(注4)：この評価はデータサイズに関する上界を示すものであり，必ずしも最適な評価になっているとは限らない．たとえば，t は最大で $n/2$ となるが，このときの文書と署名値以外に必要なデータサイズは，$2nk$ ビットであり，$(2log(n)-2)(2n+2)k$ よりおよそ $1/(2log(n))$ 倍小さな値になる．

（6） 生成された各種データのサイズを測定する

4.1.2 墨 塗 り

（1） 署名が付与された電子メールデータを表示し，個人名などの固有名詞に該当する箇所を目視により選択する

（2） 選択された箇所を墨塗りする

（3） 墨塗り後の各種データのサイズを測定する

（4） 選択されなかった箇所を「墨塗り禁止」に設定する

（5） 墨塗り禁止処理後の各種データのサイズを測定する

実験にあたっては，上記の一連の処理および測定を実行するツールを開発し，それを用いて行った．開発したツールは，墨塗り手順の (1) 以外の手順はすべて人手を介すことなく自動的に行う．なお，本実験は，データサイズを測定することが目的であるので，署名検証は行っていない．

上記署名付与手順の (3) で生成する定義情報として，本実験では，図 2 のような XML ファイルを用いた．

```
<Regions>
 <Region Id="0">
  <Position offset="0" size="76" />
 </Region>
 <Region Id="1">
  <Position offset="76" size="2" />
 </Region>
 <Region Id="2">
  <Position offset="78" size="15" />
 </Region>
 <Region Id="3">
  <Position offset="93" size="2" />
 </Region>
</Regions>
```

図 2 ブロック分割定義情報

4.2 評 価 項 目

本実験で測定した項目は，下記の通りである．

（1） 入力メールデータのサイズ [byte]

（2） 文書ブロックの数 [個]

（3） 墨塗りされたブロックの数 [個]

（4） 連続する墨塗りブロックの塊の数 [個]

（5） 乱数シードの数 [個]

（6） 乱数つきブロックハッシュ値の数 [個]

（7） 墨ブロックシードの数 [個]

（8） 墨ブロックハッシュ値の数 [個]

（9） 墨塗り署名補助データ（各種シード，ハッシュ値を記載した XML データ）のサイズ [byte]

（10） 全データ（圧縮前）（ブロック分割定義情報，署名値，公開鍵証明書，墨塗り署名補助データを含む XML データ）のサイズ [byte]

（11） 全データ（圧縮後）（ブロック分割定義情報，署名値，公開鍵証明書，墨塗り署名補助データを含む XML データを zip 圧縮したもの）のサイズ [byte]

4.3 実 験 結 果

今回の実験では，50 通のメールデータを対象に測定を行った．本実験で用いたメールデータの特性を，表 1 に示す．

入力メールデータのサイズ [Kbyte]	4.8
文書ブロックの数 [個]	1472.4
墨塗りされたブロックの数 [個]	30.4
連続する墨塗りブロックの塊の数 [個]	14.4

表 1 実験対象データの特性 (平均値)

署名付与直後，墨塗り 1 （「非開示」のみ設定）後，墨塗り 2 （「非開示」「開示かつ追加墨塗り禁止」設定）後の，各種データの平均サイズを，表 2 に示す．なお，署名後の乱数シードと墨ブロックシードは 0 個となっているが，この場合は，これとは別に，乱数シードと墨ブロックシードの共通のシードとなる値が 1 個存在する．

	署名後	墨塗り 1	墨塗り 2
乱数シード [個]	0	63.2	63.2
乱数つきブロックハッシュ [個]	0	20.3	20.3
墨ブロックシード [個]	0	1	20.3
墨ブロックハッシュ [個]	0	0	63.2
墨塗り署名補助データ [Kbyte]	0.119	6.88	11.9
全データ (圧縮前)[Kbyte]	92.8	99.7	104
全データ (圧縮後)[Kbyte]	11.0	13.6	15.5

表 2 SUMI5-tree の各種データサイズ (平均値)

5. 考 察

理論的評価結果から SUMI5-tree は以下の通り，データサイズの点で効率的な方式であると言える．

- 署名後（墨塗り前）のデータは，通常の署名と比較し，乱数シード 1 個分 (例：160 ビット) だけの増加でよい．
- 墨塗り後のデータは，全ブロック数を n，連続する墨塗りブロックの塊の個数を t，乱数のサイズとハッシュ値のサイズを k ビットとすると，たかだか $(2log(n)-2)(4t+2)k$ ビットである．

これらは，従来方式 (SUMI5) が，署名後，墨塗り後のいずれも，$2nk$ ビットのデータが必要であったのに対し，SUMI5-tree は大幅なデータ削減が可能であることを示している．またこのデータ削減は，ハッシュ関数と擬似乱数生成器の追加だけで実現しているため，処理速度の観点からも大きなコストをかけず実現可能であると言える．

一方，電子メールを想定した実験的評価の結果，ブロック分割定義情報や署名値を含む必要な全データのサイズが，署名付与直後で約 11.0Kbyte，墨塗り後で約 15.5Kbyte 程度要することがわかった．これは署名対象としたメールデータの平均サイズが，約 4.8Kbyte であることを考慮すると，かなり大きなサイズであると言える．

このようにサイズが増加した大きな要因としては，今回の実験において，ブロック分割定義情報を XML ファイルとして保持したことが挙げられる．XML ファイルは可読性が高いため，

データをわかりやすく扱える反面，タグを多数記載する必要がある，バイナリデータをテキスト化 (Base64 エンコード) する必要がある，などデータサイズの増加につながる点がある．

文書のブロック分割に関する定義情報を，たとえばバイナリデータによって，よりコンパクトに記述することは可能である．ただし，いずれにしても一般的な電子署名と異なり，文書をブロック分けして処理する墨塗り署名の場合には，ブロック分けを定義するために，なんらかの情報が必要となる点は，実装において注意が必要である．

SUMI5 等の従来の墨塗り署名の場合，各ブロックを可能な限り大きくとり，全体のブロック数を少なくすることは，データサイズ削減の観点から有効であった．しかし SUMI5-tree の場合は，理論的評価に示したとおり，データ削減効果は，対数に比例する程度にとどまる．

このことは，ブロックを大きくとるために定義情報を複雑にするよりは，たとえ総ブロック数が増えるとしても，ブロックをプリミティブな単位に設定することで，定義情報を簡潔にする(注5)ほうが，実装上は効率的になる可能性を示唆している．

6. ま と め

本稿では，Haber らに提案された SUMI5-tree について，署名データサイズの観点から，理論的および実験的に性能評価を行った．

理論的評価によると，SUMI5-tree は，従来方式である SUMI5 と比較して，データサイズを大幅に削減できることがわかった．

一方，実験的評価によると，ブロック分割のための定義情報などの影響で，相当程度大きなデータサイズを必要とすることがわかった．これは，ブロック分割を要する墨塗り署名方式一般に共通する課題であるが，SUMI5-tree の場合は，その理論的特性から，ブロックを単純な単位で細かく分割することが，実装上は効率的になる可能性があることがわかった．

文　献

[1] R. Steinfeld, L. Bull and Y. Zheng: "Content extraction signatures", ICISC, Vol. 2288 of Lecture Notes in Computer Science, Springer-Verlag, pp. 285–304 (2001).

[2] K. Miyazaki, M. Iwamura, T. Matsumoto, R. Sasaki, H. Yoshiura, S. Tezuka and H. Imai: "Digitally signed document sanitizing scheme with disclosure condition control", IEICE Trans. Fundamentals, **E88-A**, 1, pp. 239–246 (2005).

[3] K. Miyazaki, G. Hanaoka and H. Imai: "Digitally signed document sanitizing scheme from bilinear maps", Proceedings of the 2005 Symposium on Cryptography and Information Security (SCIS2005) (2005).

[4] K. Miyazaki, G. Hanaoka and H. Imai: "Invisibly sanitizable digital signature scheme", IEICE Trans. Fundamentals, **E91-A**, 1, pp. 392–402 (2008).

[5] 上山，菊池，伊豆，武仲："RSA accumulator を用いた開示条件付き墨塗り署名方式の改良", Proceedings of the 2008 Symposium on Cryptography and Information Security (SCIS2008) (2008).

[6] S. Haber, Y. Hatano, Y. Honda, W. Horne, K. Miyazaki, T. Sander, S. Tezoku and D. Yao: "Efficient signature schemes supporting redaction, pseudonymization, and data deidentification", ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security, New York, NY, USA, ACM, pp. 353–362 (2008).

[7] R. Johnson, D. Molnar, D. Song and D. Wagner: "Homomorphic signature schemes", CTRSA 2002, Vol. 2271 of Lecture Notes in Computer Science, Springer-Verlag, pp. 244–262 (2002).

[8] B. Pinkas: "Efficient state updates for key management", Proceedings of the IEEE, **92**, 6, pp. 910–917 (2004).

(注5)：たとえば，1 バイト単位にブロックを設定すれば，ブロック分割のための明示的な定義情報を持たなくても先頭からのバイト数で各ブロックを特定できる．