Vol.2022-CSEC-99 No.23
Vol.2022-SPT-49 No.23
Vol.2022-EIP-98 No.23
2022/12/23

# OpenPGP 標準における署名の匿名化

水代 拓也 [1,a)]　北須賀 輝明[1,b)]　今井 勝喜[1,c)]

**概要**：OpenPGP は認証局を必要とせず，信頼できる公開鍵を利用者が管理することで分散的に暗号化や署名検証を行う仕組みである．また，プライバシ保護のため，プライベート MAC アドレスなどのデータリンク層識別子を用いてモバイル端末の追跡可能性を減らす努力が始まっている．そこで，本研究では公開鍵をあらかじめ信頼できる知人等のみに配布するという通常とは異なる仮定のもとで，仮名としてのプライベート MAC アドレスと匿名化した OpenPGP 署名を組み合わせてローカルネットワークに送信することで，知人に対してのみ非匿名になる仕組みを提案する．公開鍵を所持する人は受信した署名を検証することで送信者の個人を特定できる．一方，署名に対応する公開鍵を所持しない人は送信者を特定できない上，同一送信者の異なる仮名を追跡することもできない．

## 1. はじめに

OpenPGP[1]（Open Pretty Goog Privacy）は認証局を必要としない代わりに利用者が公開鍵を管理することで分散的に暗号化や署名の検証を行う仕組みである．また，プライバシの観点からプライベート MAC アドレスと呼ばれるランダム化された MAC アドレスを用いることで，一意の識別子から個人のモバイル端末が追跡される可能性を減らす努力が始まっている．そこで，あらかじめ信頼できる知人等にのみ公開鍵を配布するという仮定の下，仮名としてのプライベート MAC アドレスと匿名化した OpenPGP 署名を組み合わせてローカルネットワークに送信することで，家族や友人・知人に対してのみ非匿名である仕組みを提案する．これにより，公開鍵を渡した信頼できる人にのみ仮名と個人の対応を明かすことができる．具体的には，仮名と署名の受信者は所持する公開鍵を用いて署名の検証を始め，検証成功時に使用した公開鍵の持ち主が仮名の持ち主であると認証することができる．一方で，署名に対応する公開鍵を所持していない者は送信者を特定できない上，署名を匿名化しているため，同一送信者の異なる仮名を追跡することもできない．ここで，署名の匿名化の対象範囲は図 1 での家族，同居人から知人までの関係である．

[1] 広島大学
[a)] m213251@hiroshima-u.ac.jp
[b)] kitasuka@hiroshima-u.ac.jp
[c)] imai@hiroshima-u.ac.jp

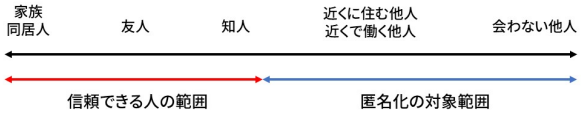


図 1: 匿名化の対象

本研究で使用する技術を説明する．OpenPGP とは分散環境において，PKI などの信頼の基盤を使用することなく鍵を管理し，暗号化・復号，署名を行うことのできる仕組みである．本研究では，OpenPGP の派生である GnuPG（Gnu Privacy Guard）[2] を使用する．プライベート MAC アドレス（仮名）とはスマートフォンが Wi-Fi などに接続する際に使用する，日時によりランダムに変化する MAC アドレスのことである．これらを用いて次のように実装する．

（1） 仮名の配布
モバイル端末は時間で変動する仮名をネットワークを用いて周囲の端末にブロードキャストする．この時，認証の際に用いる署名を添える．

（2） 公開鍵の交換
各端末には信頼している人の公開鍵があらかじめ格納されているとする．公開鍵の交換方法は本稿では定めない．

（3） 署名の検証による認証
ある仮名が誰のものなのか判定する．先ほど述べた仮名のブロードキャスト時に添える署名を利用する．端

Vol.2022-CSEC-99 No.23
Vol.2022-SPT-49 No.23
Vol.2022-EIP-98 No.23
2022/12/23

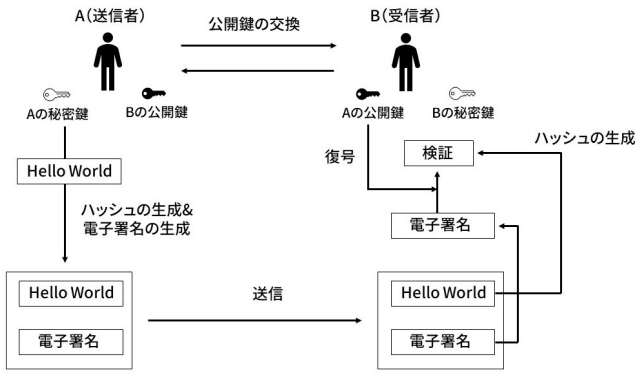


図 2: PGP の通信の流れ [4]

末 A の仮名を端末 B が受信したとする．このとき端末 B は所持する公開鍵の束の中から署名を検証可能な公開鍵を探す．公開鍵が見つかれば，この仮名は公開鍵の持ち主である端末 A のものだと判定する．鍵の管理，署名の生成には OpenPGP 標準のソフトウェアである GnuPG を用いる．

本研究では，以上の三つの仕組みのうち，まずは 3. の署名の検証による認証を行う仕組みを実装する．

## 2. 先行研究

この章では，本研究で用いる OpenPGP[1] 及び GnuPG[2] の仕様について解説する．これら二つは元々 PGP[3]（Pretty Good Privacy）と呼ばれる仕組みを基にしたものである．図 2 に PGP の通信の流れを示す．PGP では認証局のような信頼の基盤を用いず [4]，分散環境でメールやテキストの暗号化，復号化を行う．つまり，受信者と送信者が通信する場合，送信者は受信者の公開鍵を用いてテキストを暗号化して送信する．受信者は自身の秘密鍵を用いて復号してテキストを読む．このように，公開鍵の生成主にのみ復号が可能というわけである．文献 [5] では，PGP のように第三者の介入を必要とせず，自然な形で公開鍵を交換し，暗号化したメッセージを送りあうアプリケーションプラットフォームである「Musubi」が提案されている。PGP において，公開鍵の信頼性に関しては，受け取った公開鍵に信頼する人の署名が付与されているかどうかでその公開鍵を信頼するか決定する「信頼の輪」という方法がとられている．また，公開鍵は事前に交換しておく．この仕組みは盗聴されずにメッセージを交換することを目的として開発されたが，特許権の関係上，広く使われることはなかった．その代わりに OpenPGP が開発される．さらに，その派生がオープンソースソフトウェアの GnuPG である．OpenPGP と GnuPG の主な違いとしてはサポートされるアルゴリズムの違い [6] が挙げられる．ただし，どちらもメッセージフォーマットは OpenPGP 標準に従っている．

メッセージフォーマット [1] について説明する．OpenPGP のメッセージは複数のパケットで構成される．パケットはデータの塊である．メッセージの最初のオクテットはパケットタグと呼ばれ，ヘッダの形式やパケットの内容やパケット長を示す．パケットフォーマットには古い形式と新しい形式があるが，古い PGP バージョンだと古いパケットフォーマットしか対応していないので，それに応じて選択が必要である．また，署名パケットにはバージョンが 2 つあり，バージョン 3 では，基本的な署名情報を提供し，バージョン 4 ではサブパケットという形式でより多くの情報を記述できる．本研究ではバージョン 4 の署名パケットを使用する．バージョン 4 の署名パケットには次のものなどが含まれる．

（1） バージョン番号

（2） 署名の種類
バイナリドキュメントの署名，テキストドキュメントの署名などがある．

（3） 公開鍵方式
RSA 方式や DSA 方式などがある．

（4） ハッシュアルゴリズム
MD5，SHA-1，SHA256，SHA512 などがある．

（5） サブパケットの長さ

（6） サブパケットのデータ
サブパケットには署名作成日時や発行者のキー ID などが含まれる．

本研究では GnuPG を用いる．ここで GnuPG の主な依存ライブラリやパッケージ [2] などの構成を説明していく．

（1） GnuPG
GnuPG はグラフィカルなユーザインターフェースを持たないコマンドラインツールである．コマンドプロンプトや他のプログラムから直接利用可能な暗号エンジンである．本研究では，GnuPG に対して必要な変更を行う．

（2） Libgpg-error
Libgpg-error はすべての GnuPG コンポーネントに共通したエラー値を定義したライブラリである．

（3） Libgcrypt
Libgcrypt は GnuPG のコードをベースにした汎用暗

Vol.2022-CSEC-99 No.23
Vol.2022-SPT-49 No.23
Vol.2022-EIP-98 No.23
2022/12/23

号ライブラリである．対象暗号アルゴリズムは AED，DES などが挙げられ，ハッシュアルゴリズムなどの関数も提供されている．

（4） Libksba
Libksba は X.509 証明書，CMS（Cryptographic Message Syntax）データ，および関連オブジェクトを扱う作業をより容易にするためのライブラリである．X.509 とは ITU-T の公開鍵基盤の規格を指し，CMS は暗号で保護されたメッセージに関する IETF の標準規格である．

（5） Libassuan
Libassuan は GnuPG 間の IPC（InterProcess Communication）通信に用いられるライブラリである．IPC 通信とは，実行中のプログラム間でデータをやり取りする通信のことである．

（6） nPth
nPth はノンプリエンティブなスレッド実装の実現を提供するライブラリである．

## 3. 提案方式

提案方式を説明する．本研究で GnuPG を変更する部分は主に二つある．

（1） 署名の匿名化
一つ目は署名ファイルの匿名化である．OpenPGP 標準では署名ファイルにはバージョン番号，署名タイプ，暗号化アルゴリズム，ハッシュアルゴリズム，タイムスタンプ，キー ID，鍵のフィンガープリントなどが含まれる．キー ID とフィンガープリントは鍵の識別に使われるものである．そのため，キー ID やフィンガープリントが署名に含まれている状態で，仮名と一緒に送信すると，仮名を変更した後でも署名中のキー ID やフィンガープリントによって同一性が追跡できるため，匿名性が保たれない．そこで，これらを取り除く．

（2） 検証方法の変更
二つ目は検証の方法の変更である．変更前では署名の検証時，使用する公開鍵の選択の仕方として，署名のフィンガープリントを読み込み，対応する公開鍵を束の中から選ぶという方法がとられている．しかし，上でも述べたように，フィンガープリントなどを取り除く必要があるため，検証方法を変更する．また，所持するすべての公開鍵で検証する必要があるので，公開鍵を一つずつ選び，署名を検証していき，検証成功時，もしくはすべての公開鍵で失敗したときに終了する手法へと変更する．

```
0000000 89 01 b3 04 00 01 08 00 1d 16 21 04 82 ef fc a1
                                              フィンガー
0000020 1b 8c 8f a0 73 a7 74 de e1 17 7a 7c b3 ed 20 e8
        プリント
0000040 05 02 63 18 31 3e 00 0a 09 10 e1 17 7a 7c b3 ed
                                        キー ID
0000060 20 e8 47 81 0b fc 0a c2 f8 d4 0a 10 4e 43 a6 e2
0000100 ce b5 ab b8 66 c9 2d 2a 56 58 7f 44 7f 53 5d 5c
0000120 e8 6a 7d 21 33 7e 69 86 41 e8 d9 b0 e5 2c b1 41
0000140 41 0a 74 23 aa 68 15 bc 6c e3 61 a4 96 0f 11 0f
0000160 8b b3 50 06 46 3c 8c 9a a1 cc ed fe 3e b4 93 16
0000200 a8 f9 45 9b c1 a9 d2 c9 48 62 6d ff 42 ad 22 ea
0000220 a7 e1 25 6b 48 21 27 26 bd ea ec b6 58 c3 79 7f
0000240 7c 5c 8c 3e 39 43 68 d9 9b 6e 3e c5 11 1f ec c5
0000260 2e 4a cb 77 54 b5 c9 34 52 c6 ca a0 f7 41 79 d6
0000300 a1 58 b4 69 fa 76 d8 f2 ce ba 0d ac e7 d8 e4 1a
0000320 86 66 3f 8b 16 fb a7 c7 af a4 79 02 14 4c 4b 7c
0000340 06 59 7a 09 29 2f 7e 07 36 8b 15 1b 54 68 02 96
0000360 fc 13 32 ea c0 26 48 bc 95 d1 81 9a 6d 47 63 aa
0000400 9c c4 e2 44 74 a9 14 71 94 cf f3 48 5f f9 53 dd
0000420 c6 43 d4 72 44 dc 57 47 2e 6a cf 22 1c 07 92 c3
0000440 21 58 b6 80 b2 6b 00 0a b4 e7 42 b3 43 01 06 f5
0000460 f2 48 d5 07 12 b3 bb 51 12 24 39 9a 7d 69 16 56
0000500 e8 76 cb 51 13 c9 43 32 d4 62 e2 a7 ea d8 cc e8
0000520 8c ee 8e b1 48 54 44 62 97 26 cd 5e 6f c1 01 a5
0000540 5e c4 fa 88 7e 24 71 22 a3 db 6e 79 c0 be f8 07
0000560 ed 1d 65 43 c5 8a 14 fd d8 7d 23 cf f6 3d b6 52
0000600 03 8b 01 90 0d a1 1f 2a 39 6e bf ab 44 c4 2b ad
0000620 16 f1 af dc 99 a4 c6 4e 91 25 cc f1 1a e2 98 ea
0000640 59 6a e7 e4 9d e3 1b 71 84 be bb 8c 44 1e 06 77
0000660 cf 0a f2 4c 7d ed
```

図 3: 署名ファイルのバイナリ表示

これらの変更点を具体的に説明する．

### 3.1 署名の匿名化

署名ファイルは 0 と 1 のビット列により構成されるバイナリファイルであり，OpenPGP 標準により定義されている順番で様々な情報が格納される．また，各種メタ情報が含まれる部分と，ハッシュ値が含まれる部分に分けられる．図 3 に例を示す．また，署名ファイルを pgpdump[7] で解析したものを図 4 （a）に示す．図 3 ではバイナリファイルを 16 進数表示しており，先頭から 13 バイト目の 82 から，32 バイト目の e8 までがフィンガープリントであり，43 バイト目の e1 から，50 バイト目の e8 までがキー ID である．これらの中で，署名ファイルから公開鍵の特定に繋がるキー ID とフィンガープリントを取り除く．

手法としては，作成した署名ファイルから該当部分を削

Vol.2022-CSEC-99 No.23
Vol.2022-SPT-49 No.23
Vol.2022-EIP-98 No.23
2022/12/23

```
Old: Signature Packet(tag 2)(435 bytes)
 Ver 4 - new
 Sig type - Signature of a binary document(0x00).
 Pub alg - RSA Encrypt or Sign(pub 1)
 Hash alg - SHA256(hash 8)
 Hashed Sub: issuer fingerprint(sub 33)(21 bytes)
   v4 -Fingerprint - 82 ef fc a1 1b 8c 8f a0 73 a7 74
   de e1 17 7a 7c b3 ed 20 e8
 Hashed Sub: signature creation time(sub 2)(4 bytes)
   Time - Thu Sep  8 15:00:35 JST 2022
 Sub: issuer key ID(sub 16)(8 bytes)
   Key ID - 0xE1177A7CB3ED20E8
 Hash left 2 bytes - ca a9
 RSA m^d mod n(3071 bits) - ...
   -> PKCS-1
```

（a）標準の署名

```
Old: Signature Packet(tag 2)(402 bytes)
Ver 4 - new
Sig type - Signature of a binary document(0x00).
Pub alg - RSA Encrypt or Sign(pub 1)
Hash alg - SHA256(hash 8)
Hashed Sub: signature creation time(sub 2)(4 bytes)
  Time - Thu Sep  8 14:56:44 JST 2022
Hash left 2 bytes - bd 99
RSA m^d mod n(3070 bits) - ...
  -> PKCS-1
```

（b）提案する匿名の署名

図 4: 署名の例

除する方法と，署名ファイル作成時に該当情報を含めないように変更する方法がある．本研究では後者の方法を選択する．理由としては，署名作成に用いるハッシュ値を計算する際に，署名するデータと上述のメタデータを連結したものを使用するからである．そのため，署名作成後に署名ファイルから該当部分を削除するにはハッシュ値を再計算する必要があるからである．そのままのハッシュ値を用いると，署名の検証時に受信データと署名のメタデータを連結したものから得られるハッシュ値と受け取った署名から得られるハッシュ値が一致しないため正しい署名だとしても，検証に失敗する．これらの理由から署名ファイル作成時に，該当情報を含めないように変更する．よって，署名データ生成時にキー ID 情報とフィンガープリント情報を含めないように変更する．

### 3.2 検証方法の変更

次に検証方法の変更について説明していく．GnuPG における検証の流れは以下である．

(1) 検証側（受信者）は検証データである仮名と対応する署名を受信する．ここで，検証データは暗号化せず，署名とデータは分けて送信される．

(2) 署名ファイルに書かれているフィンガープリントを基に検証に使用する公開鍵を選択する．

(3) 公開鍵を用いて署名から得られるハッシュ値と検証側で計算したハッシュ値を比較し，一致すれば検証成功となる．

以上の流れの中で，2. の公開鍵を選択する際に用いるフィンガープリントを削除することと，本研究ではデータと署名を受信した時点では誰のものかわからないことから，署名の検証が成功するまで公開鍵の束の中から公開鍵を選択し続ける必要があり，そのように変更する．具体的には，公開鍵の一覧の一番上から一つずつ検証していく．途中で検証に成功した場合，検証が成功した旨とユーザ ID を出力し，終了する．すべての公開鍵で検証に失敗した場合，その旨を出力し，終了する．

### 3.3 GnuPG の変更箇所

GnuPG の変更箇所について説明する。

(1) 署名の匿名化
一つ目の署名の匿名化では，署名ファイルの作成時にキー ID とフィンガープリントを含めないようにする．具体的には，gnupg/g10/build-packet.c ファイルの build_sig_subpkt_from_sig 関数内にある署名作成時にキー ID とフィンガープリントを含める部分を実行しないようにする．これにより，署名ファイルの匿名化を行う．

(2) 検証方法の変更
二つ目の検証方法の変更ではキー ID とフィンガープリントの情報がない署名を用いて，所持する公開鍵で検証が成功するまで，検証を繰り返すように変更する．gnupg/g10/mainproc.c 内の check_sig_and_print 関数で署名の検証の実行が始まる．その関数内で do_check_sig 関数が呼ばれており，この関数内で具体的な公開鍵の選択，検証が行われる．このとき，従来の GnuPG の署名検証であれば，署名から抽出したフィンガープリントを用いて公開鍵を選ぶが，本手法ではあらかじめ一つずつ公開鍵を取得しておき，それぞれの公開鍵を do_check_sig 関数の引数として forced_pk に紐づけて実行する．forced_pk に紐付ける公開鍵を一つずつ取得する int 型の関数を get_eachkey として gnupg/g10/getkey.c ファイルに実装する．これを成功の戻り値を取得する

Vol.2022-CSEC-99 No.23
Vol.2022-SPT-49 No.23
Vol.2022-EIP-98 No.23
2022/12/23

```
$gpg --list-keys
--------------------------------
pub   rsa3072 2022-03-17 [SC] [有効期限: 2024-03-16]
      F3AAC37DA1D6B1A155CA0CEEE0899E70C233CBF3
uid           [  究極  ] Keisanki
sub   rsa3072 2022-03-17 [E] [有効期限: 2024-03-16]

pub   rsa3072 2022-05-30 [SC] [有効期限: 2024-05-29]
      3F138A4C6E1C44922C80A249236DCCA40304564C
uid           [  究極  ] raspi
sub   rsa3072 2022-05-30 [E]

pub   rsa3072 2022-07-22 [SC] [有効期限: 2024-07-21]
      0B4D48CE2FE13C59FEC4E158F488BC53A233F478
uid           [  究極  ] Hanako
sub   rsa3072 2022-07-22 [E] [有効期限: 2024-07-21]

pub   rsa3072 2022-07-22 [SC] [有効期限: 2024-07-21]
      7D76059CCDC287ACD1D2CC2777536DC84C096951
uid           [  究極  ] Masako
sub   rsa3072 2022-07-22 [E] [有効期限: 2024-07-21]

pub   rsa3072 2022-08-31 [SC] [有効期限: 2024-08-30]
      82EFFCA11B8C8FA073A774DEE1177A7CB3ED20E8
uid           [  究極  ] Bobby
sub   rsa3072 2022-08-31 [E] [有効期限: 2024-08-30]
                        .
                        .
                        .
```

図 5: 公開鍵のリスト（一部）

まで繰り返す．このように検証方法の変更を行う．

## 4. 評価

提案方式で示した二つの変更点について，正しく変更できているか評価する．今回使用環境として Ubuntu 20.04, GnuPG 2.2.35, Libgpg-error 1.45, Libgcrypt 1.8.9, Libksba 1.6.0, Libassuan 2.5.5, npth 1.6 を使用し，署名の作成から検証まで一つの PC で行う．また，使用する公開鍵の一部を図 5 に示す．ここで，ハッシュアルゴリズムには SHA256 を公開鍵は RSA 方式を採用しており，図には 5 個の鍵しか示していないが，全体で 100 個の鍵を保持している．また，匿名化した署名を変更後の検証方法で検証する場合と元の署名，検証方法での検証時間にどれほど差があるのかを比較する．

（1） 署名の匿名化

署名フォーマットの確認には pgpdump[7] を使用する．MAC アドレスを想定し，6 バイトで表現される ASCII 文字 6 文字が書かれているファイルとそれに対して，Bobby の秘密鍵で作成した署名を使用する．今回作成した署名の例を図 4 に示す．まずは，標準の署名を（a）に示す．変更前なので，フィンガープリントとキー ID が認識され，表示されている．次に提案する匿名の署名を（b）に示す．いくつかのメタ情報の中からフィンガープリントとキー ID のみが削除されていることがわかる．これにより，正しく変更されていると言える．

（2） 検証方法の変更

検証方法の変更を評価する．匿名化した署名ファイルを用いて，検証方法の変更前，変更後の状態で署名の検証を行う．図 6 に変更前の結果を示す．結果として，公開鍵が見つからない旨が表示されている．これは前述の通り，検証に使用する公開鍵を署名ファイルのフィンガープリントの情報により探すため，フィンガープリントを削除しているフォーマット変更後の署名ファイルでは探すことができなかったためである．次に図 7 に検証方法変更後に Hanako と Bobby の秘密鍵の署名を検証した結果を示す．先程とは違い，正しく署名が検証されている．また，公開鍵を選択する回数を数えていき，表示した結果，検証成功時に Hanako の公開鍵が 3 個目，Bobby の公開鍵が 5 個目の公開鍵と表示されている．これにより，公開鍵をリストの上から順番に検証成功するまで取得できていることがわかる．

（3） 検証時間の比較

次に検証時間の比較を行う．具体的には，元の署名ファイルを用いて，元の検証方法にて検証を行う方法（変更前）と匿名化した署名ファイルを用いて，変更後の検証方法で検証を行う方法（変更後）とを比較する．署名するファイルは同様に 6 バイトで表現される ASCII 文字 6 文字が書かれているファイルを使用する．変更後の検証方法では公開鍵の束の一番上の公開鍵から選択して検証するので，上から 1 つ目の公開鍵から 100 個目までの公開鍵で検証成功する場合の平均時間のグラフを図 8 に示す．このグラフは 1 つ目の公開鍵で検証が成功するパターンを 10 回行い，10 回の平均実行時間を求めた後に 2 つ目の公開鍵で検証が成功するパターンを 10 回行うというような動作を 100 個目の公開鍵まで連続して行うというプログラムで作成したものである．また，ディスクキャッシュを考慮し，一回の検証ごとに 5 秒待機するようにする．実行時間の計測には time コマンドで出力される real（プログラムの呼び出しから終了までにかかった実時間）を使用している．結果を説明する．まず，青色の線は変更前の署名ファイルと検証方法で計測したものである．グラフから何個目の公開鍵であっても 0.000 秒から 0.010 秒の間で実行が

Vol.2022-CSEC-99 No.23
Vol.2022-SPT-49 No.23
Vol.2022-EIP-98 No.23
2022/12/23

```
gpg: 2022 年 09 月 08 日 14 時 56 分 44 秒 JST に施された署名
gpg:                RSA 鍵 0000000000000000 を使用
gpg: 署名を検査できません: 公開鍵がありません
```

図 6: 検証方法変更前

```
gpg: 2022 年 11 月 28 日 23 時 45 分 43 秒 JST に施された署名
gpg:                RSA 鍵 0000000000000000 を使用
上から 3 個目の公開鍵で検証に成功しました.
gpg: "Hanako"からの正しい署名 [究極]

gpg: 2022 年 09 月 08 日 14 時 56 分 44 秒 JST に施された署名
gpg:                RSA 鍵 0000000000000000 を使用
上から 5 個目の公開鍵で検証に成功しました.
gpg: "Bobby"からの正しい署名 [究極]
```

図 7: 検証方法変更後

完了していることがわかる．これは前述の通り，署名からフィンガープリントを抽出し，所持する公開鍵と照合することで，即座に署名に対応する公開鍵を選択できるためである．続いて，赤色の線は匿名化した署名ファイルを変更後の検証方法で計測したものである．1 つ目から 10 個目までの公開鍵で検証が成功する場合は 0.005 秒から 0.015 秒の間を維持しているが，10 個目より順番が大きい公開鍵になるとばらつきはあるが，ほぼ公開鍵の検証順位に比例して検証時間が増加している．90 個目から 100 個目の公開鍵で検証成功する場合は約 0.035 秒から 0.050 秒もの時間がかかっており，変更前の方法と比べて大きな時間がかかることとなる．現段階では 100 個の公開鍵で検証しているが，今後この仕組みを活用するに当たり，さらに多くの公開鍵を所持する可能性があるため，効率的な公開鍵の選択方法を実装する必要がある．

## 5. まとめ

本研究では最終的な目的である，信頼できる人にのみ仮名と個人の対応を明かす仕組みの初期段階として，GnuPG の変更を行った．以下に具体的な項目を示す．

(1) 署名の匿名化
署名の匿名化では，署名ファイルのキー ID とフィンガープリントを削除した．これにより，署名の信頼できる人以外への匿名化を行い，仮名が追跡されるのを防ぐ．

(2) 検証方法の変更
検証方法の変更では，匿名化した署名を用いて，所持する公開鍵で検証が成功するまで，検証を繰り返すように変更した．

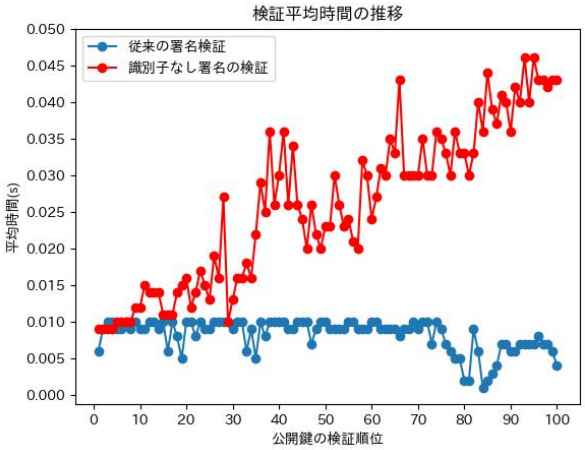


図 8: 平均実行時間

(3) 検証時間の比較
検証時間の比較では，変更前の署名ファイルと検証方法での検証実行時間と匿名化した署名と変更後の検証方法での検証実行時間を比較した．変更後では検証が成功する公開鍵の順番に比例して実行時間が増えていき，順番が後になるほど大きな時間がかかる．

今後の課題としては，今回実装した検証方法の変更において，検証時間削減のための公開鍵選択手法の模索などが挙げられる．また，まだ実装できていない仮名の配布，公開鍵の交換の部分を実装していく予定である．それと並行で，この仕組みをどのように活かすのかを検討する．

**謝辞** 本研究の一部は JSPS 科研費 JP22K11993 の助成を受けたものです．

### 参考文献

[1] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, R. Thayer, “OpenPGP Message Format”, Request for Comments: 4880, November, 2007
[2] “THE GNU PRIVACY GUARD”, https://www.gnupg.org/index.html, (2022/08/30 参照)
[3] Simson Garfinkel, 山本和彦監訳, (株) ユニテック訳, “PGP：暗号メールと電子署名”, オライリー・ジャパン, 1996
[4] “PGP を利用した電子メールの例”, https://www.ipa.go.jp/security/enc/digitalsignature/02_ML.htm, (2022/08/29 参照)
[5] Ben Dodson, Ian Vo, T. J. Purtell, Aemon Cannon, Monica S. Lam, “Musubi: Disintermediated Interactive Social Feeds for Mobile Devices”, WWW 2012： Proceedings of the 21st international conference on World Wide Web, April, 2012

Vol.2022-CSEC-99 No.23
Vol.2022-SPT-49 No.23
Vol.2022-EIP-98 No.23
2022/12/23

[6] “PGP，OpenPGP，GnuPG 暗号化の違い”，July，24，2019，https://www.ipswitch.com/jp/blog/the-difference-between-pgp-openpgp-and-gnupg-encryption（2022/11/30 参照）
[7] 山本和彦，“pgpdump”，https://github.com/kazu-yamamoto/pgpdump，（2022/11/30 参照）