株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

security font の
堅牢性と脆弱性

株式会社フュージョンシス

文書バージョン	日付	変更点
ver. 1.0	2017 年 1 月 27 日	作成

目次

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

1. 本文書の目的

security font の堅牢性と脆弱性を明らかにする。

1.1 堅牢性に関して

security font の堅牢性を定量的に評価する。ここで”堅牢性”とは、必要となる総当たり探索の数と定義する。

1.2 脆弱性に関して

何が脆弱性の原因になるかを説明する。

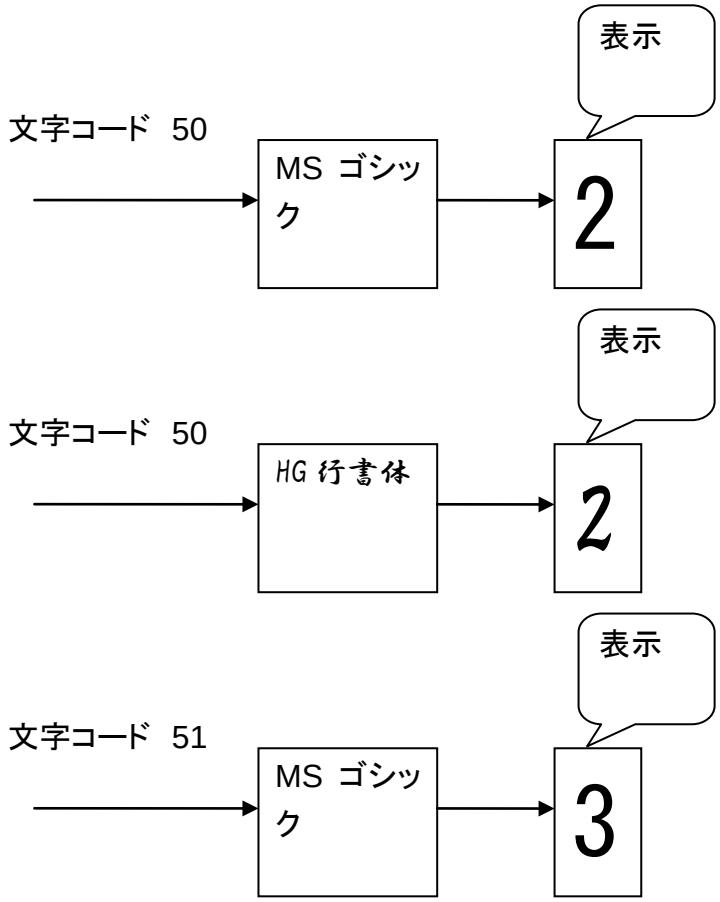
株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

2. security font の仕組み----直感的な説明

2.1 仮想機械

この仮想機械は、文字コード入力部、入力コードと表示を結びつける中間部、表示部からなる。

2.1.1 通常フォントの仮想機械の振る舞い

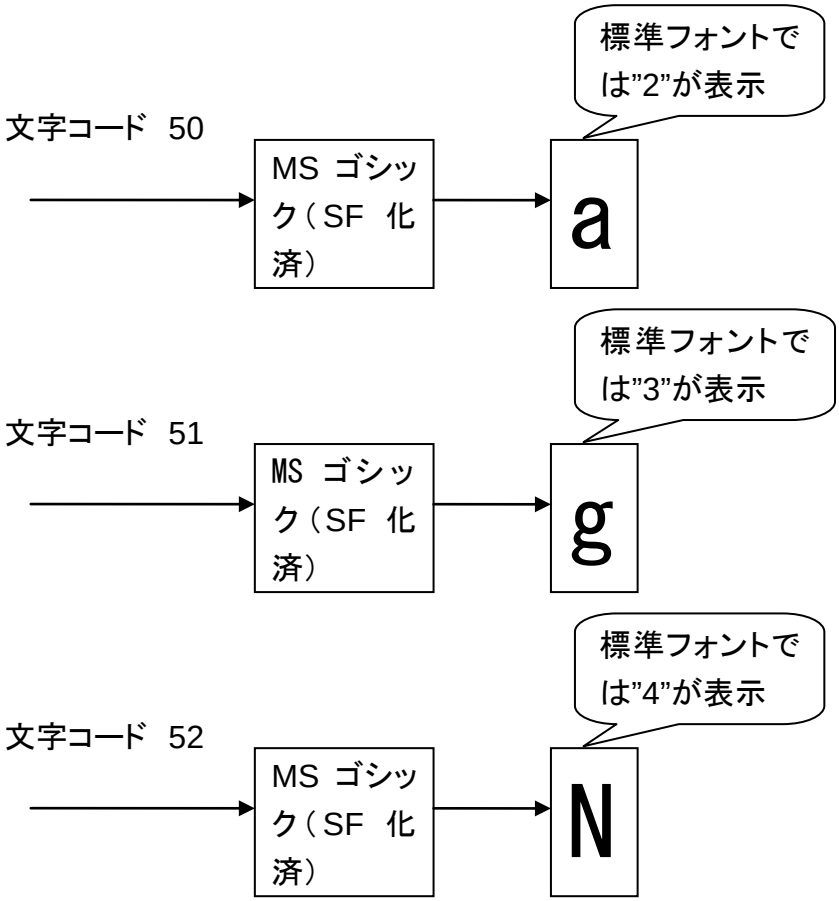


中間部は、通常は変更することはできないが、適当な方法で変更することができる。ttf (true type font)ファイル、それを web ベースにした woff (web open font format)は、この仮想機械をファイル化したものである。

2.1.2 security font の仮想機械の振る舞い

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

security font とは、この仮想機械の”入力コードと表示を結びつける中間部” が通常とは異なるようにしたものである。



2.2 security font 関連の特許など

1) 文字コードの入替方法(特許 5865565)
2) 入替えた文字コードとフォントを関連付けて保存(特許 5956092)
3) フォントにタイムスタンプと GPS 情報等を埋め込む(出願中)

security font の詳細については、特許を参照されてください。

3. 予備知識

3.1 文字コード表

文字コード表--->文字コードと表示される文字との対応関係を表形式にしたもの。

3.2 英語の略語

ttf--->true type font
woff--->web open font format

3.3 数式

8! = 8x7x6x5x4x3x2x1

$${}_nC_p = \frac{n!}{p!\,(n-p)!}$$

高校の数学教科書の”順列・組み合わせ”を、参照されてください。

3.4 マッピング、逆マッピング

マッピング--->文字コードと表示内容との対応関係

表示内容 = f(文字コード)

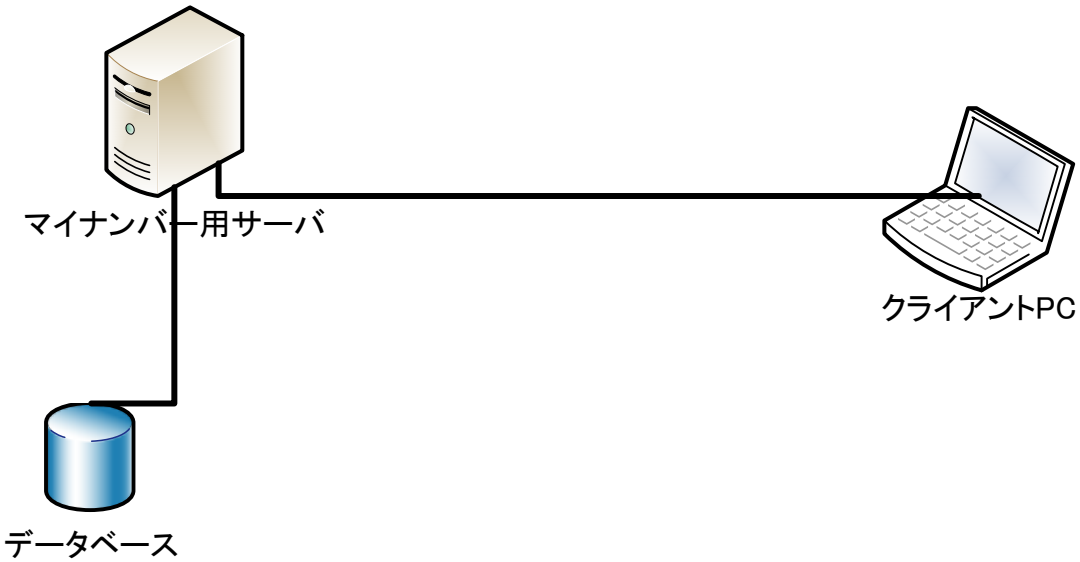
という関数形式に書ける。

逆マッピング--->表示内容と文字コードとの対応関係

122
123 security font は一つの表示に対して、複数の文字コードを対応させる。1 個の表示内容に
124 対して、1 個の文字コードが決まるわけではないので、逆マッピングは、一意に決まらない
125 （従って関数ではない）。
126
127 ### 3.5 security font の encoder と security font の decoder
128
129 security font--->標準的なマッピングとは異なったマッピングをするフォントまたはフォン
130 トファイル。SF と略記する。
131
132 security font の encoder--->逆マッピングを行う仮想機械またはプログラム。SFencoder と略
133 記
134
135 security font の decoder--->マッピングを行う仮想機械またはプログラム。SFdecoder と略記
136

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

4. 想定されるシステム例



- ✓ データベースには SFencoder で隠蔽された文字列が格納されている。
- ✓ クライアント PC からブラウザ（後でこの問題点を指摘する）でマイナンバー用のサーバにアクセスし、閲覧する。

5. security font の実例

5.1 標準的な ascii コード表

数字、アルファベット大文字とアルファベット小文字、特殊文字、若干の非表示のコントロールコードが含まれている。含まれる文字コードは、全部で 128 個である。

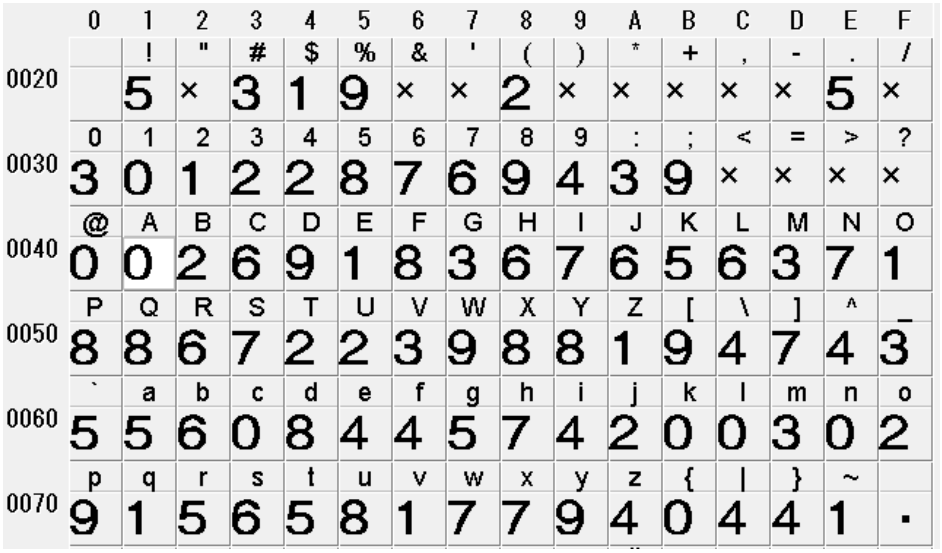
株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進
NUL	0	00	DLE	16	10	SP	32	20	0	48	30	@	64	40	P	80	50	`	96	60	p	112	70
SOH	1	01	DC1	17	11	!	33	21	1	49	31	A	65	41	Q	81	51	a	97	61	q	113	71
STX	2	02	DC2	18	12	"	34	22	2	50	32	B	66	42	R	82	52	b	98	62	r	114	72
ETX	3	03	DC3	19	13	#	35	23	3	51	33	C	67	43	S	83	53	c	99	63	s	115	73
EOT	4	04	DC4	20	14	$	36	24	4	52	34	D	68	44	T	84	54	d	100	64	t	116	74
ENQ	5	05	NAK	21	15	%	37	25	5	53	35	E	69	45	U	85	55	e	101	65	u	117	75
ACK	6	06	SYN	22	16	&	38	26	6	54	36	F	70	46	V	86	56	f	102	66	v	118	76
BEL	7	07	ETB	23	17	'	39	27	7	55	37	G	71	47	W	87	57	g	103	67	w	119	77
BS	8	08	CAN	24	18	(	40	28	8	56	38	H	72	48	X	88	58	h	104	68	x	120	78
HT	9	09	EM	25	19	)	41	29	9	57	39	I	73	49	Y	89	59	i	105	69	y	121	79
LF*	10	0a	SUB	26	1a	*	42	2a	:	58	3a	J	74	4a	Z	90	5a	j	106	6a	z	122	7a
VT	11	0b	ESC	27	1b	+	43	2b	;	59	3b	K	75	4b	[	91	5b	k	107	6b	{	123	7b
FF*	12	0c	FS	28	1c	,	44	2c	<	60	3c	L	76	4c	\¥	92	5c	l	108	6c	|	124	7c
CR	13	0d	GS	29	1d	-	45	2d	=	61	3d	M	77	4d	]	93	5d	m	109	6d	}	125	7d
SO	14	0e	RS	30	1e	.	46	2e	>	62	3e	N	78	4e	^	94	5e	n	110	6e	~	126	7e
SI	15	0f	US	31	1f	/	47	2f	?	63	3f	O	79	4f	_	95	5f	o	111	6f	DEL	127	7f

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

5.2 security font の文字コード表の 1 例

この節の議論は、この文書の最後の参考文献に挙げた [高見 2015] からの引用である。



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0020		5	×	3	1	9	×	×	2	×	×	×	×	×	5	×
	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0030	3	0	1	2	2	8	7	6	9	4	3	9	×	×	×	×
	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0040	0	0	2	6	9	1	8	3	6	7	6	5	6	3	7	1
	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0050	8	8	6	7	2	2	3	9	8	8	1	9	4	7	4	3
	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0060	5	5	6	0	8	4	4	5	7	4	2	0	0	3	0	2
	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
0070	9	1	5	6	5	8	1	7	7	9	4	0	4	4	1	·

例えば、MS ゴシックを使って、”ABCDE”と表示するには、65（16 進で 41）、66（16 進で 42）、67（16 進で 43）、68（16 進で 44）、69（16 進で 45）という文字コードを MS ゴシックという仮想機械に送る（標準の ascii 表を参照）。

一方 MS ゴシック SF（security font 化された MS ゴシック。文字コードと表示との対応に改変がある）では、ABCDE の位置に 02961 が来ているので、MS ゴシックでは”ABCDE”と表示される文字コードの並びに対しては、MS ゴシック SF では

02961

と表示される。

MS ゴシック SF で、”12345”という表示を得たいときは、MS ゴシックでは

Eo#9t

の位置にある 5 個の文字コード、あるいは

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

2jG^K

の位置にある 5 個の文字コードを MS ゴシック SF に送る。

1 個の数字に関して、8 個の文字コードが使われているので、Eo#9t のような表現は、非常に多数存在する。その個数は、この文書中で計算する。

5.3 security font の強み

80 個の文字コードで表示される文字を、標準的な ascii 文字コード表とは異なる表示に変えている。security font の文字コード表の総数は、”0”を 8 個、”1”を 8 個、”2”を 8 個、.... ”9”を 8 個を、1 列に並べる、その並べ方の数と同じになる（10 種類の数字がそれぞれ 8 個ずつある。これら全部で 80 個の数字を、1 列に並べる並べ方の数。高校数学の順列・組み合わせを参照）、

$$\alpha = \frac{80!}{8!8!8!8!8!8!8!8!8!8!}$$

である（128 個の文字コードのうち、非標準的な表示をして良い 80 個の文字コードは、予め決められているとする）。計算を実行すると

$$\alpha = 6.32 \ E+72$$

α は大きな数であるが、全部の文字コード表を書き下すことができる。総当りすれば、実際に使われた SF に行き当たることは可能である。しかし、文字コード表の数は α 個あり、非常に多数である。SF の文字コード表のあまりの多さが、ハッキングに対する防御になる。security font の堅牢性は、もともとの文字を暗号によって隠蔽しているのではなく、その文字を表示する文字コードを文字コード中に配置するその方法が多数存在することである。

この議論は先で更に深化させる。

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

6. 必要となる文字コード表の大きさ

security font の仕様では、表示される文字と、それを表示する文字コードの対応は、1 対 n を許している。8 を使うことが多いので、この文書でも 8 を考える。

6.1 隠蔽する文字が数字だけの場合

10 個の数字の表示に対して、80 個の文字コードが必要になる。ascii 文字コード表は 128 個の文字コードを含んでいるので、security font の元になる文字コード表として、ascii コード表でも十分である。

6.2 隠蔽する文字が数字とアルファベットの場合

数字とアルファベットの場合は、(アルファベット大文字 26＋アルファベット小文字 26＋数字 10) x 8 = 496 個の文字コードが必要となるので、少なくとも 16bit (256x256=65536 個の文字コードを収容可能) 系の文字コード表が必要になる。

MS ゴシックや MS 明朝の文字コードはシフト jis (16bit 系) であり、収容可能な文字コードの数は、65536 個である。従って security font の元になる文字コード表としてシフト jis コード表を使うことができる。

6.3 隠蔽する文字が漢字を含む場合

漢字を扱うには、シフト jis で表示される 9402 個の文字表示に対して 9402x8=75216 個の文字コードが必要となる。従って security font の元になる文字コード表として、シフト jis 文字コード表では不十分で、24bit 長 (256x256x256 = 約 1667 万) または 32bit 長 (256x256x256x256 = 約 42 億) の文字コード表が必要になる。この文書では、32bit 長 (4byte 長) の文字コード表を考える。

7. 表示と文字コードとの対応の種類

7.1 1 個の表示に対して、1 個の文字コードを割り当てる------換字式暗号

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

本文書では 1 個の数字の表示に対して、8 個の文字コードを割り振っているが、1 個の数字の表示に対して、1 個の文字コードを割り振る場合は、”換字式暗号”と呼ばれ、昔から存在している。標準文字コード表で、

12345

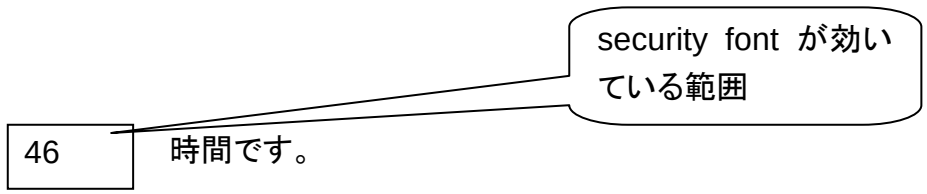
を表示させる文字コードを、例えば換字式の security font で

34567

を表示させるようにすると

1 日は 24 時間です。

という文章が



といったおかしな文章になり、若干の隠蔽効果はある。ただハッカーに対する防御力は 0 である。

7.2 1 個の表示に対して、8 個の文字コードを割り当てる

このことの意義は、隠蔽効果が増すことである。ascii 文字コード内で数字の表示 1 個に対して、8 個の文字コードを用意するだけで

α = 6.32 E＋72

個の security font の文字コード表が存在する。

7.3 1 個の表示に対して、m (m>8)個の文字コードを割り当てる

更に多くの security font の文字コード表が存在する。m > 8 ならば

$$\alpha' = \frac{(10 \times m)!}{m!m!m!m!m!m!m!m!m!m!}$$

のとき

$$\alpha' > \alpha$$

である。もちろん、m には上限がある(10 x m が文字コード表で収容できる文字数を超えることはできない)。

8. AES と security font との比較

8.1 AES (rijndael)

暗号化された文字列から平文を回復するには、大変な計算量を要する。一方鍵があれば、暗号化された文字列の暗号を解除できる。

8.2 security font

隠蔽された文字列から正常な表示を得るためには、かなりの計算量が必要になる(この文書中で計算を行う)。

一方 security font の場合、decoder があれば、隠蔽された文字列を正常に表示することができる。その意味で、SFdecoder は、AES の場合の、鍵がはじめから付随している decoder (暗号処理の文脈では、"decryptor")のようなものと言える。SFdecoder は、ユーザの手に届かない場所に置くことが必須である。

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

324

	encoder	decoder	堅牢性
AES（rijndael）	アルゴリズムは公開されており、誰でもが入手できる。 ”encoder が盗まれる”という概念自体ない。 encoder（encryptor）単体では暗号は解除できない。	アルゴリズムは公開されており、誰でもが入手できる。 ”decoder が盗まれる”という概念自体ない。 decoder（decryptor）単体では暗号は解除できない。	隠蔽された文字列から元の文字列を推測することはほぼ不可能
security font	encoder 単体で正常な表示に到達できる（若干の作業が必要だが） ユーザには入手させてはならない。	decoder 単体で正常な表示ができる。 ユーザには入手させてはならない。	隠蔽された文字列から元の文字列を推測することは、ほぼ不可能。 ただ元の文字列が統計的な偏りを持っている場合には、マッピングの推測を許す可能性がある。

325
326
327
328

9. ハッキング

以下のような 4 通りのハッキングを考える。

	decoder または encoder	暗号鍵
case1	盗まれた	盗まれた
case2	盗まれた	盗まれなかった
case3	盗まれなかった	盗まれた
case4	盗まれなかった	盗まれなかった

9.1 encoder または decoder が盗まれた場合

AES（rijndael）などの暗号は、鍵が盗まれていなければ、暗号化された出力文字列から入力文字列を特定するのは、非常に困難である（暗号の種類によって、その困難さは変わる。一つの特定の暗号方法の強度は、鍵の長さに依存する）。

security font の場合は、暗号鍵は盗まれていなくても、decoder あるいは encoder が盗まれた場合、完全に無防備になる。以下でそれを説明する。

9.1.1 encoder が盗まれた場合

encoder が盗まれた状況を考える。

1111....

という非常に長い文字列を encoder を入れると、1 がどの 8 箇所にマップされているか、ほぼ完全に（8 箇所がすべて出しつくされる保証はない）特定できる。次に

2222....

という非常に長い文字列を入れると、2 がどの 8 箇所にマップされているか、ほぼ完全に（8 箇所がすべて出しつくされる保証はない）特定できる。以下順に

3333....
4444....

と入れていけば、文字コード表の 80 箇所をほぼ完全に特定できる。

9.1.2 decoder が盗まれた場合

非標準的な文字コードの列（MS ゴシックといった標準のフォントで表示すると文字化けする）を、encode 時に使った encoder に対応する decoder に入れれば、正常な表示が得られる。

9.2 encoder または decoder が盗まれた場合の結論

以上の議論から、一番右の列に一部結論を書くことができる。

	decoder または encoder	暗号鍵	初見の SF 化された文字列から、入力文字列を特定可能か？
case1	盗まれた	盗まれた	完全に無防備
case2	盗まれた	盗まれなかった	完全に無防備
case3	盗まれなかった	盗まれた	
case4	盗まれなかった	盗まれなかった	

通常は、case4 の状態で運用される。したがってこの評価が一番重要である。以下では case4 の評価を行う。

10. 堅牢性の定量的な評価を行うにあたって

decoder や encoder が盗まれていない状態で、総当り探索の数を計算する。これは AES が鍵のない状態で総当りにして平文を回復するための計算量を評価するのに相当する。

security font の堅牢性の源泉は、文字列そのものの隠蔽ではなく、同程度に尤もらしい（尤度が同様に高い）文字コード表が、非常に多数存在することである。decoder や encoder や鍵が盗まれていない場合、どれくらい、同程度に尤もらしい（尤度が同様に高い）文字コード表が存在するかを計算する。

1）数字だけを扱う場合
2）数字、アルファベット、漢字、かな、特殊文字を扱う場合

の 2 つの場合を考える。

11. 数字に特化した security font の堅牢性の定量的な評価

11.1 ascii コード表を元にする場合

11.1.1 前提条件

- ✓ 数字だけを扱う。
- ✓ 80 個の数字を流し込む文字コード表内の領域（複数の文字コード）は予め指定されているものとする。

11.1.2 可能な security font 文字コード表の数

”0”を表示する文字コードを 8 個
”1”を表示する文字コードを 8 個
”2”を表示する文字コードを 8 個
....
”9”を表示する文字コードを 8 個

の合計 80 個の文字コードを 1 列に並べる方法の数は、

$$\alpha = \frac{80!}{(8!\,8!\,8!\,8!\,8!\,8!\,8!\,8!\,8!\,8!)}$$

である。計算すると

$$\alpha = 6.32\ \mathrm{E}+72$$

である。security font の文字コード表の中から、実際にエンコードに使われた security font の文字コード表を、偶然に正しく選ぶ確率は、

$$1/\alpha$$

であり、非常に 0 に近い。

11.2 4 バイト長の文字コード表全体から、重複なく 80 箇所を選ぶ場合

4 バイトの文字コード表の中から、

表示”0”に対応する文字コードを 8 個
表示”1”に対応する文字コードを 8 個
....
表示”9”に対応する文字コードを 8 個

を重複なしに選ぶ方法の数は、次のように計算できる。

1 回目の操作　表示”0”に対応する文字コードを、256x256x256x256 の文字コード中から選ぶ方法の数
2 回目の操作　表示”0”に対応する文字コードを、256x256x256x256-1 の文字コード中から選ぶ方法の数
3 回目の操作　表示”0”に対応する文字コードを、256x256x256x256-2 の

文字コード中から選ぶ方法の数

....

8 回目の操作　　表示”0”に対応する文字コードを、256x256x256x256-7 の文字コード中から選ぶ方法の数

9 回目の操作　　表示”1”に対応する文字コードを、256x256x256x256-8 の文字コード中から選ぶ方法の数

10 回目の操作　　表示”1”に対応する文字コードを、256x256x256x256-9 の文字コード中から選ぶ方法の数

....

80 回目の操作　　表示”9”に対応する文字コードを、256x256x256x256-80 の文字コード中から選ぶ方法の数

は

$$\beta = {}_{256\times256\times256\times256}C_{80}$$

である。計算すると

$$\beta = 6.05E+651$$

である。security font の文字コード表を、偶然に正しく選ぶ確率は、

$$1/\beta$$

である。この数は非常に小さい（この数は $1/\alpha$ よりも遥かに小さい）。

11.3 どの文字コードが使われているか完全に分かってしまった場合

マイナンバーを表示する権限のない自治体の職員が、表示用のフォントとして MS ゴシックが指定してあるブラウザを使って、マイナンバーを表示しようとしたとき、

FeZ. . . .

と表示されたとする。この時、80 個の文字コードのうち、10 進で 70、101、90 の文字コードは使われていることが判明する。その職員が毎日コツコツと使われている文字コードを記録していけば、80 個の文字コードすべてが分かることもあり得る。80 個すべての使われている文字コードがわかれば、

$$\frac{80!}{(8!\,8!\,8!\,8!\,8!\,8!\,8!\,8!\,8!\,8!)}$$

まで、可能な文字コード表の数は減少する。これは ascii コード表で、非標準的なマッピングを許す 80 箇所が予め決まっている場合と、同じ数である。

11.4 α と β の関係

α と β の大小関係は

$$\alpha < \beta$$

であるが、使われていることがわかっている文字コードが 80 個に満たない場合は、α と β の間の値を取る。

11.5 数字に特化した SF の堅牢性の最小値

decoder が盗まれない限り、使われている文字コードが判明しても、

$$\alpha$$

を下回ることがないということである。最初は使われている文字コードに関する情報が無くて、堅牢性は β だったが、使われている文字コードが分かっていくと、最終的には、α まで堅牢性が落ちるということである。

12. 漢字にも対応できる SF の堅牢性の定量的評価

基本的な考え方は、数字の場合と全く同じである。実際の実装でも 1 つの文字の表示に対して、8 個の文字コードを割り当ている。計算は数字だけを扱う場合と同様に計算される。簡単のため

- ✓ 漢字
- ✓ ひらがな、カタカナ
- ✓ アルファベット
- ✓ 数字
- ✓ 特殊文字

の合計を 3000 文字とする。このとき必要な文字コードの数は

24000（=3000x8）

である。

12.1 文字コード表全体から、重複なく 24000 箇所を選ぶ場合

1 回目の操作 "0"を表示する文字コードを 256x256x256x256 の文字コード中から選ぶ方法の数
2 回目の操作 "0"を表示する文字コードを 256x256x256x256-1 の文字コード中から選ぶ方法の数
3 回目の操作 "0"を表示する文字コードを 256x256x256x256-2 の文字コード中から選ぶ方法の数
....

8 回目の操作 "0"を表示する文字コードを 256x256x256x256-7 の文字コード中から選ぶ方法の数
9 回目の操作 "1"を表示する文字コードを 256x256x256x256-8 の文字コード中から選ぶ方法の数
10 回目の操作 "1"を表示する文字コードを 256x256x256x256-9 の文字コード中から選ぶ方法の数

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

561 ....
562 ....
563
564 23999 回目の操作 ”フ”を表示する文字コードを 256x256x256x256-23999
565 の文字コード中から選ぶ方法の数
566 24000 回目の操作 ”フ”を表示する文字コードを 256x256x256x256-24000
567 の文字コード中から選ぶ方法の数
568
569 は、全部で 24000 個の文字コード（1 種類の表示に対して 8 個の文字コード。3000 種類分）
570 を、256x256x256x256 個の文字コードの中から、重複なしに選ぶ場合の数である。
571

572 γ = $_{256\times256\times256\times256}C_{24000}$

573
574 計算すると
575
576 γ =2.60E+136486
577
578 である。なお
579
580 $\alpha < \beta < \gamma$
581
582 である。
583

株式会社フュージョンシス
〒105-0023 東京都港区芝浦 1-14-8-901
http://fusionsys.com/ | info@fusionsys.com

12.2 どの文字コードが使われているか完全にわかってしまった場合

先程の自治体職員が、使われている文字コードを記録していき、最終的に使われている 24000 個の文字コードが全て判明したとする。この場合、SF の数は、1 文字の表示に対して、8 個ずつある文字コード 24000 個を一列に並べる方法の数だから、

$$\delta = \frac{24000!}{8!8!\ldots\ldots8!\text{（3000 回の繰り返し）}}$$

で与えられる。計算すると

$$\delta = 1.0E+80888$$

である。

12.3 γとδの関係

γとδの大小関係は

$$\delta < \gamma$$

である。使われている文字コードが特定しきれていない場合、つまり 24000 個の文字コードがすべて判明しているわけではない場合は、δとγの間の値になる。

12.4 漢字にも対応できる SF の堅牢性の最小値

いくら使われている文字コードが判明しても、堅牢性は

$$\delta$$

を下回ることがないということである。

13. security font の脆弱性

脆弱性は、堅牢性と異なって、定量的な評価が難しい。どのような原因で脆弱になるかを説明する。

13.1 起こることが容易に想定される脆弱性

マイナンバーを扱うシステムを、システム実装例で示したような、web ベースで実装をするとする。また security font でマイナンバーを保護するとする。一つの実装法は woff（web open font format）を使う方法である。ブラウザで表示するとき、サーバは許可されたユーザにのみ、マイナンバーが表示できる security font をダウンロードさせ、他のユーザに関しては、ダウンロードさせない。

許可されていないクライアント側のブラウザーは、マイナンバー部分を標準的なフォントで表示しようとするので、文字化けが起こる。一方、許可されたユーザでは、マイナンバー部分が正常に表示される。

しかし、woff はクライアント側の pc にダウンロードされるので（memory に load される前に、ネット関係の temp フォルダに入る）、ハッカーに渡る可能性がある。ハッカーは、入手した woff に対応する encoder で encode された文字を、文字化け無く表示することができる。すなわち decoder が盗まれた状況が発生する。

13.2 上記の脆弱性の克服法 その 1

security font 化された woff が一旦ネット関係の temp フォルダに入ってしまうと、色々な対策（例えば、表示が終わったら、security font 化された woff を削除するなど）をしても、ハッカーの攻撃から防御することは難しいと考えられる。

ブラウザではなく、専用のアプリケーションを c 言語（c 言語は基本的には、逆コンパイルができない。java、c#.net、VisualBasic.net など、中間言語に変換されている言語は、実行形式からソースを復元することが可能）で実装し、security font 化された ttf にも暗号化を施す。このようにすることで、堅牢性は、ある程度は担保される。ただ暗号の強度とは

異なって、c 言語の逆コンパイルの難しさは、程度の問題であり、定量的な評価はできない。

13.3 上記の脆弱性の克服法 その 2

マイナンバーに関するシステムを web ベースで構築することを再度考える。α や δ は、非常に大きいので、

- ✓ 1 桁ごとに異なる security fonts（複数形）を使う
- ✓ ブラウザ画面を refresh すると、更に別の security fonts（複数形）が使われる
- ✓ 別の人のマイナンバーを表示するときは、更に別の security fonts（複数形）を使う

といった実装方法が可能である。100 人分のマイナンバーを表示したとしても、高々

1200 個（=12x100）

個の security fonts が使われるだけである。先に書いたようにブラウザーで SF で保護されたマイナンバーを表示するとき、decoder はユーザの pc にダウンロードされるが、上記のように特定の個人のマイナンバー中の 1 文字しか表示できない、security fonts が多数ダウンロードされるわけで、ハッキングは不可能とはいえないかもしれないが、非常に困難であると考えられる。”克服法 その 1”は c 言語を使って専用アプリを実装するなど、難度の高い、大仰な実装になるが、”克服法 その 2”は比較的工数のかからない実装であり、筆者には現実的であるように思える。

13.4 マッピングを統計的な偏りから推測する

13.4.1 マイナンバーのチェックデジット桁に関する脆弱性

マイナンバーの 12 桁の最初の 11 桁の各桁は、{”0”,”1”,”2”,”3”,”4”,”5”,”6”,”7”,”8”,”9”} がランダムに発生すると考えてよいが、12 桁目はチェックデジット（重み 2,3,4,5,6,7 モジュラス 11）なので、ランダムに発生しているわけではない。チェックデジットがどういう分布をするか実験的に求めたとしよう。仮にチェックデジットとして、”5”が発生することが 1 番多いことが実験的に判明したとすると、12 桁目の数字に対応する SF の文字コードで一番多数回出現するものは、”5”に対応していると推測できる。このように SF は、もとの数字の分布がランダムではないときに、脆弱である可能性がある。

13.4.1.1 上記の脆弱性の克服法 その1

本体の 11 桁とチェックデジットに 2 個の異なる security fonts を使う。チェックデジットで使っている文字コードが推測されても、本体 11 桁が使っている文字コードと無関係である。

13.4.1.2 上記の脆弱性の克服法 その 2

12 桁全てに 12 個の異なる security fonts を使う。1 桁目で使われている 80 個の文字コードが分かっても、2 桁目で使われている 80 個の文字コードを推測することはできない。以下同様に、他の桁で使われている 80 個の文字コードを推測することはできない。

13.4.2 英単語、英文を含む場合の脆弱性

たとえ、decoder や encoder が盗まれていなくても、アルファベットの固有名詞、英語の文章などが、隠蔽されている場合には、

- ✓ 英文では、”e”、”a”、”t”の出現頻度が、他の文字に比べて高い。
- ✓ 英単語の綴りの統計的性質（"t"の前後には、"h"や"s"が来ることはあるが、"z"や"p"が来ることは少ないなど）

を使って、map の数を絞りこめる可能性がある。

http://www7.plala.or.jp/dvorakjp/hinshutu.htm

によると（CNN・ABC・VOA の各ニュースの各分野からランダムに 4 万 1600 文字を抜き出して各文字についてカウントしたもの）、

	回数	ﾊﾟｰｾﾝﾃｰｼﾞ
e	4663	11.40962588
a	3452	8.446499792
t	3345	8.184687661
i	2914	7.130098608
o	2882	7.051799653
s	2850	6.973500697
n	2767	6.770412782
r	2543	6.222320096
h	1739	4.255058847
l	1583	3.87335144
d	1582	3.870904598
c	1306	3.195576109
u	1207	2.953338716
m	1092	2.671951846
p	827	2.023538623

ということなので、英字新聞などの通常の英文に security font が適用されたという前提知識があるとすると、”e”や”a”を表示する文字コード 8 個が等確率で出現すると仮定し、

1 番目に多数回出現する非標準的な文字コード
2 番目に多数回出現する非標準的な文字コード
....
8 番目に多数回出現する非標準的な文字コード
9 番目に多数回出現する非標準的な文字コード
....

など上位の非標準的な文字コードに、”e”や”a”が対応していると推測することができる。

また、"t"の前後には、"h"や"s"が来ることはあるが、"z"や"p"が来ることは少ないなどの知識を

援用すると、更に可能性を絞りこめる可能性がある。以下の表が示すように、英文中のアルファベット 2 文字の連続は偏っている。

	回数	ﾊﾟｰｾﾝﾃｰｼﾞ
th	821	2.008857569
st	373	0.912672197
ed	373	0.912672197
ou	325	0.795223764
ea	249	0.609263745
ai	193	0.472240574
ch	161	0.393941618
tr	125	0.305855294
sh	110	0.269152658
ly	100	0.244684235
rt	98	0.23979055
gh	95	0.232450023
ry	75	0.183513176
ei	52	0.127235802
ht	43	0.105214221
oi	13	0.031808951

13.4.2.1 上記の脆弱性の克服法

- ✓ 適当な文字数
- ✓ 1 個の記事
- ✓ 1 個の文章
- ✓ 1 個の文字

のいずれかを基本単位として、それぞれの基本単位に異なる SF を使う（細かく分割すれば、クライアント側にダウンロードされる SF のファイルサイズは増える）。実装には手間がかかるが、堅牢性は増す。文中の特定の基本単位で使われた文字コードが分かっても、他の基本単位で使われている SF を推測することができなくなる。

13.4.3 漢字、かな、アルファベットを含む場合の脆弱性

日本語にも当然偏りがある。”かな”の偏りは以下のようになるそうである。

仮名頻度表（カッコ内は出現回数）

あ (1672)	か (3518)	さ (1067)	た (3075)	な (2724)	は (2493)	ま (1307)	や (527)	ら (1563)	わ (2445)
い (6653)	き (2386)	し (4571)	ち (1240)	に (3062)	ひ (541)	み (784)	ゆ (310)	り (1804)	を (1919)
う (5095)	く (2563)	す (1611)	つ (1411)	ぬ (22)	ふ (554)	む (349)	よ (1115)	る (2376)	ん (4993)
え (696)	け (1169)	せ (1414)	て (2499)	ね (367)	へ (272)	め (904)		れ (3045)	
お (1080)	こ (2456)	そ (1101)	と (3227)	の (4022)	ほ (531)	も (1669)		ろ (497)	
	が (2008)	ざ (196)	だ (1244)		ば (530)				
	ぎ (451)	じ (1523)	ぢ (0)		び (358)				
	ぐ (155)	ず (186)	づ (102)		ぶ (317)				
	げ (389)	ぜ (284)	で (1806)		べ (273)				
	ご (268)	ぞ (127)	ど (759)		ぼ (171)				

このような偏りを利用して、security font の文字コードのマッピングを推測することが考えられる。

13.4.3.1 上記の脆弱性の克服法

- ✓ 適当な文字数
- ✓ 1 個の記事
- ✓ 1 個の文章
- ✓ 1 個の文字

のいずれかを基本単位として、それぞれの基本単位に異なるSFを使う（細かく分割すれば、クライアント側にダウンロードされる SF のファイルサイズは増える）。実装には手間がかかるが、堅牢性は増す。文中の特定の基本単位で使われた文字コードが分かっても、他の基本単位の場所で使われている SF を推測することができなくなる。

14. AES と security font の堅牢性の質的な違い

共通鍵暗号は、暗号鍵を盗まれると無力化される。しかし鍵が盗まれない限りは、ハッカーは鍵を順次生成して、暗号解除を目指すほかに手はない（rijndael の encoder や decoder のアルゴリズムは公開されており、”encoder や decoder が盗まれる”という概念はそもそもない）。

一方 security font の場合は、暗号鍵を盗まれても、あまり影響はない。しかし、どの文字コードに対して何が表示されるかという情報があると、初見の出力文字列から入力文字列を特定可能である。したがって security font の場合は encoder または decoder のどちらか一方が盗まれれば、解読ができる。

decoder や encoder が盗まれておらず、元の文字列が明確な統計的な性質を持っていないのならば、たとえ使われた文字コードが分かっても、その文字コードをマッピングに用いている文字コード表があまりにも多数あり、元の文字を絞り込めない。security font の堅牢さは、文字コード表が多数個存在することである。

15. security font 内での暗号の位置づけ

security font では暗号による文字の隠蔽は行われていない。弊社の実装では、rjindael を使っているが、これは文字コード表内での適当な配置を自動的に決定するためだけに使っている。文字コードは ascii 文字コード表の場合、128 しかないので、虱潰しにしても大した計算量ではなく、rijndael 暗号の強度は、security font の堅牢性に直接的には関係しない。

16. 伝統的な方法-----データベースのレコードを暗号化する

これは、伝統的に使われてきた方法で、安全性の高い暗号を用いれば、堅牢性は非常に高い。反面、共通鍵暗号で暗号化した場合は、鍵の配送や鍵の扱いに困難が伴う。

例えば、データベースのレコード内では暗号化されているマイナンバーを表示する場合には、デー

タベースからレコードを引き出した後に、鍵を持った復号器で暗号を解除することが必要になるが、これは共通鍵をプログラム内で操作することが必要になり、security 上好ましくない。

17. 総合評価

以下は、5 点満点である。極めて主観的だが、それはご容赦を願います。

	堅牢性	実装しやすいか	実績	得点合計
AES（rijndael)	5	2 （鍵の配送や鍵の扱いが難しい）	5	12
security font	2.1（encoder や decoder が盗まれないのであれば、実用レベル）	5	0	7.1

17.1 コメント

security font の実績は、ないのが当たり前といえば当たり前である。実績の部分を差し引いて考えると、AES が 7 点であり、security font が 7.1 点である。

security font の堅牢性は、出現する頻度に統計的な偏りのない数字の場合が最も高く、文字の出現頻度に様々な偏りのある英語の文章、英単語、日本語文章の場合が低くなると考えられる。ただし既に書いたように、様々な対処法はある。

18. 更に堅牢な方法

18.1 タイムスタンプをマッピングに入れる方法（特許申請済）

タイムスタンプをマッピングに入れる方法（特許申請済）は、既に実装済みである。

表示開始---->この security font が作られた日時
表示終了---->この security font が作られた日時から 1 年後

といった、1 年間だけ有効な security font が容易に実装できる。いわば”使い捨て”の隠蔽手段が実現される。

18.2 gps 情報をマッピングに入れる方法（特許申請済）

緯度経度をマッピングに入れる方法（特許申請済）は、既に実装済みである。地球上の特定の場所でないと、security font が表示できないといった機能が容易に実装できる。

19. 参考文献

[高見 2015] セキュリティフォント概要_20151215.docx

■