

PKI トークンユーティリティ
mToken CryptoID 用

RIBIG INC.

内容

プログラムについて	3
デモ版について	3
1. PIN	4
2. List コマンド	7
3. Gen コマンド.....	8
RSA キーペア生成	8
AES 秘密鍵の生成.....	9
4. DEL コマンド	10
5. CSR コマンド.....	12
6. CA コマンド	15
6.1 トークン CA 作成.....	15
6.1.1 ルート CA 作成.....	15
6.1.2 中間 CA 作成	16
6.2 CSR 署名.....	21
トークン CA による署名.....	21
6.3 発行済み証明書の無効化.....	23
6.4 トークンCAのCRL作成.....	24
7. IMPORT コマンド	26
CRT フォルダ内の証明書ファイルのインポート.....	27
PEM 形式 RSA 秘密鍵のインポート.....	28
PFX/P12 ファイルのインポート (拡張子 pfx と p12 はどちらでも構いません)	30
AES 秘密鍵インポート	31
8. Export コマンド	34
証明書エクスポート.....	34
RSA 公開鍵エクスポート.....	35
9. Enc & Dec コマンド	37
RSA 公開鍵で暗号化、RSA 秘密鍵で復号化.....	37
OpenSSL で公開鍵暗号化、トークンの秘密鍵復号化	38
トークンで公開鍵暗号化、OpenSSL で秘密鍵復号化	39
AES 鍵で暗号化/復号化.....	40
10. 署名 / 検証.....	45
署名.....	45
検証.....	45
トークンと OpenSSL による署名と検証	46

トークン内の証明書が特定CAにより発行されたものかどうかを検証	47
11. INIT コマンド.....	49
12. PIN コマンド	50
SO Pin 変更.....	50
USER PIN 変更.....	50
USER PIN リセット.....	51
13. Set コマンド.....	52
14. トークンフィルタ.....	55
--slot オプション	55
--token オプション.....	56
付録 1.....	58

プログラムについて

- 本プログラムは接続トークン全てにコマンドを実行します。コマンド対象にしないトークンは接続しないようにしてください。
- 接続した複数トークンから SlotID/トークンラベルを指定して、特定のトークンだけを操作対象にすることができます。詳細はトークンフィルタの章を参照してください。
- 配布 ZIP はプログラム本体と PDF マニュアルが含まれます。
 - Token.exe
 - Manual.pdf
 - Openssl.exe
 - libssl-3.dll
 - libcrypto-3.dll

 - [CSR]
 - [CRT]
 - createCA.ps1
 - sign.ps1

[CSR],[CRT]は空。CSR や証明書を保存するときに使われます。

デモ版について

- トークンのユーザ PIN は 123456 でなければなりません(PIN / INI コマンドは例外)。本プログラムは常に ユーザ PIN 123456 を出力、トークンのユーザ PIN が 123456 でなければ “Login Error”になります。PIN / INI コマンドはコマンドラインで指定した PIN が使われます。
- デモ版は評価用です。その他の目的での利用はできません。

1. PIN

本プログラムでトークン进行操作するには、ほとんどのコマンドでユーザ PIN を指定する必要があります。

例 e:

```
>token gen --pin 123456 -id test
- GEN コマンドは指定 ID 名で RSA キーペアを生成
```

```
.\token gen --id test --pin 123456
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Generated Key Pair ***
```

****オプション指定はハイフン2つが必要です; --pin, --id 等 もし、ハイフン1つで指定すると option error になります。*

```
token list --pin 123456
- LIST コマンドトークン内の証明書、キーペア、秘密鍵を表示
```

```
.\token list --pin 123456
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
id=test
- public key
- private key
```

コマンドを実行する毎に PIN を入力するのは面倒です。PIN の異なる複数のトークンが接続されていたら、それぞれの PIN を指定しなければなりません、コマンドラインでそれぞれの PIN を正しく入力することは不可能です。

本プログラムはこの PIN 指定問題を PIN 自動入力機能で解消しています。トークンを同じフ

フォルダにトークンのシリアル番号と同名の設定ファイル(シリアル番号.INI)が存在したら、そのファイル内に記載されている PIN を読み込みます。本プログラムがシリアル番号, 25FEB2B2DB013B0B のトークンを検出したら、同じフォルダ内の設定ファイル

25FEB2B2DB013B0B.ini

を読み込みます。設定ファイルが見つからなければ最終的に PIN は未定、もしくは、コマンドラインで指定されている PIN になります。

設定ファイルの [PIN]セクションの“User” と “SO” キーでユーザ PIN と SO PIN を設定します。

```
[PIN]
USER=123456
SO=admin123
```

設定ファイルは手動で作成しても構いませんが、自動作成させるのが便利です。本プログラムで PIN 変更コマンドを実行すると、トークン PIN 変更後、設定ファイル作成/更新します。

```
>token pin --change --pin 123456 --newpin 12345678
➔ PIN コマンドの--change オプションでユーザ PIN を変更します。
この例では 123456 から 12345678 に変更します。トークンのユーザ PIN 変更度、設定ファイルの USER キーが 12345678 に更新されます。
```

```
token.exe pin --change --pin 123456 --newpin 123456
➔ 変更前、変更後の PIN は同じでも構いません
```

```
.\token.exe pin --change --pin 123456 --newpin 123456
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** USER Pin Successfully Changed ***
```

現行ユーザ PIN が 12345678 のトークンを複数接続して、以下コマンドを実行するとすべ

てのトークンの PIN を 123456 に変更、すべてのトークン用の設定ファイルが作成/更新されます。

```
>token pin --change --pin 12345678 --newpin 123456
```

SO PIN 変更

```
>token.exe pin --change_so --sopin admin123 --newpin admin123  
-> PIN コマンドの --change_so オプションで SO PIN を変更します。コマンド  
実行後、設定ファイルの SO キーに admin123 が設定されます。
```

設定ファイルが作成されているトークンには PIN を指定する必要がなくなります。

```
>.\token list  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
id=test  
- public key  
- private key
```

**デモ版は設定ファイル/コマンドラインでどのようなPINを指定しても、ユーザ PIN 123456 は固定されます。

多数のトークンの PIN を INI ファイルで管理するのは面倒かも知れません。INI ファイルだけでなくデータベース(MariaDB)に書き込むオプションが用意されています。詳細は付録1を参照ください。

2. List コマンド

LIST コマンドはトークン内の RSA キー、証明書、秘密鍵を一覧表示します。また、トークンのシリアル番号、Slot 番号、ラベル、モデルを表示します。

```
> .\token.exe list
>>reading tokens...over<<

Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----

id=rsakey1
- public key
- private key

id=aeskey1
- AES Key
```

トークン内のオブジェクト(公開鍵、秘密鍵、証明書、AES 鍵)は ID で識別されます。同じ種類のオブジェクトは同一 ID で保存することはできません。

トークン情報のみを表示するには --info オプションを指定します。

```
> .\token.exe list --info
>>reading tokens...over<<

Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
```


3. Gen コマンド

GEN コマンドは RSA キーペア、AES キーをトークン内部で生成します。既定では RSA キーペアを生成します。

RSA キーペア生成

```
>token gen --id test  
→ ID“test”の RSA キーペアを生成
```

```
.\token gen --id test  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Generated Key Pair ***
```

既に ID “test” のキーペアが存在すると、キーペアは生成されません。

```
.\token gen --id test  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
---> private/public keys with the same id found
```

既存キーペアを上書きするには --new オプションを指定します。

```
.\token gen --id test --new  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
---> priv obj deleted  
---> pub obj deleted  
*** Generated Key Pair ***
```

AES 秘密鍵の生成

AES 秘密鍵を生成するには --aes オプションを指定します。

```
>token.exe gen --id test --aes
```

```
.\token gen --id test --aes
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Generated AES Key (id=test )***

> .\token list
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
id=test
- public key
- private key
- AES Key
```

ここでは ID “test” で AES 秘密鍵を生成しています。既に ID “test” で RSA キーペアが保存されていますが、オブジェクトクラスが異なるため同じ ID で AES 秘密鍵は保存できません。

AES は対称暗号方式なので、暗号化した秘密鍵で復号化します。トークン内部で生成した AES 秘密鍵は、外部に取り出すことはできません。そのため、あるトークンの AES 秘密鍵で暗号化したデータは、同じトークンの AES 秘密鍵でのみ復号化可能です。

あるトークンで暗号化したデータを別のトークンで復号化するには、外部 AES 秘密鍵をトークンにインポートします。同じ AES 秘密鍵を複数トークンで共有できます。詳細は IMPORT コマンド章を参照ください。

4. DEL コマンド

DEL コマンドはトークン内の RSA キーペア、AES 秘密鍵、証明書を削除します。

```
>token del --id test  
→ ID “test”のオブジェクトを削除します。
```

```
.\token del --id test  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Private key( id= test ) deleted ***  
*** AES key( id= test ) deleted ***
```

CryptoID は、トークン内部の RSA 公開鍵/証明書を対応する秘密鍵と一体になっているとみなします。秘密鍵を削除すると対応する公開鍵/証明書は削除されます。公開鍵/証明書を削除するには、秘密鍵を削除しなければなりません。

個別オブジェクトを削除するオプション

```
--privkey    RSA キーペア/証明書を削除  
--aes        AES 秘密鍵を削除
```

```
>token del -id xxx --privkey|--aes  
→ ID “xxx”で識別される RSA キーペア/証明書 | AES 秘密鍵を削除
```

```
.\token del --id test --aes  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** AES key( id= test ) deleted ***
```

```
.\token del --id test --privkey
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Certificate( id= test ) deleted ***
*** Public key( id= test ) deleted ***
*** Private key( id= test ) deleted ***
```

5. CSR コマンド

CSR コマンドは、RSA 秘密鍵から CSR(Certificate Signing Request) を作成します。

```
>token csr --id test
```

→ ID “test” の RSA 秘密鍵から CSR を作成、PEM フォーマットで画面に出力

```
.\token csr --id test
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
/CN=25FEB2B2DB013B0B/OU=/O=/ST=/L=/C=JP
-----BEGIN CERTIFICATE REQUEST-----
MIICbTCCAUVCAQAwwKDEZMBcGA1UEAwQMjVGRUlyQjJEQjAxM0lwQjELMAkGA1UE
...
...
Ww==
-----END CERTIFICATE REQUEST-----
```

出力をファイルに保存するには `-out` オプションにファイル名を指定します。

```
>token csr --id test --out test-csr.pem
```

→ 作成した CSR をファイル“test-csr.pem” に保存.

```
.\token csr --id test --out test-csr.pem
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
/CN=25FEB2B2DB013B0B/OU=/O=/ST=/L=/C=JP
*** CSR 'C:\xxx\yyy\test-csr.pem' created ***
```

ファイル名の代わりにディレクトリ名を指定することもできます。指定するディレクトリは存在しなければなりません。

```
>token csr --id test -out (folder name)
```

```

.\token csr --id test --out h:\users
>>reading tokens...over<<
Output folder : h:\users
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
/CN=25FEB2B2DB013B0B/OU=/O=/ST=/L=/C=JP
*** CSR "h:\users\25FEB2B2DB013B0B-test.csr" created ***

```

--out に . (ドット) を指定すると同じフォルダ内の CSR フォルダを指定したことになります。

--out にフォルダ名を指定すると、ファイル名は自動生成されます。

ファイル名: トークンシリアル番号-CSR を作成した RSA 秘密鍵の ID.CSR

シリアル番号 25FEB2B2DB013B0B のトークン内部の ID “test” の秘密鍵から CSR を作成したとするとファイル名は、25FEB2B2DB013B0B-test.csr になります

```

.\token csr --id test --dir .
>>reading tokens...over<<
Output folder : C:\xxx\yyy\CSR
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
/CN=25FEB2B2DB013B0B/OU=/O=/ST=/L=/C=JP
*** CSR "C: C:\xxx\yyy\CSR \25FEB2B2DB013B0B-test.csr" created ***

```

PEM 形式で CSR ファイルが CSR フォルダに保存されます。複数トークンが接続されていると、すべてのトークンに対してコマンドが適用されます。指定 ID の RSA 秘密鍵がトークン内であれば CSR ファイルが作成されます。

既定では Common Name(CN) はトークンのシリアル番号になります。--subj オプションで属性を指定することができます。

Must be one line.

```
>.\token csr --id test --dir . --subj  
"/cn=ribig.co.jp/ou=sales/o=ribig/st=kanagawa/l=Yokohama/C=JP"  
>>reading tokens...over<<  
  
Output folder : C:\xxx\yyy\CSR  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
/CN=ribig.co.jp/OU=sales/O=ribig/ST=kanagawa/L=Yokohama/C=JP  
*** CSR "C: C:\xxx\yyy\CSR\25FEB2B2DB013B0B-test.csr" created ***
```

コマンドの画面への出力をファイルにリダイレクトすることができます。

```
>.\token csr --id test >> mytest.csr  
>>reading tokens...over<<  
  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
/CN=25FEB2B2DB013B0B/OU=/O=/ST=/L=/C=JP
```

標準出力をファイル mytest.csr にリダイレクトします。

パワーシェルではファイルへの出力は Unicode エンコードされます。最初に2バイトは BOM (Byte Order Marker) で ff fe になっています。

00000000	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	
00000000	ff	fe	2d	00	2d	00	2d	00	2d	00	2d	00	42	00	45	00	[-,.,-.,-.,B.E.
00000010	47	00	49	00	4e	00	20	00	43	00	45	00	52	00	54	00	G.I.N. .C.E.R.T.
00000020	49	00	46	00	49	00	43	00	41	00	54	00	45	00	20	00	I.F.I.C.A.T.E. .
00000030	52	00	45	00	51	00	55	00	45	00	53	00	54	00	2d	00	R.E.Q.U.E.S.T.-.
00000040	2d	00	2d	00	2d	00	2d	00	0d	00	0a	00	4d	00	49	00	-.,-,-,.....M.I.

OpenSSL はこのようなファイルを正当な CSR ファイルとみなしません。エディタでプレーンテキストファイルとして保存しなおすなどして BOM を取り除いてください。

6. CA コマンド

CA コマンドで

CA(認証局)作成
CSR への署名(証明書発行)
証明書無効化处理

を行います。

CSR を署名するには、独自 CA を立てなければなりません。CA は OpenSSL を使って作成できますが、CA の秘密鍵をトークン内で生成、トークン内に格納されているときに CA コマンドで CA 関連操作を行えます。

6.1 トークン CA 作成

独自 CA を立てるには、自己署名証明書を持つルート CA が最低限1つ必要です。CA コマンドでルート CA を作成できます。また、CAによって署名された証明書を持つ中間認証局の作成もできます。

6.1.1 ルート CA 作成

```
> token ca --gen --id CA
```

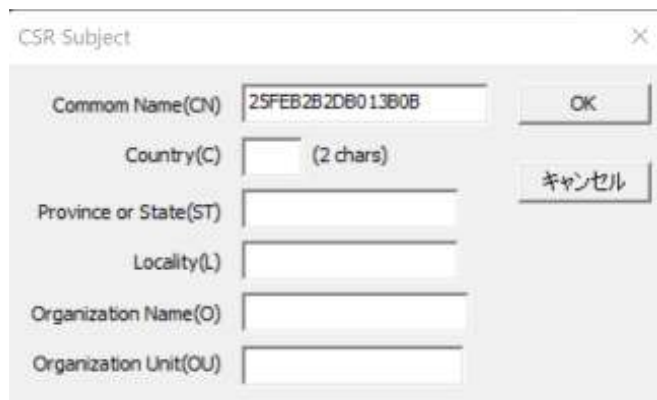
→ ルート CA の秘密鍵、証明書を ID SomeID でトークンに格納
本プログラムと同じフォルダに “CA-トークンシリアル-ID“ という名前の CA
フォルダを作成

```
> .\token.exe ca --gen --id CA
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
*** Generated Key Pair ***

*** CA Certificate succesfully set ***
```


トークンに鍵と証明書が設定、同時に“CA-25FEB2B2DB013B0B-CA” というフォルダが作成されます。

--subj オプションが指定されていないければ、自己署名する CSR の属性の入力を求めるダイアログが表示されます。既定の CN はトークンにシリアル番号です。



*CN の入力は必須

コマンドラインでの -subj オプション指定(例)

```
>¥token.exe ca --id CA --gen  
--subj /CN=TokenCA/L=Yokohama/ST=Kanagawa/C=JP/O=RiBiG  
Inc./OU=System”
```

秘密鍵はトークン内に格納され、CA フォルダには置かれません。

6.1.2 中間 CA 作成

中間 CA は上位 CA で署名された証明書を持つ CA です。作成する中間 CA、または、上位 CA の秘密鍵がトークン内に格納されているのは、次の3つの組み合わせになります。

	上位 CA	中間 CA
ケース 1	トークン内	トークン内
ケース 2	フォルダ内	トークン内
ケース 3	トークン内	フォルダ内

ケース 1 上位 CA/中間 CA の秘密鍵がどちらもトークン内

中間 CA の秘密鍵を格納するトークンを接続してください。また、上位 CA の秘密鍵を格納しているトークンも接続してください。

```
> .\token.exe ca --gen --id IntermediateCA --upperCA  
@25FEB2B2DB013B0B-CA
```

→ 中間 CA の秘密鍵、証明書を ID “IntermediateCA” で格納。証明書は – upperCA で指定されている上位 CA の秘密鍵で署名

--upperCA オプション書式: @ + token serial number + “-” + ID

```
> .\token.exe ca --gen --id IntermediateCA --upperCA @25FEB2B2DB013B0B-CA  
>>reading tokens...over<<  
Slot: 1 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )  
-----  
*** Generated Key Pair ***  
*** CA Certificate succesfully set ***  
  
> .\token.exe list --slot 1  
>>reading tokens...over<<  
Slot: 1 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )  
-----  
id=IntermediateCA  
- certificate CN=597909FB7876EC19  
- public key  
- private key
```

--subj オプションが指定されていないければ、中間 CA の CSR 属性の入力を求めるダイアログが表示されます。既定の CN はトークンにシリアル番号です。

上位 CA の秘密鍵を格納しているトークンの INI ファイルが見つければ、INI ファイルのトークンの PIN でアクセスします。見つからなければ、上位 CA トークンの PIN 入力を促されます。

ルート CA 同様、中間 CA の CA フォルダが本プログラムと同じフォルダに作成されます。

ケース 2 上位 CA の秘密鍵はフォルダ内 / 中間 CA の秘密鍵はトークン内
中間 CA の秘密鍵を格納するトークンを接続します。

```
> .¥token.exe ca --gen --id IntermediateCA --upperCA CA
```

→ ID “IntermediateCA” で中間 CA の秘密鍵、証明書を作成/格納。証明書は -upperCA で指定されている本プログラムと同じフォルダ内の CA フォルダ内の秘密鍵で署名。

--upperCA オプションで上位 CA のフォルダを指定します。上位 CA のフォルダは以下のように証明書、秘密鍵を保持しているものとします。

```
[CA folder]
    cacert.pem
    [private]
        cakey.pem
```

証明書は cacert.pem, 秘密鍵は [private]フォルダ内に cakey.pem として置かれていなければなりません。

中間 CA の CA フォルダが本プログラムと同じフォルダに作成されます。

Case 3 中間 CA の秘密鍵はフォルダ / 上位 CA の秘密鍵はトークン
上位 CA の秘密鍵を含むトークンを接続してください。

```
¥token.exe ca --gen --lowerCA --id CA lowerCA
```

→ 中間 CA を本プログラムと同じフォルダ内に “lowerCA” というフォルダ内に作成、その証明書はトークン内の上位 CA の秘密鍵 (ID =CA)で署名

```
.¥token.exe ca --gen --lowerCA --id CA lowerCA
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
-->Created new CA private key
-->Created new CA certificate ( signed by UpperCA token=25FEB2B2DB013B0B,id=CA )
*** CA created successfully in C:\xxx\yyy\zzz\lowerCA ***
```

もし、指定して中間 CA フォルダと同名フォルダが存在したら、処理は中止します。

```
xxx Directory C:¥xxx¥yyy¥zzz¥¥lowerCA already exists xxx
```

中間 CA の秘密鍵はフォルダ内に置かれますので、鍵を保護するためのパスワードの入力を求められます。



--subj オプションが指定されていないければ、中間 CA の CSR 属性の入力を求めるダイアログが表示されます。既定の CN はトークンにシリアル番号です。



6.2 CSR 署名

トークン CA による署名

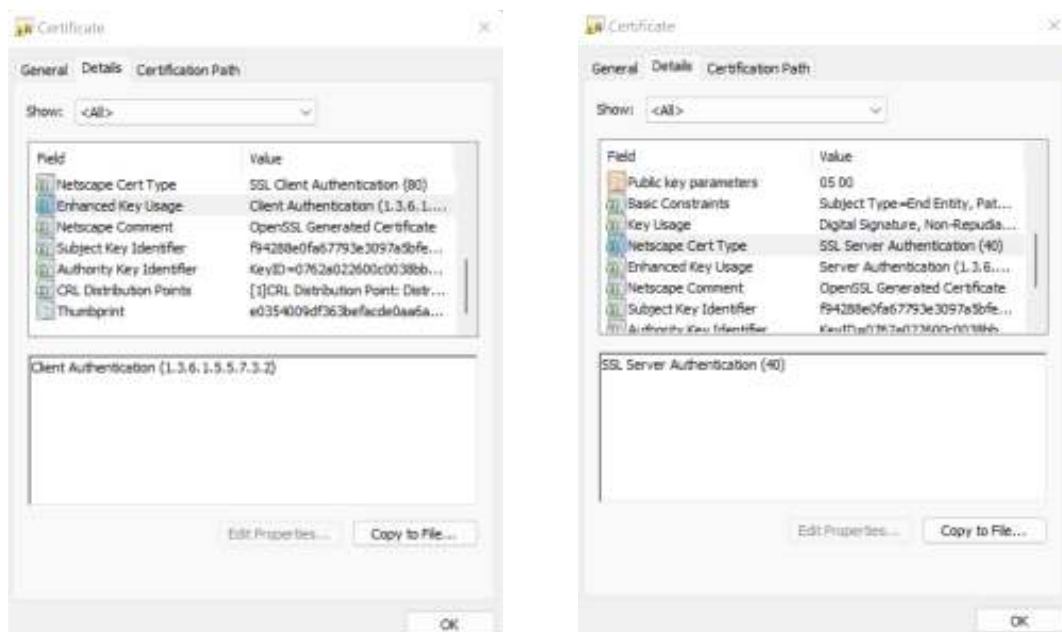
```
>token.exe ca --sign --id CA --in .some.csr --out some.csr.crt --certtype client
```

- CSR ファイル “some.csr” をトークン内の ID で指定 CA 秘密鍵で署名、証明書ファイルは some.csr.crt として保存。証明書はクライアント認証用

```
.\token.exe ca --sign --id CA --in some.csr --out some.csr.crt --certtype client
PKI Token Utility
>>reading tokens...over<<

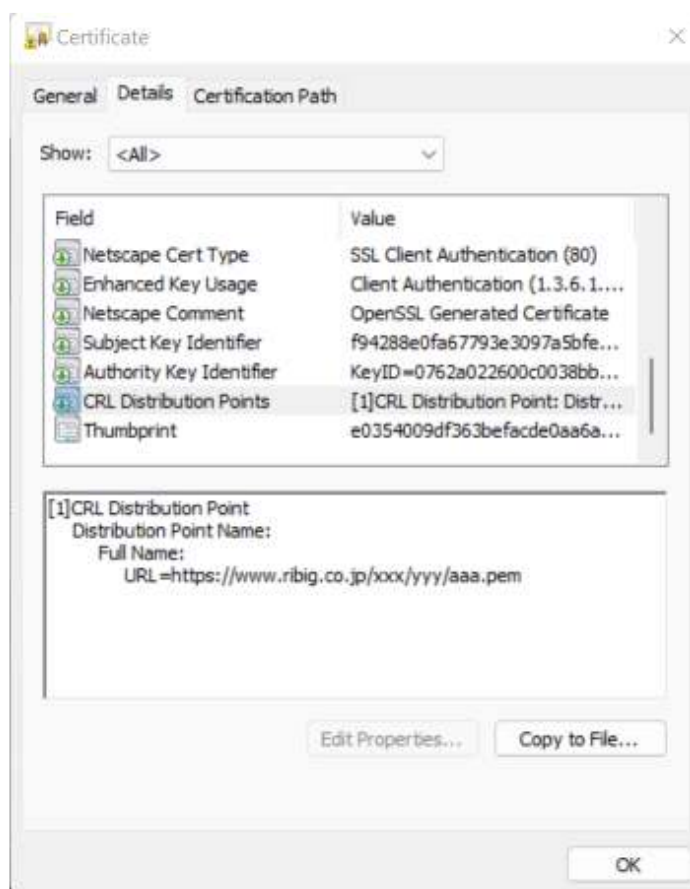
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E)
-----
*** Signed C:\xxx\yyy\zzz\some.csr ***
--> C:\xxx\yyy\zzz\some.csr.crt
```

--certtype オプションには client, server, codesign のいずれかを指定。--certtype オプションを指定しなければ、証明書用途はすべての目的用になります。



--crlurl オプションで CDP (Certification Revocation Distribution Point) —
CRL(Certificate Revocation List) を置く URL—を証明書に設定できます。

```
.\token.exe ca --sign --id CA --in some.csr --out some.csr.crt --certtype client --crlurl  
https://www.ribig.co.jp/xxx/yyy/aaa.pem  
PKI Token Utility  
>>reading tokens...over<<  
  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )  
-----  
*** Signed C:\xxx\yyy\zzz\some.csr ***  
--> C:\xxx\yyy\zzz\some.csr.crt
```



--in オプションで .(dot)を指定すると本プログラムと同じフォルダ内の CSR フォルダを意味します。

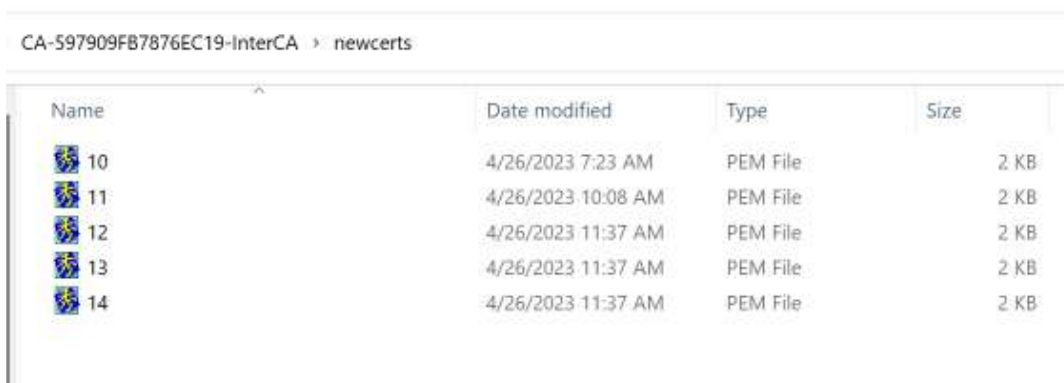
--out オプションで(dot)を指定すると本プログラムと同じフォルダ内の CRT フォルダを意味します。この場合、証明書名は、CSR ファイルに .crt 拡張子が追加されたものになります。

```
>.\token.exe ca --sign --id CA --in some.csr --out .
PKI Token Utility
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
*** Signed C:\xxx\yyy\zzz\some.csr ***
--> C:\xxx\yyy\zzz\CRT\some.csr.crt
```

6.3 発行済み証明書の無効化

トークン CA によって発行された証明書を無効化することができます。発行済み証明書は、CA フォルダ内の newcerts フォルダに置かれています。無効化する証明書を見つけて無効化処理を実施します。

トークン 597909FB7876EC19 の ID InterCA の CA フォルダ



Name	Date modified	Type	Size
10	4/26/2023 7:23 AM	PEM File	2 KB
11	4/26/2023 10:08 AM	PEM File	2 KB
12	4/26/2023 11:37 AM	PEM File	2 KB
13	4/26/2023 11:37 AM	PEM File	2 KB
14	4/26/2023 11:37 AM	PEM File	2 KB


```
>token ca --revoke -id interCA -in ( certificate file)
```

```
>.\token ca --revoke --id InterCA --in 11.pem
PKI Token Utility
>>reading tokens...over<<

Slot: 0 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )
-----
*** Revoking Certificate 11. ***
```

証明書 11.pem を無効化。[newcerts]フォルダ内のファイル名を指定します。証明書の絶対パスを指定すると指定されたファイルが使われます。

CA フォルダの “index.txt” は発行済み証明書の状態を保持しています。各行の最初の文字で有効 V(alid) / 無効(R(evoked)) かが分かります。

6.4 トークンCAのCRL作成

トークンCA用の証明書無効リストを作成できます。

```
>token ca --gencrl -id CA
```

```
>.\token.exe ca --gencrl --id InterCA
PKI Token Utility
>>reading tokens...over<<

Slot: 0 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )
-----
*** File C:\xxx\yyy\CA-597909FB7876EC19-InterCA\crl.pem created ***
```

CRL ファイル (crl.pem) が CA フォルダ CA-597909FB7876EC19-InterCA に作成されます。このファイルを証明書内に設定したCDPで公開できます。CRLの既定の有効期限は30日。

--days オプションで有効期限は設定できます。

```
>token ca --gencrl -id CA -days 90
```

CRLの有効期限は90日

指定して有効期限ごとに CRL は更新しつづけなければなりません。

7. IMPORT コマンド

このコマンドは外部の RSA キーペア、AES 秘密鍵、証明書をトークンに取り込みます。

```
>token import --id test --in CRT¥25FEB2B2DB013B0B-test.crt
カレンドフォルダの CRT フォルダ内の 25FEB2B2DB013B0B-test.crt 証明
書ファイルを“test” という ID でインポート
```

```
>.\token import --id test --in crt\25FEB2B2DB013B0B-test.crt
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--- private key ( id=test ) for the certificate found in the token ---
*** Certificate succesfully imported (crt\25FEB2B2DB013B0B-test.crt) ***
```

トークン内には証明書の公開鍵に対応する秘密鍵がなければなりません。本プログラムは証明書ファイルのインポート中に対応秘密鍵が存在するかどうか確認します。秘密鍵が見つければ、その秘密鍵の ID とコマンドラインで指定された ID を比較、一致すればそのままインポートします。上の例では ID が一致しているため、コマンドラインで指定された ID でインポートされます。

コマンドラインで指定した ID と、見つかった秘密鍵の ID が一致しなければ、秘密鍵の ID でインポートを試みます。

```
>token import --id test2 --in CRT¥25FEB2B2DB013B0B-test.crt
```

```
>.\token import --id test2 --in crt\25FEB2B2DB013B0B-test.crt
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--- private key ( id=test ) for the certificate found in the token ---
      (id for this certificate will be test)
*** Certificate succesfully imported (crt\25FEB2B2DB013B0B-test.crt) ***
```

この例ではコマンドラインで ID “test2” を指定していますが、証明書の公開鍵と一致する秘密鍵が ID “test” のため、その秘密鍵と同じ ID でインポートされています。

同じコマンドを再度実行すると ID “test” で証明書が既に存在するためインポートは失敗します。

```
> .\token import --id test2 --in crt\25FEB2B2DB013B0B-test.crt
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--- private key ( id=test ) for the certificate found in the token ---
      (id for this certificate will be test)
xxx certificate with id(test) already exists xxx
```

--new オプションを指定すると既存の証明書を上書きします。

```
> .\token import --id test2 --in crt\25FEB2B2DB013B0B-test.crt --new
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--- private key ( id=test ) for the certificate found in the token ---
      (id for this certificate will be test)
*** Certificate succesfully imported (crt\25FEB2B2DB013B0B-test.crt) ***
*** Import Successful ***
```

CRT フォルダ内の証明書ファイルのインポート

証明書ファイルを指定する代わりに、証明書ファイルが保存されたフォルダを—in オプションに指定することができます。

```
>token import --in (folder name)
```

→ 指定フォルダ内の CRT, PEM, PFX 拡張子のファイルをインポート

指定フォルダ内のファイルは決められたフォーマットの名前で保存されていなければなりません。

フォーマット

シリアル番号-ID.CRT|PEM|PFX

本プログラムはフォルダ内のファイル名からシリアル番号と ID を取り出し、シリアル番号と一致するトークンが接続しているかどうか確認、接続していれば ID でインポートします。

ファイル拡張子は、CRT(証明書ファイル)、PEM(RSA キー、証明書) または PFX/P12(秘密鍵+証明書)でなければなりません。

--in オプションで .(ドット)を指定すると同じフォルダの CRT フォルダを指定したことになります。

```
>token import --in .
```

→CRT フォルダ内の CRT, PEM, PFX ファイルをインポート

```
> .\token import --in .
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--- Private key (ID='ribig') for the certificate found in the token ---
*** Certificate succesfully imported (C:\xxx\yyy\CRT\25FEB2B2DB013B0B-ribig.crt) ***
*** Import Successful ***

Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--- Private key (ID='test') for the certificate found in the token ---
*** Certificate succesfully imported (C:\xxx\yyy\CRT\25FEB2B2DB013B0B-test.crt) ***
*** Import Successful ***
```

この例では2つの証明書ファイル(25FEB2B2DB013B0B-ribig.crt、25FEB2B2DB013B0B-test.crt)が CRT フォルダにあり、シリアル番号 25FEB2B2DB013B0B のトークンに ID “ribig”と ID “test” でそれぞれインポートされています。1度のコマンド実行で複数トークン、複数 ID で証明書をインポートすることができます。

PEM 形式 RSA 秘密鍵のインポート

トークンにインポートする RSA 鍵は openssl.exe で生成できます。

RSA 秘密鍵生成コマンド

パスワード無

```
OpenSSL genrsa -out privkey.pem 2048
```

パスワード保護

```
OpenSSL genrsa -aes256 -passout pass:abc123 -out privkey.pem  
2048
```

秘密鍵は通常、パスワードで保護するようにしてください。

RSA 秘密鍵ファイルを指定してトークンにインポートします。

```
>token.exe import --id imported --in privkey.pem
```

→RSA 秘密鍵 privkey.pem を ID “imported”でインポート

PEM ファイルはコンテンツを示すヘッダにより秘密鍵、公開鍵、証明書であるかを知ることができます。本プログラムはヘッダでコンテンツを識別します。

```
> .\token import --id ribig --in privkey.pem  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Private/Public key succesfully imported (privkey.pem) ***  
*** Import Successful ***
```

パスワード保護されたファイルをインポートすると途中でパスワードを求められます。



コマンドラインの `-pass` オプションでパスワードを指定することもできます。

```
> .\token import --id imported --in privkey.pem --pass abc123
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Private/Public key succesfully imported (privkey.pem) ***
*** Import Successful ***
```

PFX/P12 ファイルのインポート (拡張子 pfx と p12 はどちらでも構いません)

PFX/P12 ファイルは秘密鍵と証明書を含みます。

PFX ファイル作成(パスワード無)

```
>openssl pkcs12 -export -inkey privkey.pem -in imported.crt -out
client.pfx -passout pass:
```

PFX ファイル作成(パスワード保護)

```
>openssl pkcs12 -export -inkey privkey.pem -in imported.crt -out
client.pfx -passout pass:abc123
```

PFX/P12 ファイルは秘密鍵を含みますので、パスワードで保護するようにしてください。

```
>token import --id ribig --in client.p12
→ client.p12 を ID “ribig” でインポート。パスワード保護されていれば途中でパ
スワード入力を求められます。--pass オプションで事前に指定することもできます。
```

```
> .\token import --id ribig --in client.p12
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Import Successful ***
```

RSA 秘密鍵をインポートすると公開鍵は自動的に作成れます。

```
> .\token list
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
id=ribig
- certificate CN=25FEB2B2DB013B0B
- public key
- private key
```

AES 秘密鍵インポート

外部 AES 秘密鍵をトークンにインポートできます。生の秘密鍵はインポートしません。本プログラムは、与えられたパスワードとソルトから PBKDF2 (Password Based Key Derivation Function 2)アルゴリズムで 秘密鍵と IV を生成してトークンにインポートします。

ソルトは8バイト。16バイトの16進文字列で指定します。乱数を生成する RAND コマンドでランダムな8バイトを生成して16バイトの16進文字列を表示できます。

```
>token rand -len 8 -hex
→ 8バイト乱数生成、16進文字列として出力
```

```
> .\token rand --len 8 --hex
>>reading tokens...over<<
Serial : 25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
78c8a3c7b5e0eef6
```



```
>token rand -len 8 -hex -out salt.txt
```

→8バイト乱数生成、16進文字列をファイル“salt.txt”として出力

```
> .\token rand --len 8 --hex --out salt.txt
> .\token import --id imported --aes --pass PASSWORD --salt file:salt.txt
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** AES imported. Remember pass/salt ***
```

AES 秘密鍵インポート.

```
>token import --id imported --aes --pass PASSWORD --salt
7f2f8c7406ddd03e
```

パスワード PASSWORD, ソルト7f2f8c7406ddd03e でAES秘密鍵をインポート

```
> .\token import --id imported --aes --pass PASSWORD --salt 7f2f8c7406ddd03e
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** AES imported. Remember pass/salt ***
```

--p オプションを指定すると生成された生の秘密鍵と IV を表示できます。

```
> .\token import --id imported --aes --pass PASSWORD --salt 7f2f8c7406ddd03e --p
>>reading tokens...over<<
Slot: 0 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )
-----
salt=7F2F8C7406DDD03E
key=B8E67A17279A5971F488E84746E6BF6A
iv=A460D0EE3F13699DB3B2E403B365BE51
*** AES imported. Remember pass/salt ***
```

--salt オプションにはファイル名を指定できます。ファイルは16バイト16進文字列が含まれ

ていなければなりません。

8. Export コマンド

このコマンドは RSA 公開鍵、証明書をエクスポートします。

証明書エクスポート

```
>token export -id test -cert
```

→ID “test” の 証明書を標準出力にエクスポート

```
> .\token export --id test --cert
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Certificate( id= test ) found ***
-----BEGIN CERTIFICATE-----
MIIODCCAyCgAwIBAgIBBTANBgkqhkiG9w0BAQsFADBRMQ4wDAYDVQQDDAVyaWJp
ZzEPMA0GA1UECwwGc3lzdGVtMQ4wDAYDVQQKDAVSaUJpRzERMA8GA1UECAwIS2Fu
...
-----END CERTIFICATE-----
```

--out オプションで出力ファイルを指定します。

```
>token export -id test -cert --out cert-test.pem
```

→ ID “test” の 証明書をファイル“cert-test.pem”にエクスポート

```
> .\token export --id test --cert --out cert-test.pem
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Certificate( id= test ) found ***
--> cert-test.pem
```

RSA 公開鍵エクスポート

>token export --id test --pubkey --out pubkey-test.pem
→ ID “test”の RSA 公開鍵をファイル “pubkey-test.pem”に出力

```
> .\token export --id test --pubkey --out pubkey-test.pem
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E)
-----
*** Public key( id= test ) found ***
--> pubkey-test.pem
```

既定では Pkcs#1 PEM で出力されます。

--ssh-key オプションを指定すると ssh-key 形式で出力します。

```
> .\token export --id test --pubkey --ssh-key
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E)
-----
*** Public key( id= test ) found ***
ssh-
rsaAAAA3NzaC1yc2EAAAADAQABAAQAC5Yx/elxWgTLq8YyrVH8eu4HsaD8t7Wz4Y0+FePWzu6i7q7LjihcT/M8Q9DEmKdmcmqnlz92zMUfaveWDuphyLTrKIClyqH61IEfzvWzHdTqx1
...
```

ssh-key を SSH サーバの authorized_key ファイルにコピー。SSH クライアント側ではダウンロードした puTTY-CAC の pageant.exe で公開鍵をエクスポートしたトークンへの各セク許可を設定してください。WinSCP や TeraTerm など pageant 対応の SSH クライアントはトークン公開鍵で SSH サーバにアクセスできるようになります。

--pkcs8 オプションを指定すると PKCS#8 PEM で出力できます。

```
.\token.exe export --id test --pubkey --pkcs8
-----BEGIN PUBLIC KEY-----
MIIBJjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5x1k5/mmuJorJrhZsQ0S
6jlqpXhM3bJJyoWUY5cl3elQRgWRdSKYvMSw3JtyDJL0xzuWBAN1wO0btG0RawHd
...
TvmodWhPXhyGxuz0EJX8XJOMCsRO8r5JYa78tCKWu9H8ITxuvcuGYWMPFVnjqnk
GQIDAQAB
-----END PUBLIC KEY-----
```

9. Enc & Dec コマンド

これらのコマンドはトークン内部の暗号鍵でファイルを暗号化、復号化します。

RSA 公開鍵で暗号化、RSA 秘密鍵で復号化

公開鍵で暗号化、秘密鍵で復号化します。暗号化可能なデータサイズは鍵サイズに依存します。2048 ビット (256 バイト) 鍵ではおおよそ最大 256 バイトです。

```
>token enc --id test --rsa --in data.txt --out data.enc.bin  
→ ファイル "data.txt" を ID "test" の公開鍵で暗号化、ファイル  
"data.enc.bin"に出力
```

```
>token dec --id test --rsa --in data.enc.bin --out data.dec.txt  
→ ファイル "data.enc.bin" を ID "test" の秘密鍵で復号化、ファイル  
"data.dec.txt"に出力
```

```
> .\token enc --id test --rsa --in data.txt --out data.enc.bin  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Encryption successful ***  
  
>.\token dec --id test --rsa --in data.enc.bin --out data.dec.txt  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Decryption successful ***
```

既定では出力はバイナリ形式になります。--base64 オプションを指定すると BASE64 形式で出力します。

```
>token enc --id test --rsa --in data.txt --out data.enc.bin --  
base64
```

復号化の入力はバイナリ、BASE64、どちらの形式でも構いません。本プログラムが自動検出します。

トークン内部で生成した RSA 公開鍵で暗号化したデータは、同じトークンの対応する秘密鍵でのみ復号化できます。別のトークンで復号化するには、外部 RSA 秘密鍵を複数のトークンにインポートします。

外部 RSA 秘密鍵、公開鍵があれば、トークンで暗号化したデータを OpenSSL で復号したり、OpenSSL で暗号化したデータをトークンで復号したりすることができます。

- OpenSSL で公開鍵暗号化、トークンで秘密鍵復号化
- トークンで公開鍵暗号化、OpenSSL で秘密鍵復号化
-

OpenSSL で公開鍵暗号化、トークンの秘密鍵復号化

```
>openssl pkeyutl -encrypt -inkey pubkey.pem -pubin  
--in data.txt -out data.enc.bin
```

→ 外部公開鍵“pubkey.pem”でファイル “data.txt”を暗号化、ファイル “data.enc.bin”に出力

```
>token dec --id test --rsa --in data.enc.bin -out data.dec.txt
```

→ ID “test”の秘密鍵でファイル data.enc.bin を復号化、ファイル data.dec.txt に出力

```

// generate a RSA private key
>.\OpenSSL genrsa rsakey1.pem 2048

// extract the public key from the private key
>.\OpenSSL rsa -in rsakey1.pem -pubout -out rsapubkey1.pem

// import the generated private key to a token
>.\token.exe import --id rsakey1 --privkey --in rsakey1.pem

>.\token list
id=rsakey1
- public key
- private key

//encrypt by the external public key
> .\OpenSSL pkeyutl -encrypt -inkey rsapubkey1.pem -pubin --in data.txt -out data.enc.bin

//decrypt by the token's private key
> .\token dec --id rsakey1 --in data.enc.bin --out data.dec.txt --rsa

```

トークンで公開鍵暗号化、OpenSSLで秘密鍵復号化

```
>token enc -id test --rsa --in data.txt --out data.enc. bin
```

→ ID “test”のRSA公開鍵で暗号化

```
>openssl pkeyutl -decrypt -inkey privkey.pem -in data.enc.bin -
out data.dec.txt
```

→ 外部RSA秘密鍵 “privkey.pem”でOpenSSL復号化


```

//encrypt by token's public key
>.\token enc --id rsakey1 --in data.txt --out data.enc.txt --rsa
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Encryption successful ***

//decrypt by the externa private key
>.\OpenSSL.exe pkeyutl -decrypt -inkey rsakey1.pem -in data.enc.txt -out data.dec.txt

```

AES 鍵で暗号化/復号化

トークン内で生成された鍵、インポートされた鍵の2種類の AES 秘密鍵があります。

トークン内で生成された鍵の利用

外部に取り出すことはできませんので、トークン内の AES 鍵で暗号化したデータは同じトークン内の AES 鍵でのみ復号化できます。

```

>token enc --id test --rsa --in data.txt --out data.bin
→ID “test”のAES鍵でファイル data.txt を暗号化、data.binに出力

```

```

> .\token enc --id test --aes --in data.txt --out data.bin
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--> By generated key
*** Encryption successful ***

```

ファイル “data.bin” を復号化できるのは、同じトークンの ID test の AES 鍵だけです。

```

token dec --id test --aes --in data.bin --out data.dec.txt
→ID test のAES鍵でファイル data.binを復号化、data.dec.txt に出力.

```

```

> .\token dec --id test --aes --in data.bin --out data.dec.txt
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--> By generated key
*** Decryption successful ***

```

インポートされた AES 鍵

外部 AES 鍵をトークンにインポートしたものです。複数のトークンに同じ AES 鍵をインポートできますので、同一鍵を複数のトークンで共有できます。また、OpenSSL で暗号、復号化することも可能です。

インポートされた AES 鍵による暗号化、復号化はトークン内で生成された AES 鍵による暗号化、復号化と同じ方法で行います。同じ外部 AES 鍵をインポートとしたトークン同士で暗号化、復号化が可能です。

以下の通りに AES 鍵をインポートしたとします。--pass オプションで指定したパスワード、--salt オプションで指定したソルトから PBKDF2(Password Based Key Derivation Function)アルゴリズムで AES 鍵と IV が生成されトークンに保存されます

```

> .\token import --id imported --aes --pass PASSWORD --salt 800963890e21bbf0 --p
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
salt=800963890E21BBF0
key=1BA005A59B4674610320760D2F4775CE
iv=1E24308D2054FF2BF412562FC970A30E
*** AES imported. Remember pass/salt ***

```

-p オプションが指定されているため、生成された鍵/IV が表示されています。

インポート鍵による暗号化と復号化

```
> .\token enc --id imported --aes --in data.txt --out data.bin
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
--> By imported key
--- File created(data.bin) ---
*** Encryption successful ***

> .\token dec --id imported --aes --in data.bin --out data.dec2.txt
>>reading tokens...over<<
Slot: 0 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )
-----
--> By imported key
--- File created(data.dec.txt) ---
*** Decryption successful ***
```

ファイル“data.txt” をシリアル番号 25FEB2B2DB013B0B のトークンのインポート鍵で暗号化、シリアル番号 597909FB7876EC19 のトークンのインポート鍵で復号化しています。どちらの鍵にも同じ -pass/--salt で AES 鍵をセットしています。

OpenSSL は PBKDF2 をサポートしています。インポート AES 鍵と同じパスワードとソルトを OpenSSL に与えれば、インポート鍵で暗号化したデータを復号化したり、OpenSSL で暗号化したデータをインポート鍵で復号化できます。

インポート鍵で暗号化したファイル“some.bin” を OpenSSL で復号化

```
>.\OpenSSL enc -aes-128-cbc -d -in some.bin -pbkdf2 -k PASSWORD -S 800963890E21BBF0 -out
some.dec.txt
```

-k (パスワード) と -S(ソルト) はインポート鍵と同じでなければなりません。また、-pbkdf2 オプションも指定してください (-d で復号化を指定)

OpenSSL AES 鍵暗号化

```
> .\OpenSSL enc -aes-128-cbc -e -in some.txt -pbkdf2 -k PASSWORD -S 800963890E21BBF0 -out  
some.bin
```

トークンのインポート鍵復号化

```
> .\token dec --id imported --aes --in some.bin --out some.dec1.txt  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
--> By imported key  
  
*** Decryption successful ***
```

トークンの PBKDF2 アルゴリズムの既定のメッセージダイジェスト(ハッシュ)は SHA256、また、繰り返し回数は 10,000 です。OpenSSL の PBKDF2 の既定も同じです。

もし、AES 鍵インポート時に異なる値をメッセージダイジェスト(ハッシュ)や繰り返し回数をセットしたら、OpenSSL にも同じ値を明示的にセットしなければなりません。

インポートでハッシュ = sha512,繰り返し回数 = 9999 を指定

```
> .\token import --id imported2 --aes --pass PASSWORD --salt 800963890e21bbf0 --md sha512 --  
iter 9999  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** AES imported. Remember pass/salt ***
```

このインポート鍵 ID imported 2 で暗号化したファイルを OpenSSL で復号化するには、同じハッシュ、繰り返し回数を指定しなければなりません。

トークンのインポート鍵で暗号化

```
> .\token enc --id imported2 --aes --in some.txt --out some.bin
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
--> By imported key

*** Encryption successful ***
```

OpenSSL で復号化

```
>.\OpenSSL enc -aes-128-cbc -d -pbkdf2 -k PASSWORD -S 800963890E21BBF0 -md sha512 -iter
9999 -in some.bin -out some.dec3.txt
```

10. 署名 / 検証

これらのコマンドはファイルの署名を作成したり、署名の正当性を検証します。

署名

```
>token sign --id test --in some.txt --sig myfile.sig --hash sha512
```

→ファイル“some.txt”の署名を ID “test” の RSA 秘密鍵で作成して、署名をファイル myfile.sig へ出力
既定では署名はバイナリ形式で出力。--base64 オプションで BASE64 形式で出力。

```
> .\token sign --id test --in some.txt --sig myfile.sig --hash sha512
>>reading tokens...over<<
hash -> sha512
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Signature file created (myfile.sig) successful ***

> dir myfile.sig
Mode                LastWriteTime         Length Name
----                -
-a----             2022/mm/dd   hh:mm           256 myfile.sig
```

署名するファイルのコンテンツのハッシュを指定 RSA 秘密鍵で暗号化したものが署名になります。既定はハッシュアルゴリズムは SHA256. 上の例では SHA512 を指定しています。

RSA 秘密鍵のサイズが 2048 ビット (256 バイト)なので署名サイズが 256 バイトになっています。

検証

```
>token.exe verify --id test --in some.txt --sig myfile.sig --hash sha512
```

→ ファイル myfile.sig に保存された署名が、RSA 秘密鍵がファイル “some.txt” に対して作成したものかどうか、対応する RSA 公開鍵で検証。署名ファイルはバイナリ形式、BASE64 形式どちらでも構いません。

```

> .\token.exe verify --id test --in some.txt --sig myfile.sig --hash sha512
>>reading tokens...over<<
hash -> sha512
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Verify ok ***

```

検証は、署名データを公開鍵で復号化、入力ファイル(some.txt)のハッシュと一致するかどうか確認します。署名時のハッシュと、検証時のハッシュは同じでなければなりません。

トークン内で生成した RSA 秘密鍵による署名は、対応するトークン内の公開鍵でのみ検証可能です。異なるトークンで署名を検証するには、外部 RSA 秘密鍵を複数トークンにインポートします。

トークンと OpenSSL による署名と検証

外部 RSA 秘密鍵があれば、OpenSSL と鍵をインポートしたトークンで相互に署名作成・検証が可能です。

1. OpenSSL 署名 / トークン検証

```
> .\openssl dgst --sha512 -sign rsakey1.pem -sha512 -out myfile.sig data.txt
```

→ファイル data.txt のSHA512ハッシュ作成、ハッシュを秘密鍵 rsakey1.pem で署名

```
> token verify --id imported --hash sha512 --in some.txt --sig myfile.sig
```

→署名 (myfile.sig)を ID“imported”の公開鍵で検証

```

// sign using OpenSSL
>.\openssl dgst --sha512 -sign rsakey1.pem -sha512 -out myfile.sig data.txt

//verify by token
.\token verify --id rsakey1 --hash sha512 --in data.txt --sig myfile.sig
>>reading tokens...over<<
hash -> sha512
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )
-----
*** Verify ok ***

```

2. トークン署名 / OpenSSL 検証

```
> token sign --id rsakey1 --hash sha512 --in data.txt --sig myfile.sig  
→ファイル“data.txt”をハッシュ SHA512 で署名、myfile.sigに出力
```

```
>openssl dgst --sha512 -verify rsapubkey1.pem -sha512 -signature  
myfile.sig some.txt
```

→署名“myfile.sig”が公開鍵 rsapubkey1.pem に対応する秘密鍵で some.txt に対する署名であることを検証。ハッシュアルゴリズムは署名時のものと同じ SHA512

OpenSSL では署名はバイナリ形式でなければなりません。BASE64 形式になっていたら変換してください。

```
>openssl base64 -d -a -in myfile.sig -out myfile.sig.bin  
→BASE64 形式署名 myfile.sig をバイナリ形式に変換
```

```
// sign by token  
>.\token sign --id rsakey1 --hash sha512 --in data.txt --sig myfile.sig  
>>reading tokens...over<<  
hash -> sha512  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Signature file created (myfile.sig) successful ***  
  
//verify by OpenSSL  
.\openssl dgst --sha512 -verify rsapubkey1.pem -sha512 -signature myfile.sig data.txt  
Verified OK
```

トークン内の証明書が特定CAにより発行されたものかどうかを検証

CA の証明書があれば、トークン内の証明書がそのCAによって発行されたものかどうかを検証できます。


```
>token verify -id rsakey1 -in cacert.pem -cacert  
→ID “rsakey1” の証明書が CA によって発行されたかどうかを検証(CAの証明書  
は cacert.pem)
```

本プログラムは最初に証明書の証明書発行体のDN(Distinguished Name)が、CA 証明書の Subject DN と一致することを確認後、証明書内の署名を CA 証明書内の公開鍵で検証します。

```
.\token verify --id rsakey1 --in cacert.pem --cacert  
>>reading tokens...over<<  
  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
CA Certificate Subject DN : /CN=ribig/O=RiBiG/OU=system/L=/S=Kanagawa/C=JP  
Certificate Issuer DN : /CN=ribig/O=RiBiG/OU=system/L=/S=Kanagawa/C=JP  
*** Verified successfully ***
```

11. INIT コマンド

このコマンドはトークンを初期化します。初期化には SO PIN が必要です。

```
>token init --sopin (current sopin) --newpin admin123 --pin 123456  
→トークン初期化. 初期化後のトークンの PIN は  
    > SO Pin: admin123  
    > User PIN: 123456
```

```
>.\token init --sopin admin123 --newpin admin123 --pin 123456  
>>reading tokens...over<<  
  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Token Successfully Initialized ***  
( Default USER Pin : 123456 )  
( Default SO Pin : admin123 )
```

既定のトークンラベルはシリアル番号になります。--label オプションでラベルを指定できます。

```
>token init --sopin ribig --label mytoken --newpin admin123 --pin
```

```
>.\token init --sopin admin123 --label mytoken --newpin admin123 --pin 123456  
>>reading tokens...over<<  
  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=25FEB2B2DB013B0B, Model=E )  
-----  
*** Token Successfully Initialized ***  
( Default USER Pin : 123456 )  
( Default SO Pin : admin123 )  
  
> .\token.exe list  
>>reading tokens...over<<  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mytoken, Model=E )  
-----
```

123456

12. PIN コマンド

このコマンドは USER PIN と SO PIN の変更、USER PIN のリセットを行います。

SO PIN の変更には 現在の SO PIN を与える必要があります。USER PIN の変更にも現在の USER PIN が必要です。現在の SO PIN があれば、USER PIN をリセットできます。

SO Pin 変更

```
>token.exe pin --change_so -sopin admin1234 --newpin  
admin123
```

→ SO Pin を admin1234 から admin123 に変更

```
>.\token.exe pin --change_so --sopin admin1234 --newpin admin123  
>>reading tokens...over<<  
  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mytoken, Model=E )  
-----  
*** SO Pin Successfully Changed ***
```

設定ファイル 25FEB2B2DB013B0B.ini の SOキーは更新されます。

[PIN]

USER=123456

SO=admin123

USER PIN 変更

```
>token.exe pin --change --pin 123456 --newpin 12345678
```

➔ User Pin を 123456 から 12345678 に変更

```
>.\token.exe pin --change --pin 123456 --newpin 12345678  
>>reading tokens...over<<  
  
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mytoken, Model=E )  
-----  
*** USER Pin Successfully Changed ***
```

設定ファイル 25FEB2B2DB013B0B.ini の USERキーは更新されます。

```
[PIN]
USER=12345678
SO=admin123
```

USER PINリセット

```
>token.exe pin --reset --sopin admin123 --newpin 123456
→ USER PINを 123456にリセット
```

```
.\token.exe pin --reset --sopin admin123 --newpin 123456
>>reading tokens...over<<

Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mytoken, Model=E )
-----
*** USER Pin Successfully Reset ***
```

設定ファイル 25FEB2B2DB013B0B.ini の USERキーは更新されます。

***デモ版はトークンの User Pinが123456を前提としています。トークンの User Pinを他の値に変更すると Login Error が発生しはじめます。

```
>token.exe list
>>reading tokens...over<<

Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mytoken, Model=E )
-----
xxx Login Error xxx
```

13. Set コマンド

このコマンドはトークンのラベルを変更します。

```
>token set -label (新しいラベル)
```

```
> .\token.exe set --label new_label
>>reading tokens...over<<
Slot: 0 - Serial:69050A34778EB5B5 ( Label=mToken CryptolD, Model=E )
-----
*** Label set successfully (new_label) ***

> .\token.exe list
>>reading tokens...over<<
Slot: 0 - Serial:69050A34778EB5B5 ( Label=new_label, Model=E )
-----
```

--label オプションに. (ドット) and ..(2つのドット)を指定すると . (ドット) はシリアル番号、..(2つのドット)はトークンのモデル名に展開されます。

```
>token set --label . (トークンシリアル番号)
>token set --label .. (トークンモデル名)
```

```
> .\token.exe set --label .
>>reading tokens...over<<
Slot: 0 - Serial:69050A34778EB5B5 ( Label=new_label, Model=E )
-----
*** Label set successfully (69050A34778EB5B5) ***

> .\token.exe set --label ..
>>reading tokens...over<<
Slot: 0 - Serial:69050A34778EB5B5 ( Label=69050A34778EB5B5, Model=E )
-----
*** Label set successfully (E) ***
```

ラベル名の代わりに正規表現を指定することもできます。書式は以下の通りです。

```
--label /マッチ regex/置き換え regex
```

正規表現は “/” (スラッシュ) で始まります。 マッチ regex は “シリアル番号:ラベル” に対して適用されます。もし、シリアル番号 69050A34778EB5B5、ラベル “mToken CryptoID” のトークンであれば、

```
“69050A34778EB5B5:mToken CryptoID”
```

に対してマッチ regex が適用され、一致する部分が置き換え regex で置き換えられます。

PowerShell ではバックティック(`) がエスケープ文字です。

```
> .\token.exe set --label "/(.*):[A-Za-z0-9]+\s.*"/$2-$1"  
>>reading tokens...over<<  
Slot: 0 - Serial:69050A34778EB5B5 ( Label=mToken CryptoID, Model=E )  
-----  
*** Label set successfully (mToken-69050A34778EB5B5) ***
```

マッチ regex, (.*):[A-Za-z0-9]+\s.* は “69050A34778EB5B5:mToken CryptoID”

全体に一致するため、すべてが置き換え regex で置き換えられます。

\$2 は mToken, \$1 はシリアル番号なので、最終的にラベルは “mToken-69050A34778EB5B5”。

置き換え reg 内で # (シャープ) 文字を使うと特別な意味を持ちます。本プログラムは # をプログラムが設定ファイルで管理するシリアル番号に置き換えます。# を見つけると、設定ファイルから現在のシリアル番号の値を読み込み、# と置き換え、設定ファイルのシリアル番号の値を増加させます。また、設定ファイルでシリアル番号の書式 (printf と同じ) を指定できます。

Token.INI

```
[TokenLabel]  
SerialNo=3  
Format=%05d
```

現在のシリアル番号は 3. %05d という書式指定子により 00003 として出力されます

```
> tokene set --label "/(.*):([A-Za-z0-9]+)¥s.*\/`$2-#"
```

→置き換え regex の#は 00003 に置き換えられます。設定ファイルの SerialNo は 4 に更新。次の置き換え regex の#は 00004 に置き換えられます。複数トークンが接続されていると、シリアルが1ずつ増加したラベルがセットされます。

```
> .\token.exe set --label "/(.*):([A-Za-z0-9]+)\s.*\/`$2-#"
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken CryptoID, Model=E )
-----
*** Label set successfully (mToken-00003) ***

Slot: 1 - Serial:69050A34778EB5B5 ( Label=mToken CryptoID, Model=E )
-----
*** Label set successfully (mToken-00004) ***

Slot: 2 - Serial:597909FB7876EC19 ( Label=mToken CryptoID, Model=E )
-----
*** Label set successfully (mToken-00005) ***
```

14. トークンフィルタ

複数のトークンが接続されていると、すべてのトークンに対してコマンドが実行されます。

LIST コマンドを実行すると、すべてのトークンがリスト表示されます。

```
> .\token.exe list
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
id=rsaKey1
- public key
- private key

id=aeskey1
- AES Key

Slot: 1 - Serial:69050A34778EB5B5 ( Label=mToken-00004, Model=E )
-----

Slot: 2 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )
-----
```

他のコマンドも同様です。

複数トークンを接続していても、特定のトークンのみに対してコマンドを実行することもできます。

--slot オプション

接続トークンは固有の Slot id が割り当てられます。コマンドラインで対象となるトークンの Slot Id を指定します。

>token list --slot 2 Slot id = 2 のトークン選択
>token list --slot 1,2 複数トークンの選択も可能

```
> .\token.exe list --slot 0,2
>>reading tokens...over<<

Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
id=rsakey1
- public key
- private key

id=aeskey1
- AES Key

Slot: 2 - Serial:597909FB7876EC19 ( Label=mToken-00005, Model=E )
-----
```

--token オプション

トークンのラベル、シリアル番号で選択できます。

--token ラベル:シリアル番号

ラベルとシリアル番号に完全一致したトークンを選択します。

ラベルのみ指定することもできます。

>token list -token "mToken CryptoID"
→ ラベル "mToken CryptoID"をもったトークン選択

シリアル番号のみの指定も可能です(コロン+シリアル番号)

>token list -token : 25FEB2B2DB013B0B

→ シリアル番号 25FEB2B2DB013B0B のトークンを選択

```
> .\token.exe export --id rsakey1 --pubkey --token :25FEB2B2DB013B0B
>>reading tokens...over<<
Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
*** Public key( id= rsakey1 ) found ***
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAttrxdN7YN0ttCUUwDLvxFlrPDTkdLWK0KQbrviNUwBT3OQsspcePV
..
cJHxHcV2QcUMmc4UrCG9ItQIgPxlxrZ5RQIDAQAB
-----END RSA PUBLIC KEY-----

> .\token.exe list --token :25FEB2B2DB013B0B
>>reading tokens...over<<

Slot: 0 - Serial:25FEB2B2DB013B0B ( Label=mToken-00003, Model=E )
-----
id=rsakey1
- public key
- private key

id=aeskey1
- AES Key
```

以上の方法は完全一致しなければなりません。部分一致させるには、正規表現を指定します。
正規表現は /(スラッシュ)で始まります。

```
>token list --token /^mToken-.+
→ ラベルが“mToken-“始まるトークンを選択
```

```
>token csr -id test -dir . --token /^mToken-.+
→ラベルが“mToken-“始まるトークンを選択
```

付録 1

本プログラムで PIN とラベルを変更すると、新しい PIN /ラベルは設定ファイル INI に書き込まれます。

多数のトークンを INI ファイルで管理するのは面倒なため、INI ファイルだけでなくデータベースに PIN/ラベルを書き出すことも可能になっています。データベースは MariaDB

Token.ini 設定ファイルにデータベース接続、利用するデータベース名を指定します。

[Db]	
ip=127.0.0.1	DB サーバの IP アドレス
dbName=tokenDb	データベース名
user=test	データベースユーザ名
pass=testpass	データベースユーザパスワード
#port=3306	データベースポート (既定 3306)

- データベース内に“tokens” という名前のテーブル、“update_val” という名前のストアドプロシージャが必要です。
- 指定ユーザはテーブルに対して“Select, Insert, Update” 権限、ストアドプロシージャ実行権限(“Execute” global 権限)が必要です

次のページに MariaDB に必要なデータベース、テーブル、ストアドプロシージャを作成する sql を示します

```

-- create a database named tokendb
CREATE DATABASE IF NOT EXISTS `tokendb` /*!40100 DEFAULT
CHARACTER SET utf8mb4 */;
USE `tokendb`;

-- create a table named "tokens"
CREATE TABLE IF NOT EXISTS `tokens` (
  `SERIALNO` varchar(50) NOT NULL DEFAULT '0',
  `USER_PIN` varchar(50) DEFAULT NULL,
  `SO_PIN` varchar(50) DEFAULT NULL,
  `LABEL` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`SERIALNO`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- create a stored procedure named update_val
DELIMITER //
CREATE PROCEDURE `update_val`(
  IN `param_serialno` VARCHAR(24),
  IN `param_column` VARCHAR(50),
  IN `param_val` VARCHAR(32)
)
BEGIN
DECLARE ser VARCHAR(32);
SELECT serialno into ser FROM tokens WHERE serialno=param_serialno;

if ser IS null then
# insert
  if param_column = "user_pin" then #user pin
    INSERT INTO tokens (serialno, user_pin) VALUES ( param_serialno,
param_val );
  ELSE
    if param_column = "so_pin" then
      INSERT INTO tokens (serialno, so_pin) VALUES ( param_serialno,
param_val );
    ELSE

```

```

        INSERT INTO tokens (serialno, label) VALUES ( param_serialno,
param_val );
        END if;
    END if;

ELSE
#update
    if param_column = "user_pin" then #user pin
        UPDATE tokens SET user_pin = param_val WHERE serialno =
param_serialno;
    else
        if param_column = "so_pin" then
            UPDATE tokens SET so_pin = param_val  WHERE serialno =
param_serialno;
        else
            UPDATE tokens SET label = param_val  WHERE serialno =
param_serialno;
        END if;
    END if;
END if;

END//
DELIMITER ;

```