署名済み Tcl スクリプト

署名付き TCL スクリプト機能を使用すると、デジタル署名を生成する証明書を作成し、そのデジタル署名を使用してツールコマンド言語（TCL）スクリプトに署名することが可能になります。この機能は、既存のスクリプトおよび証明書でも動作します。デジタル署名の認証が確認されてから、Tcl インタープリタへの信頼できるアクセスでスクリプトが実行されます。スクリプトにデジタル署名がない場合、そのスクリプトは信頼できないスクリプト用の限定モードで実行されるか、まったく実行されません。

署名済み Tcl スクリプトに関する前提条件

この機能が動作するには、Cisco Public Key Infrastructure（PKI）設定のトラストポイント コマンドを有効にする必要があります。

署名付き TCL スクリプトの制約事項

この機能が動作するには、次を実行している必要があります。

- Cisco IOS 暗号イメージ
- OpenSSL Version 0.9.7a 以降
- Expect

署名済み Tcl スクリプトについて

署名済み Tcl スクリプト機能は Tcl スクリプトにセキュリティを導入します。この機能を使用すると、デジタル署名を生成する証明書を作成し、そのデジタル署名を使用して Tcl スクリプトに署名することが可能になります。この証明書は、Tcl スクリプトを実行する前にそれらを検査します。スクリプトに Cisco 発行のデジタル証明書が含まれているかどうかを確認します。さらに、第三者がデジタル署名でスクリプトに署名することもできます。独自に社内で開発した TCL スクリプトに署名したい場合や、サード パーティ製が開発したスクリプトを使用したい場合もあります。スクリプトに正しいデジタル署名が含まれている場合は本物であると見なされ、Tcl インタープリタにフル アクセスで実行されます。スクリプトにデジタル署名がない場合、そのスクリプトはセーフ Tcl モードという限定されたモードで実行されるか、またはまったく実行されません。

署名付き Tcl スクリプトを作成し、使用するには、次の概念を理解する必要があります。

Cisco PKI

Cisco PKI を使用すると、IP セキュリティ（IPSec）、セキュア シェル（SSH）、セキュア ソケット レイヤ（SSL）などのセキュリティ プロトコルをサポートする証明書管理を実現できます。PKI は以下のエンティティで構成されています。

- セキュアなネットワークで通信する複数のピア
- 証明書を発行および維持する認証局（CA）を最低 1 つ
- デジタル証明書（証明書の有効期間、ピアの ID 情報、セキュアな通信に使用する暗号キー、CA 発行のシグニチャなどで構成）
- 登録要求を処理し CA の負荷を軽減する登録局（RA）（任意）
- 証明書失効リスト（CRL）を配信するメカニズム（Lightweight Directory Access Protocol（LDAP）、HTTP など）

PKI を使用すると、セキュアなデータ ネットワークで暗号化情報と ID 情報を配信、管理、失効するためのスケーラブルでセキュアなメカニズムを実現できます。セキュアな通信に関係するルーティング デバイスはすべて、あるプロセスを経て PKI に登録されます。そのプロセスでは、ルーティング デバイスが Rivest, Shamir, and Adelman（RSA）キー ペア（秘密キーが 1 つ、公開キーが 1 つ）を生成し、信頼されているルーティング デバイス（CA またはトラストポイントともいいます）でキーの ID を確認します。

各ルーティング デバイスが PKI に登録されると、PKI のすべてのピア（エンド ホストともいいます）は、CA が発行したデジタル証明書を付与されます。セキュアな通信セッションをネゴシエーションする必要があるときは、ピアはデジタル証明書を交換します。ピアは証明書内の情報を基に他のピアの ID を確認し、証明書内の公開キーを使って、暗号化されたセッションを確立します。

RSA キーペア

RSA キー ペアは、公開キーと秘密キーで構成されます。PKI を設定する場合、証明書登録要求に公開キーを含める必要があります。証明書が付与された後、ピアが公開キーを使用して、デバイスに送信されるデータを暗号化できるように、公開キーが証明書に組み込まれます。秘密キーはデバイスに保持され、ピアによって送信されたデータの復号化と、ピアとネゴシエーションするときの、トランザクションのデジタル署名に使用されます。

RSAキーペアには、キーのモジュラス値が含まれています。モジュラス値に応じて、RSAキーのサイズが決まります。モジュラス値が大きいほど、RSA キーの安全性が高まります。ただし、モジュラス値が大きくなると、キーの生成にかかる時間が長くなり、キーのサイズが大きくなると暗号化処理および復号化処理にかかる時間が長くなります。

証明書およびトラストポイント

認証局（CA。トラストポイントともいいます）は、証明書要求を管理し、参加ネットワークデバイスに証明書を発行します。証明書要求の管理や証明書発行などのサービスにより、参加デバイスを一元的に管理します。またこれらのサービスによって受信者は、明示的に信頼してアイデンティティを確認し、デジタル証明書を作成できます。PKIの動作を開始する前に、CAは独自の公開キー ペアを生成し、自己署名 CA 証明書を作成します。その後、CA は、証明書要求に署名し、PKI に対してピア登録を開始できます。

CAは、サードパーティのCAベンダーが提供するCAを使用するか、内部のCA、つまりCisco証明書サーバーを使用します。

署名済み Tcl スクリプトの設定方法

キー ペアの生成

キーペアは、秘密キーと公開キーで構成されます。秘密キーは公開されず、作成者のみがアクセス可能にすることを意図しています。公開キーは秘密キーから生成され、公開されることを前提としています。

キーペアを生成するには、**openssl genrsa** コマンドを使用した後、**openssl rsa** コマンドを使用します。

手順の概要

1. **openssl genrsa -out** *private-key-file bit-length*
2. **ls -l**
3. **openssl rsa -in** *private-key-file* **-pubout -out** *public-key-file*
4. **ls -l**

手順の詳細

ステップ 1 **openssl genrsa -out** *private-key-file bit-length*

このコマンドは、*bit-length* ビット長の秘密キーを生成し、そのキーを *private-key-file* ファイルに書き込みます。

```
Host% openssl genrsa -out privkey.pem 2048
```

例：

```
Generating RSA private key, 2048 bit long modulus
.........+++
.....................................................................+++
e is 65537 (0x10001)
```

ステップ 2 **ls -l**

このコマンドは、現在のディレクトリ内の各ファイルに対する詳細な情報（権限、所有者、サイズ、最終変更日時を含む）を表示します。

例：

```
Host% ls -l

total 8
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
```

privkey.pem ファイルには、**openssl genrsa** コマンドを使用して生成した秘密キーが含まれています。

ステップ 3 **openssl rsa -in** *private-key-file* **-pubout -out** *public-key-file*

このコマンドは、*private-key-file* ファイル内の指定された秘密キーに基づいて公開キーを作成し、その公開キーを *public-key-file* ファイルに書き込みます。

例：

```
Host% openssl rsa -in privkey.pem -pubout -out pubkey.pem

writing RSA key
```

ステップ 4 **ls -l**

このコマンドは、現在のディレクトリ内の各ファイルに対する詳細な情報（権限、所有者、サイズ、最終変更日時を含む）を表示します。

例：

```
Host% ls -l

total 16
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

pubkey.pem ファイルには、**openssl rsa** コマンドを使用して生成された公開キーが含まれます。

証明書の生成

証明書を生成するには、次のタスクを実行します。X.509 証明書を生成するには、**openssl req** コマンドを使用します。

手順の概要

1. **openssl req -new -x509 -key** *private-key-file* **-out** *certificate-file* **-days** *expiration-days*
2. **ls -l**

手順の詳細

ステップ 1 **openssl req -new -x509 -key** *private-key-file* **-out** *certificate-file* **-days** *expiration-days*

このコマンドは、*private-key-file* ファイルに保存された秘密キーにフルアクセスできる X.509 証明書を作成し、*certificate-file* ファイルに証明書を保存します。証明書は *expiration-days* 日内に期限が切れるように設定されます。

コマンドを実行するには、プロンプトが表示された時点で次の識別名（DN）情報を入力します。

- 国名
- 州、行政区分（都道府県）名
- 組織名
- 組織部署名
- 共通名
- メール アドレス

各プロンプトの角括弧で囲まれたテキストは、Enter を押す前に値を入力しなかった場合に使用されるデフォルト値を示します。

次に、privkey.pem ファイル内の秘密キーに対するフル アクセスを持つ X.509 証明書を生成する方法の例を示します。証明書は cert.pem ファイルに書き込まれ、生成日の 1095 日後に期限切れになります。

例：

```
Host% openssl req -new -x509 -key privkey.pem -out cert.pem -days 1095

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value, If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:US
```

```
State or Province Name (full name) [Berkshire]:California

Locality Name (eg, city) [Newbury]:San Jose

Organization Name (eg, company) [My Company Ltd]:Cisco Systems, Inc.

Organizational Unit Name (eg, section) []:DEPT_ACCT

Common Name (eg, your name or your server's hostname) []:Jane

Email Address []:janedoe@company.com
```

ステップ 2 **ls -l**

このコマンドは、現在のディレクトリ内の各ファイルに対する詳細な情報（権限、所有者、サイズ、最終変更日時を含む）を表示します。

例：

```
Host% ls -l

total 24
-rw-r--r--   1 janedoe eng12       1659 Jun 12 15:01 cert.pem
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

cert.pem ファイルには、**openssl req** コマンドを使用して作成された X.509 証明書が含まれています。

Tcl スクリプトの署名

Tcl スクリプトに署名するには、次のタスクを実行します。TCL ファイル、および OpenSSL ドキュメントの出力に、pkcs7（PKCS#7）フォーマットで署名する必要があります。

Tcl ファイルに署名するには、**openssl smime** コマンドと **-sign** キーワードを使用します。

手順の概要

1. **openssl smime -sign -in** *tcl-file* **-out** *signed-tcl-file* **-signer** *certificate-file* **-inkey** *private-key-file* **-outform DER -binary**
2. **ls -l**

手順の詳細

ステップ 1 **openssl smime -sign -in** *tcl-file* **-out** *signed-tcl-file* **-signer** *certificate-file* **-inkey** *private-key-file* **-outform DER -binary**

このコマンドは、*certificate-file* に保存されている証明書と、*private-key-file* に保存されている秘密キーを使用して Tcl ファイル名 *tcl-file* に署名し、署名済みの Tcl ファイルを *signed-tcl-file* ファイルに DER PKCS#7 形式で書き込みます。

例：

```
Host% openssl smime -sign -in hello -out hello.pk7 -signer cert.pem -inkey privkey.pem -outform DER
 -binary
```

ステップ 2 ls -l

このコマンドは、現在のディレクトリ内の各ファイルに対する詳細な情報（権限、所有者、サイズ、最終変更日時を含む）を表示します。

例：

```
Host% ls -l

total 40
-rw-r--r--   1 janedoe eng12       1659 Jun 12 15:01 cert.pem
-rw-r--r--   1 janedoe eng12        115 Jun 13 10:16 hello
-rw-r--r--   1 janedoe eng12       1876 Jun 13 10:16 hello.pk7
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

hello.pk7 ファイルには、hello という名前の未署名の TCL ファイルから **openssl smime** コマンドと cert.pem ファイル内の X.509 証明書を使用して作成された、署名済み Tcl ファイルが含まれています。

署名の確認

署名がデータと一致していることを確認するには、**openssl smime** コマンドと **-verify** キーワードを使用して次のタスクを実行します。Tcl の元の内容を入力ファイルに提供する必要があります。これは、ファイルに元の内容が含まれていないためです。

手順の概要

1. **openssl smime -verify -in** *signed-tcl-file* **-CAfile** *certificate-file* **-inform DER -content** *tcl-file*
2. **ls -l**

手順の詳細

ステップ 1 openssl smime -verify -in *signed-tcl-file* -CAfile *certificate-file* -inform DER -content *tcl-file*

このコマンドは、*certificate-file* 内の信頼認証局（CA）証明書を使用して DER PKCS#7 形式で *signed-tcl-file* に保存されている署名付き Tcl ファイルを確認した後、デタッチされた内容を *tcl-file* ファイルに書き込みます。

次に、入力ファイルの hello.pk7 を使用して署名を確認する例を示します。

例：

```
Host% openssl smime -verify -in hello.pk7 -CAfile cert.pem -inform DER -content hello

puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
```

```
puts "tcl_interactive = $tcl_interactive"
Verification successful
```

（注） SSL コマンドページでは、**-in** *filename* が暗号化または署名される入力メッセージか、復号または確認される MIME メッセージとして説明されています。詳細については、http://www.openssl.org/ を参照してください。

ステップ 2 **ls -l**

このコマンドは、現在のディレクトリ内の各ファイルに対する詳細な情報（権限、所有者、サイズ、最終変更日時を含む）を表示します。

例：

```
Host% ls -l

total 40
-rw-r--r--   1 janedoe eng12       1659 Jun 13 10:18 cert.pem
-rw-r--r--   1 janedoe eng12        115 Jun 13 10:17 hello
-rw-r--r--   1 janedoe eng12       1876 Jun 13 10:16 hello.pk7
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

hello ファイルには、**openssl smime** コマンドを **-verify** キーワードで実行して、署名付き Tcl ファイル hello.pk7 からデタッチされた内容が含まれています。確認に成功した場合、署名者の証明書が cert.pem ファイルの X.509 証明書に書き込まれます。

シグニチャの非バイナリデータへの変換

バイナリから非バイナリ データにシグニチャを変換するには、次のタスクを実行します。

手順の概要

1. **xxd -ps** *signed-tcl-file* > *nonbinary-signature-file*
2. **#Cisco Tcl Signature V1.0** を最初の行に表示するスクリプトを作成し、コメント文字（#）を入力ファイルの各行の先頭に挿入し、入力ファイルの名前にテキスト文字列「_sig」を追加して形成された名前のファイルに各行を書き込みます。
3. 非バイナリ シグニチャ ファイルを含むファイルの名前（*nonbinary-signature-file*）を入力引数として指定して、スクリプトを実行します。
4. **ls -l**
5. **cat** *signed-tcl-file* *commented-nonbinary-signature-file* > *signed-tcl-script*
6. **cat** *signed-tcl-script*

手順の詳細

ステップ 1 **xxd -ps** *signed-tcl-file* > *nonbinary-signature-file*

このコマンドは、*signed-tcl-file* のシグニチャをバイナリから非バイナリのデータに変換して *nonbinary-signature-file* ファイルに 16 進ダンプとして保存します。

例：

```
Host% xxd -ps hello.pk7 > hello.hex
```

ステップ 2 **#Cisco Tcl Signature V1.0** を最初の行に表示するスクリプトを作成し、コメント文字（#）を入力ファイルの各行の先頭に挿入し、入力ファイルの名前にテキスト文字列「_sig」を追加して形成された名前のファイルに各行を書き込みます。

次に、**cat** コマンドを使用して、my_append という名前のスクリプトファイルの内容を表示する例を示します。

例：

```
Host% cat my_append

#!/usr/bin/env expect
set my_first {#Cisco Tcl Signature V1.0}
set newline {}
set my_file [lindex $argv 0]
set my_new_file ${my_file}_sig
set my_new_handle [open $my_new_file w]
set my_handle [open $my_file r]
puts $my_new_handle $newline
puts $my_new_handle $my_first
foreach line [split [read $my_handle] "\n"]  {
   set new_line {#}
   append new_line $line
   puts $my_new_handle $new_line
}

close $my_new_handle
close $my_handle
```

ステップ 3 非バイナリ シグニチャ ファイルを含むファイルの名前（*nonbinary-signature-file*）を入力引数として指定して、スクリプトを実行します。

この例では、my_append スクリプトが、入力として指定された非バイナリ シグニチャ ファイル hello.hex を使用して実行されています。出力ファイルには、hello.hex_sig という名前が付けられます。

例：

```
Host% my_append hello.hex
```

ステップ 4 **ls -l**

このコマンドは、現在のディレクトリ内の各ファイルに対する詳細な情報（権限、所有者、サイズ、最終変更日時を含む）を表示します。

例：

```
Host% ls -l

total 80
-rw-r--r--   1 janedoe eng12       1659 Jun 13 10:18 cert.pem
-rw-r--r--   1 janedoe eng12        115 Jun 13 10:17 hello
```

```
-rw-r--r--   1 janedoe eng12       3815 Jun 13 10:20 hello.hex
-rw-r--r--   1 janedoe eng12       3907 Jun 13 10:22 hello.hex_sig
-rw-r--r--   1 janedoe eng12       1876 Jun 13 10:16 hello.pk7
-rwxr--r--   1 janedoe eng12        444 Jun 13 10:22 my_append
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

hello.hex ファイルには、署名済み Tcl ファイル hello.pk7 のバイナリ シグニチャから変換された非バイナリデータ（16 進数のダンプとして格納）が含まれています。my_append ファイルには、入力ファイルの各行の先頭にコメント文字を挿入するスクリプトが含まれています。この hello.hex_sig ファイルは、非バイナリ シグニチャ ファイルで my_append スクリプトを実行して作成されたファイルです。

ステップ 5 **cat** *signed-tcl-file commented-nonbinary-signature-file* > *signed-tcl-script*

このコマンドは非バイナリ シグニチャ ファイル（*commented-nonbinary-signature-file*）の内容を、DER PKCS#7 形式で保存された署名済みの Tcl ファイル（*signed-tcl-file* ファイル）に追加します。連結された出力が *signed-tcl-script* ファイルに書き込まれます。

例：

```
Host% cat hello hello.hex_sig > hello.tcl
```

ステップ 6 **cat** *signed-tcl-script*

このコマンドは、署名済み Tcl ファイルと非バイナリ シグニチャ ファイルから分離された内容を連結した *signed-tcl-script* ファイルの内容を表示します。

例：

```
Host% cat hello.tcl

puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
puts "tcl_interactive = $tcl_interactive"
#Cisco Tcl Signature V1.0
#3082075006092a864886f70d010702a08207413082073d020101310b3009
#06052b0e03021a0500300b06092a864886f70d010701a08204a13082049d
#30820385a003020102020100300d06092a864886f70d0101040500308195
#310b300906035504061302555331133011060355040813 0a43616c69666f
#726e69613111300f060355040713 0853616e204a6f7365311c301a060355
#040a1313436973636f2053797374656d732c20496e632e310e300c060355
#040b13054e53535447310d300b060355040313044a6f686e3121301f0609
#2a864886f70d01090116126a6c6175746d616e40636973636f2e636f6d30
#1e170d3037303631323232303134335a170d3130303631313232303133
#5a308195310b3009060355040613025553311330110603550408130a4361
#6c69666f726e69613111300f060355040713 0853616e204a6f7365311c30
#1a060355040a1313436973636f2053797374656d732c20496e632e310e30
#0c060355040b13054e53535447310d300b060355040313044a6f686e3121
#301f06092a864886f70d01090116126a6c6175746d616e40636973636f2e
#636f6d30820122300d06092a864886f70d0101010500 0382010f00308201
#0a0282010100a751eb5ec1f3009738c88a55987c07b759c36f3386342283
#67ea20a89d9483ae85e0c63eeded8ab3eb7a08006689f09136f172183665
#c971099ba54e77ab47706069bbefaaab8c50184396350e4cc870c4c3f477
#88c55c52e2cf411f05b59f0eaec0678ff5cc238fdce2263a9fc6b6c244b8
#ffaead865c19c3d3172674a13b24c8f2c01dd8b1bd491c13e84e29171b85
#f28155d81ac8c69bb25ca23c2921d85fbf745c106e7aff93c72316cbc654
#4a34ea88174a8ba7777fa60662974e1fbac85a0f0aeac925dba6e5e850b8
#7caffce2fe8bb04b61b62f532b5893c081522d538005df81670b931b0ad0
```

```
#e1e76ae648f598a9442d5d0976e67c8d55889299147d0203010001a381f5
#3081f2301d0603551d0e04160414bc34132be952ff8b9e1af3b93140a255
#e54a667c3081c20603551d230481ba3081b78014bc34132be952ff8b9e1a
#f3b93140a255e54a667ca1819ba48198308195310b300906035504061302
#5553311330110603550408130a43616c69666f726e69613111300f060355
#0407130853616e204a6f7365311c301a060355040a1313436973636f2053
#797374656d732c20496e632e310e300c060355040b13054e53535447310d
#300b060355040313044a6f686e3121301f06092a864886f70d0109011612
#6a6c6175746d616e40636973636f2e636f6d820100300c0603551d130405
#30030101ff300d06092a864886f70d01010405000382010100c83c1b074
#6720929c9514af6d5df96f0a95639f047c40a607c83d8362507c58fa7f84
#aa699ec5e5bef61b2308297a0662c653ff446acfbb6f5cb2dd162d939338
#a5e4d78a5c45021e5d4dbabb8784efbf50cab0f5125d164487b31f5cf933
#a9f68f82cd111cbab1739d7f372ec460a7946882874b0a0f22dd53acbd62
#a944a15e52e54a24341b3b8a820f23a5bc7ea7b2278bb56838b8a4051926
#af9c167274ff8449003a4e012bcf4f4b3e280f85209249a390d14df47435
#35efabce720ea3d56803a84a2163db4478ae19d7d987ef6971c8312e280a
#aac0217d4fe620c6582a48faa8ea5e3726a99012e1d55f8d61b066381f77
#4158d144a43fb536c77d6a3182027730820273020101301308195310b
#300906035504061302555331133011060355040813a43616c69666f726e
#69613111300f060355040713085361e204a6f7365311c301a060355040a
#1313436973636f2053797374656d732c20496e632e310e300c060355040b
#13054e53535447310d300b060355040313044a6f686e3121301f06092a86
#4886f70d01090116126a6c6175746d616e40636973636f2e636f6d020100
#300906052b0e03021a0500a081b1301806092a864886f70d010903310b06
#092a864886f70d010701301c06092a864886f70d010905310f170d303730
#363133313731363434385a302306092a864886f70d01090431160414372cb3
#72dc607990577fd0426104a42ee4158d2b305206092a864886f70d01090f
#31453043300a06082a864886f70d0307300e06082a864886f70d03020202
#0080300d06082a864886f70d0302020140300706052b0e030207300d0608
#2a864886f70d0302020128300d06092a864886f70d010101050004820100
#72db6898742f449b26d3ac18f43a1e7178834fb05ad13951bf042e127eea
#944b72b96f3b8ecf7eb52f3d0e383bf63651750223efe69eae04287c9dae
#b1f31209444108b31d34e46654c6c3cc10b5baba887825c224ec6f376d49
#00ff7ab2d9f88402dab9a2c2ab6aa3ecceeaf5a594bdc7d3a822c55e7daa
#aa0c2b067e06967f22a20e406fe21d9013ecc6bd9cd6d402c2749f8bea61
#9f8f87acfbc9e10d6ce91502e34629adca6ee855419afafe6a8233333e14
#ad4c107901d1f2bca4d7ffaadddbc54192a25da662f8b8509782c76977b8
#94879453fbb00486ccc55f88db50fcc149bae066916b350089cde51a6483
#2ec14019611720fc5bbe2400f24225fc
```

証明書を使用したデバイスの設定

証明書を使用してデバイスを設定するには、次のタスクを実行します。

始める前に

すでに、Cisco IOS 暗号化イメージが用意されている必要があります。用意されていない場合は、証明書を設定できません。

手順の概要

1. **enable**
2. **configure terminal**
3. **crypto pki trustpoint** *name*
4. **enrollment terminal**

5. **exit**
6. **crypto pki authenticate** *name*
7. プロンプトで、ベースが暗号化された CA 証明書を入力します。
8. **scripting tcl secure-mode**
9. **scripting tcl trustpoint name** *name*
10. **scripting tcl trustpoint untrusted** {**execute** | **safe-execute** | **terminate**}
11. **exit**
12. **tclsafe**

手順の詳細

ステップ 1 enable

特権 EXEC モードを有効にします。パスワードを入力します（要求された場合）。

例：

```
Device> enable
```

ステップ 2 configure terminal

グローバル コンフィギュレーション モードを開始します。

例：

```
Device# configure terminal
```

ステップ 3 crypto pki trustpoint *name*

デバイスが認証局（CA）*mytrust* を使用して、CA トラストポイント コンフィギュレーション モードを開始することを宣言します。

例：

```
Device(config)# crypto pki trustpoint mytrust
```

ステップ 4 enrollment terminal

カットアンドペーストによる手動での証明書登録を指定します。このコマンドが有効になると、デバイスはコンソール端末に証明書要求を表示します。これにより、このターミナルに発行済みの証明証が入力できるようになります。

例：

```
Device(ca-trustpoint)# enrollment terminal
```

ステップ 5 exit

CA トラストポイント コンフィギュレーション モードを終了し、グローバル コンフィギュレーション モードに戻ります。

例：

```
Device(ca-trustpoint)# exit
```

ステップ 6 **crypto pki authenticate** *name*

CA 証明書を取得して、認証します。証明書フィンガープリントをチェックするよう求められた場合、証明書フィンガープリントをチェックします。

（注） CA では、証明書が自己署名されるため、このコマンドを実行するときは、CA 管理者に問い合わせて、CA の公開キーを手動で認証する必要があります。

例：

```
Device(config)# crypto pki authenticate mytrust
```

ステップ 7 プロンプトで、ベースが暗号化された CA 証明書を入力します。

例：

```
Enter the base 64 encoded CA certificate.
End with a blank line or the word "quit" on a line by itself
MIIEuDCCA6CgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBnjELMAkGA1UEBhMCVVMx
EzARBgNVBAgTCkNhbGlmb3JuaWExETAPBgNVBAcTCFNhbiBKb3NlMRwwGgYDVQQK
ExNDaXNjbyBTeXN0ZW1zLCBJbmMuMQ4wDAYDVQQLEwVOU1NURzEWMBQGA1UEAxMN
Sm9obiBMYXV0bWFubjEhMB8GCSqGSIb3DQEJARYSamxhdXRtYW5AY2lzY28uY29t
MB4XDTA2MTExNzE3NTgwMVoXDTA5MTExNjE3NTgwMVowgZ4xCzAJBgNVBAYTAlVT
MRMwEQYDVQQIEwpDYWxpZm9ybmlhMREwDwYDVQQHEwhTYW4gSm9zZTEcMBoGA1UE
ChMTQ2lzY28gU3lzdGVtcywgSW5jLjEOMAwGA1UECxMFTlNTVEcxFjAUBgNVBAMT
DUpvaG4gTGF1dG1hbm4xITAfBgkqhkiG9w0BCQEWEmpsYXV0bWFuQGNpc2NvLmNv
bTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALxtqTMCirMb+CdyWLuH
oWAM8CEJDwQggL7MWBhoi3TSMd/ww2XBB9biBtdlH6jHsjCiOwAR5OorakwfPyf7
mvRJ2PqJALs+Vn93VBKIG6rZUl4+wdOx686BVddIZvEJQPbROiYTzfazWV70aLMV
bd7/B7vF1SG1YK9y1tX9p9nZyZ0x47OAXetwOaGinvlG7VNuTXaASBLUjCRZsIlz
SBrXXedBzZ6+BuoWm1FK45EYSlag5Rt9RGXXMBqzx91iyhrJ3zDDmkExa45yKJET
mAgDVMcpeteJtif47UDZJK30g4MbMyx/c8WGhmJ54qRL9BZEPmDxMQkNP1018MAl
Q8sCAwEAAaOB/jCB+zAdBgNVHQ4EFgQU9/ToDvbMR3JfJ4xEa4X47oNFq5kwgcsG
A1UdIwSBwzCBwIAU9/ToDvbMR3JfJ4xEa4X47oNFq5mhgaSkgaEwgZ4xCzAJBgNV
BAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMREwDwYDVQQHEwhTYW4gSm9zZTEc
MBoGA1UEChMTQ2lzY28gU3lzdGVtcywgSW5jLjEOMAwGA1UECxMFTlNTVEcxFjAU
BgNVBAMTDUpvaG4gTGF1dG1hbm4xITAfBgkqhkiG9w0BCQEWEmpsYXV0bWFuQGNp
c2NvLmNvbYIBADAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBBAUAA4IBAQBtEs/4
MQeN9pT+XPCPg2ObQU8y2AadI+I34YK+fDHsFOh68hZhpszTN2VpNEvkFXpADhgr
7DkNGtwTCla481v70iNFViQVL+inNrZwWMxoTnUNCK7Hc5kHkXt6cj0mvsefVUzx
Xl70mauhESRVlmYWrJxSsrEILerZYsuv5HbFdand+/rErmP2HVyfdntLnKdSzmXJ
5lwE/Et2QtYNGor0OBlLesowfslR3LhHi4wn+5is7mALgNw/NuTiUr1zH18OeB4m
wcpBIJsLaJu6ZUJQl7IqdswSa3fHd5qq0/k8P9z0YAYrf3+MFQr4ibvsYvHlO087
o2Js1gW4qz34pqNh
Certificate has the following attributes:
       Fingerprint MD5: 1E327DBB 330936EB 2FB8EACB 4FD1133E
      Fingerprint SHA1: EE7FF9F4 05148842 B9D50FAC D76FDC9C E0703246
% Do you accept this certificate? [yes/no]: yes
Trustpoint CA certificate accepted.
% Certificate successfully imported
```

ステップ 8 **scripting tcl secure-mode**

インタラクティブ Tcl スクリプトのシグニチャ確認を有効にします。

```
Device(config)# scripting tcl secure-mode
```

ステップ 9 **scripting tcl trustpoint name** *name*

設定済みの既存のトラストポイント名と証明書を関連付け、Tcl スクリプトを確認します。

```
Device(config)# scripting tcl trustpoint name mytrust
```

ステップ 10 **scripting tcl trustpoint untrusted** {**execute** | **safe-execute** | **terminate**}

（任意）シグニチャ確認に失敗したか、信頼できないモードであるかにかかわらず、**execute**、**safe-execute** または **terminate** の 3 つのキーワードのいずれかを使用してインタラクティブ Tcl スクリプトを実行できます。

- **execute** ：シグニチャの確認に失敗しても、Tcl スクリプトを実行します。 **execute** キーワードを設定すると、シグニチャの確認は一切実行されません。

（注） シグニチャの確認が実行されないため、通常、このキーワードの使用は推奨されません。

execut キーワードは、内部テスト用に提供されており、これにより柔軟性が向上します。たとえば、証明書の期限が切れていても、他の設定が有効であり、既存の設定で作業したい場合は、execute キーワードを使用して、期限の切れた証明書で対処することができます。

- **safe-execute** ：スクリプトをセーフ モードで実行できます。tclsafe コマンドを使用し、インタラクティブ Tcl シェル セーフ モードを開始すると、使用可能なセーフ モード Tcl コマンドを確認できます。この限定されたセーフ モードで何が使用できるかをより深く理解するには、tclsafe Exec コマンドを使用してオプションを確認します。
- **terminate** ：すべてのスクリプトの実行を停止し、デフォルトの動作に戻します。デフォルト ポリシーは終端します。最後のトラストポイント名が削除されると、信頼できないアクションも削除されます。信用できないアクションは、TCL 用に最低でも 1 つのトラストポイント名が設定されていない場合は開始されません。

次に、シグニチャ確認が失敗した場合に、**safe-execute** キーワードを使用して Tcl スクリプトをセーフモードで実行する例を示します。

```
Device(config)# scripting tcl trustpoint untrusted safe-execute
```

ステップ 11 **exit**

グローバル コンフィギュレーション モードを終了し、特権 EXEC モードに戻ります。

```
Device(config)# exit
```

ステップ 12 **tclsafe**

（任意）インタラクティブ Tcl シェルの信頼できないセーフ モードを有効にします。これにより、Cisco コマンドライン インターフェイスから信頼できないセーフ モードで手動により Tcl コマンドを実行できるようになります。

```
Device# tclsafe
```

例：

トラストポイントの確認

デバイス内に設定されているトラストポイントを表示するには、**show crypto pki trustpoints** コマンドを使用します。

手順の概要

1. **enable**
2. **show crypto pki trustpoints**

手順の詳細

ステップ 1 enable

このコマンドでは、特権 EXEC モードをイネーブルにします。

例：

```
Device> enable
```

ステップ 2 show crypto pki trustpoints

このコマンドは、デバイスに設定されているトラストポイントを表示します。

例：

```
Device# show
 crypto pki trustpoints

Trustpoint mytrust:
    Subject Name:
    ea=janedoe@cisco.com
    cn=Jane
    ou=DEPT_ACCT
    o=Cisco
    l=San Jose
    st=California
    c=US
          Serial Number: 00
    Certificate configured.
```

署名済み Tcl スクリプトの確認

署名済み Tcl スクリプトが正しく実行していることを確認するには、**debug crypto pki transactions** コマンドと **tclsh** コマンドを使用します。

手順の概要

1. **enable**
2. **debug crypto pki transactions**
3. **tclsh** *flash:signed-tcl-file*

手順の詳細

ステップ 1 enable

このコマンドでは、特権 EXEC モードをイネーブルにします。

例：

```
Device> enable
```

ステップ 2 debug crypto pki transactions

このコマンドは、CA とデバイス間のやり取りのトレース（メッセージ タイプ）のデバッグ メッセージを表示します。

例：

```
Device# debug crypto pki transactions

Crypto PKI Trans debugging is on
```

ステップ 3 tclsh *flash:signed-tcl-file*

このコマンドは、Tcl シェルで Tcl スクリプトを実行します。

（注） ファイルは、署名付きの Tcl ファイルである必要があります。

例：

```
Device# tclsh flash:hello.tcl

hello
argc = 0
argv =
argv0 = flash:hello.tcl
tcl_interactive = 0
device#
*Apr 21 04:46:18.563: CRYPTO_PKI: locked trustpoint mytrust, refcount is 1
*Apr 21 04:46:18.563: The PKCS #7 message has 0 verified signers.
*Apr 21 04:46:18.563: CRYPTO_PKI: Success on PKCS7 verify!
*Apr 21 04:46:18.563: CRYPTO_PKI: unlocked trustpoint mytrust, refcount is 0
```

次の作業

- 暗号の概要については、『*Security Configuration Guide*』の「Part 5: Implementing and Managing a PKI」の項を参照してください。

署名済み Tcl スクリプトの設定例

キー ペアの生成の例

次に、キー ペア（秘密キーと公開キー）を生成する方法の例を示します。

秘密キーの生成：例

```
Host% openssl genrsa -out privkey.pem 2048
Generating RSA private key, 2048 bit long modulus
.........+++
.....................................................................................+++
e is 65537 (0x10001)
Host% ls -l
total 8
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
Host%
```

秘密キーからの公開キーの生成

```
Host% openssl rsa -in privkey.pem -pubout -out pubkey.pem
writing RSA key
Host% ls -l
total 16
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

証明書の生成の例

次に、証明書を生成する例を示します。

```
Host% openssl req -new -x509 -key privkey.pem -out cert.pem -days 1095
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value, If you enter '.', the field will be left
 blank.
-----
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:California
Locality Name (eg, city) [Newbury]:San Jose
Organization Name (eg, company) [My Company Ltd]:Cisco Systems, Inc.
Organizational Unit Name (eg, section) []:DEPT_ACCT
Common Name (eg, your name or your server's hostname) []:Jane
```

```
Email Address []:janedoe@company.com
Host% ls -l
total 24
-rw-r--r--   1 janedoe eng12       1659 Jun 12 15:01 cert.pem
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

Tcl スクリプトの署名の例

次に、Tcl スクリプトに署名する例を示します。

```
Host% openssl smime -sign -in hello -out hello.pk7 -signer cert.pem -inkey privkey.pem
-outform DER -binary
Host% ls -l
total 40
-rw-r--r--   1 janedoe eng12       1659 Jun 12 15:01 cert.pem
-rw-r--r--   1 janedoe eng12        115 Jun 13 10:16 hello
-rw-r--r--   1 janedoe eng12       1876 Jun 13 10:16 hello.pk7
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
```

署名の確認の例

次に、署名を確認する例を示します。

```
Host% openssl smime -verify -in hello.pk7 -CAfile cert.pem -inform DER -content hello
puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
puts "tcl_interactive = $tcl_interactive"
Verification successful
```

非バイナリデータを使用した署名の変換の例

次に、TCL シグニチャを非バイナリ データに変換する方法の例を示します。

```
#Cisco Tcl Signature V1.0
Then append the signature file to the end of the file.
Host% xxd -ps hello.pk7 > hello.hex
Host% cat my_append
#!/usr/bin/env expect
set my_first {#Cisco Tcl Signature V1.0}
set newline {}
set my_file [lindex $argv 0]
set my_new_file ${my_file}_sig
set my_new_handle [open $my_new_file w]
set my_handle [open $my_file r]

puts $my_new_handle $newline
puts $my_new_handle $my_first
foreach line [split [read $my_handle] "\n"]  {
   set new_line {#}
   append new_line $line
   puts $my_new_handle $new_line
}
```

```
close $my_new_handle
close $my_handle
Host% my_append hello.hex
Host% ls -l
total 80
-rw-r--r--   1 janedoe eng12       1659 Jun 12 15:01 cert.pem
-rw-r--r--   1 janedoe eng12        115 Jun 13 10:16 hello
-rw-r--r--   1 janedoe eng12       3815 Jun 13 10:20 hello.hex
-rw-r--r--   1 janedoe eng12       3907 Jun 13 10:22 hello.hex_sig
-rw-r--r--   1 janedoe eng12       1876 Jun 13 10:16 hello.pk7
-rwxr--r--   1 janedoe eng12        444 Jun 13 10:22 my_append
-rw-r--r--   1 janedoe eng12       1679 Jun 12 14:55 privkey.pem
-rw-r--r--   1 janedoe eng12        451 Jun 12 14:57 pubkey.pem
Host% cat hello hello.hex_sig > hello.tcl
Host% cat hello.tcl
puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
puts "tcl_interactive = $tcl_interactive"
#Cisco Tcl Signature V1.0
#3082075006092a864886f70d010702a08207413082073d020101310b3009
#06052b0e03021a0500300b06092a864886f70d010701a08204a13082049d
#30820385a00302010202010300d06092a864886f70d0101040500308195
#310b300906035504061302555331133011060355040813a43616c69666f
#726e69613111300f060355040713085361e204a6f7365311c301a060355
#040a1313436973636f2053797374656d732c20496e632e310e300c060355
#040b13054e53535447310d300b060355040313044a6f686e3121301f0609
#2a864886f70d01090116126a6c6175746d616e40636973636f2e636f6d30
#1e170d3037303631323232303134335a170d3130303631313232303134335
#5a308195310b300906035504061302555331133011060355040813a4361
#6c69666f726e69613111300f060355040713085361e204a6f7365311c30
#1a060355040a1313436973636f2053797374656d732c20496e632e310e30
#0c060355040b13054e53535447310d300b060355040313044a6f686e3121
#301f06092a864886f70d01090116126a6c6175746d616e40636973636f2e
#636f6d30820122300d06092a864886f70d0101010500038201f00308201
#0a0282010100a751eb5ec1f3009738c88a55987c07b759c36f3386342283
#67ea20a89d9483ae85e0c63eeded8ab3eb7a08006689f09136f172183665
#c971099ba54e77ab47706069bbefaaab8c50184396350e4cc870c4c3f477
#88c55c52e2cf411f05b59f0eaec0678ff5cc238fdce2263a9fc6b6c244b8
#ffaead865c19c3d3172674a13b24c8f2c01dd8b1bd491c13e84e29171b85
#f28155d81ac8c69bb25ca23c2921d85fbf745c106e7aff93c72316cbc654
#4a34ea88174a8ba7777fa60662974e1fbac85a0f0aeac925dba6e5e850b8
#7caffce2fe8bb04b61b62f532b5893c081522d538005df81670b931b0ad0
#e1e76ae648f598a9442d5d0976e67c8d55889299147d0203010001a381f5
#3081f2301d0603551d0e04160414bc34132be952ff8b9e1af3b93140a255
#e54a667c3081c20603551d230481ba3081b78014bc34132be952ff8b9e1a
#f3b93140a255e54a667ca1819ba48198308195310b300906035504061302
#55533113301106035504081 30a43616c69666f726e69613111300f060355
#0407130853616e204a6f7365311c301a060355040a1313436973636f2053
#797374656d732c20496e632e310e300c060355040b13054e53535447310d
#300b060355040313044a6f686e3121301f06092a864886f70d0109011612
#6a6c6175746d616e40636973636f2e636f6d820100300c0603551d130405
#30030101ff300d06092a864886f70d0101040500038201010000c83c1b074
#6720929c9514af6d5df96f0a95639f047c40a607c83d8362507c58fa7f84
#aa699ec5e5bef61b2308297a0662c653ff446acfbb6f5cb2dd162d939338
#a5e4d78a5c45021e5d4dbabb8784efbf50cab0f5125d164487b31f5cf933
#a9f68f82cd111cbab1739d7f372ec460a7946882874b0a0f22dd53acbd62
#a944a15e52e54a24341b3b8a820f23a5bc7ea7b2278bb56838b8a4051926
#af9c167274ff8449003a4e012bcf4f4b3e280f85209249a390d14df47435
#35efabce720ea3d56803a84a2163db4478ae19d7d987ef6971c8312e280a
#aac0217d4fe620c6582a48faa8ea5e3726a99012e1d55f8d61b066381f77
#4158d144a43fb536c77d6a3182027730820273020101308 19b308195310b
```

```
#300906035504061302555331133011060355040813 0a43616c69666f726e
#69613111300f060355040713 0853616e204a6f7365311c301a060355040a
#1313436973636f2053797374656d732c20496e632e310e300c060355040b
#13054e53535447310d300b0603550403 13044a6f686e3121301f06092a86
#4886f70d01090116126a6c6175746d616e40636973636f2e636f6d020100
#300906052b0e03021a0500a081b1301806092a864886f70d010903310b06
#092a864886f70d010701301c06092a864886f70d010905310f170d303730
#36313331373136343854385a302306092a864886f70d01090431160414372cb3
#72dc607990577fd0426104a42ee4158d2b305206092a864886f70d01090f
#31453043300a06082a864886f70d0307300e06082a864886f70d03020202
#0080300d06082a864886f70d0302020140300706052b0e030207300d0608
#2a864886f70d0302020128300d06092a864886f70d010101050004820100
#72db6898742f449b26d3ac18f43a1e7178834fb05ad13951bf042e127eea
#944b72b96f3b8ecf7eb52f3d0e383bf63651750223efe69eae04287c9dae
#b1f31209444108b31d34e46654c6c3cc10b5baba887825c224ec6f376d49
#00ff7ab2d9f88402dab9a2c2ab6aa3ecceeaf5a594bdc7d3a822c55e7daa
#aa0c2b067e06967f22a20e406fe21d9013ecc6bd9cd6d402c2749f8bea61
#9f8f87acfbc9e10d6ce91502e34629adca6ee855419afafe6a8233333e14
#ad4c107901d1f2bca4d7ffaadddbc54192a25da662f8b8509782c76977b8
#94879453fbb00486ccc55f88db50fcc149bae066916b350089cde51a6483
#2ec14019611720fc5bbe2400f24225fc
```

証明書を使用したデバイスの設定の例

次に、証明書でデバイスを設定する例を示します。

```
crypto pki trustpoint mytrust
 enrollment terminal
!
!
crypto pki authentication mytrust
crypto pki certificate chain mytrust
 certificate ca 00
  308204B8 308203A0 A0030201 02020100 300D0609 2A864886 F70D0101 04050030
  819E310B 30090603 55040613 02555331 13301106 03550408 130A4361 6C69666F
  726E6961 3111300F 06035504 07130853 616E204A 6F736531 1C301A06 0355040A
  13134369 73636F20 53797374 656D732C 20496E63 2E310E30 0C060355 040B1305
  4E535354 47311630 14060355 0403130D 4A6F686E 204C6175 746D616E 6E312130
  1F06092A 864886F7 0D010901 16126A6C 6175746D 616E4063 6973636F 2E636F6D
  301E170D 30363131 31373137 35383031 5A170D30 39313131 36313735 3830315A
  30819E31 0B300906 03550406 13025553 31133011 06035504 08130A43 616C6966
  6F726E69 61311130 0F060355 04071308 53616E20 4A6F7365 311C301A 06035504
  0A131343 6973636F 20537973 74656D73 2C20496E 632E310E 300C0603 55040B13
  054E5353 54473116 30140603 55040313 0D4A6F68 6E204C61 75746D61 6E6E3121
  301F0609 2A864886 F70D0109 0116126A 6C617574 6D616E40 63697363 6F2E636F
  6D308201 22300D06 092A8648 86F70D01 01010500 0382010F 00308201 0A028201
  0100BC6D A933028A B31BF827 7258BB87 A1600CF0 21090F04 2080BECC 5818688B
  74D231DF F0C365C1 07D6E206 D7651FA8 C7B230A2 3B0011E4 EA2B6A4C 1F3F27FB
  9AF449D8 FA8900BB 3E567F77 5412881B AAD9525E 3EC1D3B1 EBCE8155 D74866F1
  0940F6D1 3A2613CD F6B3595E F468B315 6DDEFF07 BBC5D521 B560AF72 D6D5FDA7
  D9D9C99D 31E3B380 5DEB7039 A1A29EF9 46ED536E 4D768048 12D48C24 59B08973
  481AD75D E741CD9E BE06EA16 9B514AE3 91184A56 A0E51B7D 4465D730 1AB3C7DD
  62CA1AC9 DF30C39A 41316B8E 72289113 98080354 C7297AD7 89B627F8 ED40D924
  ADF48383 1B332C7F 73C58686 6279E2A4 4BF41644 3E60F131 090D3F5D 25F0C025
  43CB0203 010001A3 81FE3081 FB301D06 03551D0E 04160414 F7F4E80E F6CC4772
  5F278C44 6B85F8EE 8345AB99 3081CB06 03551D23 0481C330 81C08014 F7F4E80E
  F6CC4772 5F278C44 6B85F8EE 8345AB99 A181A4A4 81A13081 9E310B30 09060355
  04061302 55533113 30110603 55040813 0A43616C 69666F72 6E696131 11300F06
  03550407 13085361 6E204A6F 7365311C 301A0603 55040A13 13436973 636F2053
  79737465 6D732C20 496E632E 310E300C 06035504 0B13054E 53535447 31163014
  06035504 03130D4A 6F686E20 4C617574 6D616E6E 3121301F 06092A86 4886F70D
```

```
  01090116 126A6C61 75746D61 6E406369 73636F2E 636F6D82 0100300C 0603551D
  13040530 030101FF 300D0609 2A864886 F70D0101 04050003 82010100 6D12CFF8
  31078DF6 94FE5CF0 8F83639B 414F32D8 069D23E2 37E182BE 7C31EC14 E87AF216
  61A6CCD3 37656934 4BE4157A 400E182B EC390D1A DC130A56 B8F35BFB D2234556
  24152FE8 A736B670 58CC684E 750D08AE C7739907 917B7A72 3D26BEC7 9F554CF1
  5E5EF499 ABA11124 55966616 AC9C52B2 B1082DEA D962CBAF E476C575 A9DDFBFA
  C4AE63F6 1D5C9F76 7B4B9CA7 52CE65C9 E65C04FC 4B7642D6 0D1A8AF4 38194B7A
  CA307EC9 51DCB847 8B8C27FB 98ACEE60 0B80DC3F 36E4E252 BD731F5F 0E781E26
  C1CA4120 9B0B689B BA654250 97B22A76 CC126B77 C7779AAA D3F93C3F DCF46006
  2B7F7F8C 150AF889 BBEC62F1 E53B4F3B A3626CD6 05B8AB3D F8A6A361
   quit
archive
 log config
scripting tcl trustpoint name mytrust
scripting tcl secure-mode
!
!
end
```

その他の参考資料

ここでは、署名付き TCL スクリプト機能の関連資料について説明します。

関連資料

関連項目	マニュアル タイトル
Cisco PKI の概要：PKI の理解と計画 PKI の実装と管理	*Security Configuration Guide, Release 12.4*
PKI コマンド：コマンド構文、コマンド モード、コマンド履歴、デフォルト、使用に関する注意事項、および例	*Cisco IOS Security Command Reference, Release 12.4*

標準

標準	タイトル
なし	--

MIB

MIB	MIB のリンク
なし	選択したプラットフォーム、Cisco ソフトウェア リリース、およびフィーチャ セットの MIB を検索してダウンロードする場合は、次の URL にある Cisco MIB Locator を使用します。 http://www.cisco.com/go/mibs

RFC

RFC	タイトル
なし	--

シスコのテクニカル サポート

説明	リンク
シスコのサポート Web サイトでは、シスコの製品やテクノロジーに関するトラブルシューティングにお役立ていただけるように、マニュアルやツールをはじめとする豊富なオンライン リソースを提供しています。 お使いの製品のセキュリティ情報や技術情報を入手するために、Cisco Notification Service（Field Notice からアクセス）、Cisco Technical Services Newsletter、Really Simple Syndication（RSS）フィードなどの各種サービスに加入できます。 シスコのサポート Web サイトのツールにアクセスする際は、Cisco.com のユーザ ID およびパスワードが必要です。	http://www.cisco.com/en/US/support/index.html

署名済み Tcl スクリプトの機能情報

次の表に、このモジュールで説明した機能に関するリリース情報を示します。この表は、ソフトウェア リリース トレインで各機能のサポートが導入されたときのソフトウェア リリースだけを示しています。その機能は、特に断りがない限り、それ以降の一連のソフトウェア リリースでもサポートされます。

プラットフォームのサポートおよびシスコ ソフトウェアイメージのサポートに関する情報を検索するには、Cisco Feature Navigator を使用します。Cisco Feature Navigator にアクセスするには、www.cisco.com/go/cfn に移動します。Cisco.com のアカウントは必要ありません。

表 *1 :* 署名済み *Tcl* スクリプトの機能情報

機能名	リリース	機能情報
署名済みTclスクリプト		署名付き TCL スクリプト機能を使用すると、デジタル署名を生成する証明書を作成し、そのデジタル署名を使用してTclスクリプトに署名することが可能になります。 次のコマンドがこの機能で導入されました。**scripting tcl secure-mode**、 **scripting tcl trustpoint name**、 **scripting tcl trustpoint untrusted**、 および **tclsafe.**

用語集

CA：認証局。証明書要求の管理と、関係する IP セキュリティ ネットワーク デバイスへの証明書の発行を担当しているサービス。このサービスは、参加デバイスを一元的に管理します。またこれらのサービスによって受信者は、明示的に信頼してアイデンティティを確認し、デジタル証明書を作成できます。

証明書：ユーザー名またはデバイス名を公開キーにバインドする電子ドキュメント。証明書は、一般的にデジタル署名を確認するために使用されます。

CRL：証明書失効リスト。失効した証明書のリストが含まれる電子ドキュメントです。CRL は、証明書を発行したCAによって作成され、デジタル署名されます。CRLには、証明書の発行日と失効日が含まれています。現行の CRL が失効すると、新しい CRL が発行されます。

IPsec：IP セキュリティ。

ピア証明書：ピアが提示する証明書で、ピアの公開キーが含まれており、トラストポイント CA によって署名されています。

PKI：公開キー インフラストラクチャ。セキュアに設定された通信に使用されているネットワーク コンポーネントの暗号キーと ID 情報を管理するシステムです。

RA：登録局。CA のプロキシとして機能するサーバーで、CA がオフラインのときでも CA の機能を継続できます。RA は CA サーバー上に設定するのが通常ですが、別アプリケーションとして、稼働のための別デバイスを必要とする場合もあります。

RSA キー：公開キー暗号化システムで、Ron Rivest（ロナルド・リベスト）、Adi Shamir（アディ・シャミア）、Leonard Adleman（レオナルド・エーデルマン）の3人によって開発されました。デバイスの証明書を取得するには、RSAキーペア（公開キーと秘密キー）が必要です。

SHA1：Secure Hash Algorithm 1。

SSH：セキュア シェル。

SSL：Secure Socket Layer。

注意事項

本ソフトウェア ライセンスに関連する通知内容を以下に示します。

OpenSSL/Open SSL Project

本製品には、OpenSSL Toolkit（http://www.openssl.org/）で使用するために OpenSSL Project によって開発されたソフトウェアが含まれています。

本製品には、Eric Young 氏（eay@cryptsoft.com）によって作成された暗号化ソフトウェアが含まれています。

本製品には、Tim Hudson 氏（tjh@cryptsoft.com）によって作成されたソフトウェアが含まれています。

ライセンスの問題

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License:

Copyright © 1998-2007 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. 本ソフトウェアの機能または使用に言及するすべての広告資料には、以下の謝辞が表示される必要があります。「本製品には、OpenSSL Toolkit で使用するために OpenSSL Project によって開発されたソフトウェアが含まれています（http://www.openssl.org/）」。
4. The names “OpenSSL Toolkit” and “OpenSSL Project” must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called “OpenSSL” nor may “OpenSSL” appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:

「本製品には、OpenSSL Toolkit（http://www.openssl.org/）で使用するために OpenSSL プロジェクトによって開発されたソフトウェアが含まれています」。

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT “AS IS”' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

本製品には、Eric Young 氏（eay@cryptsoft.com）によって作成された暗号化ソフトウェアが含まれています。本製品には、Tim Hudson 氏（tjh@cryptsoft.com）によって作成されたソフトウェアが含まれています。

Original SSLeay License:

Copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com).

The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young’s, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

“This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)”.

The word ‘cryptographic’ can be left out if the routines from the library being used are not cryptography-related.

1. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: “This product includes software written by Tim Hudson (tjh@cryptsoft.com)”.

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The license and distribution terms for any publicly available version or derivative of this code cannot be changed, i.e. this code cannot simply be copied and put under another distribution license [including the GNU Public License.]

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。