TLS Provider

The TLS provider provides utilities for working with *Transport Layer Security* keys and certificates. It provides resources that allow private keys, certificates and certficate requests to be created as part of a Terraform deployment.

Another name for Transport Layer Security is *Secure Sockets Layer*, or SSL. TLS and SSL are equivalent when considering the resources managed by this provider.

This provider is not particularly useful on its own, but it can be used to create certificates and credentials that can then be used with other providers when creating resources that expose TLS services or that themselves provision TLS certificates.

Use the navigation to the left to read about the available resources.

Example Usage

```
## This example create a self-signed certificate for a development
## environment.
## THIS IS NOT RECOMMENDED FOR PRODUCTION SERVICES.
## See the detailed documentation of each resource for further
## security considerations and other practical tradeoffs.

resource "tls_private_key" "example" {
  algorithm = "ECDSA"
}

resource "tls_self_signed_cert" "example" {
  key_algorithm   = "${tls_private_key.example.algorithm}"
  private_key_pem = "${tls_private_key.example.private_key_pem}"

  # Certificate expires after 12 hours.
  validity_period_hours = 12

  # Generate a new certificate if Terraform is run within three
  # hours of the certificate's expiration time.
  early_renewal_hours = 3

  # Reasonable set of uses for a server SSL certificate.
  allowed_uses = [
      "key_encipherment",
      "digital_signature",
      "server_auth",
  ]

  dns_names = ["example.com", "example.net"]

  subject {
      common_name  = "example.com"
      organization = "ACME Examples, Inc"
  }
}

# For example, this can be used to populate an AWS IAM server certificate.
resource "aws_iam_server_certificate" "example" {
  name             = "example_self_signed_cert"
  certificate_body = "${tls_self_signed_cert.example.cert_pem}"
  private_key      = "${tls_private_key.example.private_key_pem}"
}
```

Data Source: tls_public_key

Use this data source to get the public key from a PEM-encoded private key for use in other resources.

Example Usage

```
data "tls_public_key" "example" {
  private_key_pem = "${file("~/.ssh/id_rsa")}"
}
```

Argument Reference

The following arguments are supported:

- `private_key_pem` - (Required) The private key to use. Currently-supported key types are "RSA" or "ECDSA".

Attributes Reference

The following attributes are exported:

- `private_key_pem` - The private key data in PEM format.
- `public_key_pem` - The public key data in PEM format.
- `public_key_openssh` - The public key data in OpenSSH `authorized_keys` format, if the selected private key format is compatible. All RSA keys are supported, and ECDSA keys with curves "P256", "P384" and "P521" are supported. This attribute is empty if an incompatible ECDSA curve is selected.
- `public_key_fingerprint_md5` - The md5 hash of the public key data in OpenSSH MD5 hash format, e.g. `aa:bb:cc:...`. Only available if the selected private key format is compatible, as per the rules for `public_key_openssh`.

tls_cert_request

Generates a *Certificate Signing Request* (CSR) in PEM format, which is the typical format used to request a certificate from a certificate authority.

This resource is intended to be used in conjunction with a Terraform provider for a particular certificate authority in order to provision a new certificate. This is a *logical resource*, so it contributes only to the current Terraform state and does not create any external managed resources.

Compatibility Note From Terraform 0.7.0 to 0.7.4 this resource was converted to a data source, and the resource form of it was deprecated. This turned out to be a design error since a cert request includes a random number in the form of the signature nonce, and so the data source form of this resource caused non-convergent configuration. The data source form is no longer supported as of Terraform 0.7.5 and any users should return to using the resource form.

Example Usage

```
resource "tls_cert_request" "example" {
  key_algorithm   = "ECDSA"
  private_key_pem = "${file("private_key.pem")}"

  subject {
    common_name  = "example.com"
    organization = "ACME Examples, Inc"
  }
}
```

Argument Reference

The following arguments are supported:

- `key_algorithm` - (Required) The name of the algorithm for the key provided in `private_key_pem`.
- `private_key_pem` - (Required) PEM-encoded private key data. This can be read from a separate file using the `file` interpolation function. Only an irreversable secure hash of the private key will be stored in the Terraform state.
- `subject` - (Required) The subject for which a certificate is being requested. This is a nested configuration block whose structure is described below.
- `dns_names` - (Optional) List of DNS names for which a certificate is being requested.
- `ip_addresses` - (Optional) List of IP addresses for which a certificate is being requested.

The nested `subject` block accepts the following arguments, all optional, with their meaning corresponding to the similarly-named attributes defined in RFC5280 (https://tools.ietf.org/html/rfc5280#section-4.1.2.4):

- `common_name` (string)
- `organization` (string)
- `organizational_unit` (string)

- `street_address` (list of strings)
- `locality` (string)
- `province` (string)
- `country` (string)
- `postal_code` (string)
- `serial_number` (string)

Note: Versions of this provider prior to 1.2.0 may generate a certificate request which cannot be validated by your Certificate Authority if the `*` character is used in any of the `subject` fields, for example as part of the `common_name` when generating a request for a wildcard certificate. Strings containing a `*` and passed to the `dns_names` argument are encoded correctly.

Attributes Reference

The following attributes are exported:

- `cert_request_pem` - The certificate request data in PEM format.

tls_locally_signed_cert

Generates a TLS certificate using a *Certificate Signing Request* (CSR) and signs it with a provided certificate authority (CA) private key.

Locally-signed certificates are generally only trusted by client software when setup to use the provided CA. They are normally used in development environments or when deployed internally to an organization.

Example Usage

```
resource "tls_locally_signed_cert" "example" {
  cert_request_pem   = "${file("cert_request.pem")}"
  ca_key_algorithm   = "ECDSA"
  ca_private_key_pem = "${file("ca_private_key.pem")}"
  ca_cert_pem        = "${file("ca_cert.pem")}"

  validity_period_hours = 12

  allowed_uses = [
    "key_encipherment",
    "digital_signature",
    "server_auth",
  ]
}
```

Argument Reference

The following arguments are supported:

- `cert_request_pem` - (Required) PEM-encoded request certificate data.
- `ca_key_algorithm` - (Required) The name of the algorithm for the key provided in `ca_private_key_pem`.
- `ca_private_key_pem` - (Required) PEM-encoded private key data for the CA. This can be read from a separate file using the `file` interpolation function.
- `ca_cert_pem` - (Required) PEM-encoded certificate data for the CA.
- `validity_period_hours` - (Required) The number of hours after initial issuing that the certificate will become invalid.
- `allowed_uses` - (Required) List of keywords each describing a use that is permitted for the issued certificate. The valid keywords are listed below.
- `early_renewal_hours` - (Optional) If set, the resource will consider the certificate to have expired the given number of hours before its actual expiry time. This can be useful to deploy an updated certificate in advance of the expiration of the current certificate. Note however that the old certificate remains valid until its true expiration time, since this resource does not (and cannot) support certificate revocation. Note also that this advance update can only be performed should the Terraform configuration be applied during the early renewal period.
- `is_ca_certificate` - (Optional) Boolean controlling whether the CA flag will be set in the generated certificate. Defaults to `false`, meaning that the certificate does not represent a certificate authority.

The `allowed_uses` list accepts the following keywords, combining the set of flags defined by both Key Usage (https://tools.ietf.org/html/rfc5280#section-4.2.1.3) and Extended Key Usage (https://tools.ietf.org/html/rfc5280#section-4.2.1.12) in RFC5280 (https://tools.ietf.org/html/rfc5280):

- `digital_signature`
- `content_commitment`
- `key_encipherment`
- `data_encipherment`
- `key_agreement`
- `cert_signing`
- `crl_signing`
- `encipher_only`
- `decipher_only`
- `any_extended`
- `server_auth`
- `client_auth`
- `code_signing`
- `email_protection`
- `ipsec_end_system`
- `ipsec_tunnel`
- `ipsec_user`
- `timestamping`
- `ocsp_signing`
- `microsoft_server_gated_crypto`
- `netscape_server_gated_crypto`

Attributes Reference

The following attributes are exported:

- `cert_pem` - The certificate data in PEM format.
- `validity_start_time` - The time after which the certificate is valid, as an RFC3339 (https://tools.ietf.org/html/rfc3339) timestamp.
- `validity_end_time` - The time until which the certificate is invalid, as an RFC3339 (https://tools.ietf.org/html/rfc3339) timestamp.

Automatic Renewal

This resource considers its instances to have been deleted after either their validity periods ends or the early renewal period is reached. At this time, applying the Terraform configuration will cause a new certificate to be generated for the instance.

Therefore in a development environment with frequent deployments it may be convenient to set a relatively-short expiration time and use early renewal to automatically provision a new certificate when the current one is about to expire.

The creation of a new certificate may of course cause dependent resources to be updated or replaced, depending on the lifecycle rules applying to those resources.

tls_private_key

Generates a secure private key and encodes it as PEM. This resource is primarily intended for easily bootstrapping throwaway development environments.

> **Important Security Notice** The private key generated by this resource will be stored *unencrypted* in your Terraform state file. **Use of this resource for production deployments is *not* recommended.** Instead, generate a private key file outside of Terraform and distribute it securely to the system where Terraform will be run.

This is a *logical resource*, so it contributes only to the current Terraform state and does not create any external managed resources.

Example Usage

```
resource "tls_private_key" "example" {
  algorithm   = "ECDSA"
  ecdsa_curve = "P384"
}
```

Argument Reference

The following arguments are supported:

- `algorithm` - (Required) The name of the algorithm to use for the key. Currently-supported values are "RSA" and "ECDSA".
- `rsa_bits` - (Optional) When `algorithm` is "RSA", the size of the generated RSA key in bits. Defaults to 2048.
- `ecdsa_curve` - (Optional) When `algorithm` is "ECDSA", the name of the elliptic curve to use. May be any one of "P224", "P256", "P384" or "P521", with "P224" as the default.

Attributes Reference

The following attributes are exported:

- `algorithm` - The algorithm that was selected for the key.
- `private_key_pem` - The private key data in PEM format.
- `public_key_pem` - The public key data in PEM format.
- `public_key_openssh` - The public key data in OpenSSH `authorized_keys` format, if the selected private key format is compatible. All RSA keys are supported, and ECDSA keys with curves "P256", "P384" and "P521" are supported. This attribute is empty if an incompatible ECDSA curve is selected.
- `public_key_fingerprint_md5` - The md5 hash of the public key data in OpenSSH MD5 hash format, e.g. `aa:bb:cc:...`. Only available if the selected private key format is compatible, as per the rules for

`public_key_openssh`.

Generating a New Key

Since a private key is a logical resource that lives only in the Terraform state, it will persist until it is explicitly destroyed by the user.

In order to force the generation of a new key within an existing state, the private key instance can be "tainted":

```
terraform taint tls_private_key.example
```

A new key will then be generated on the next `terraform apply`.

tls_self_signed_cert

Generates a *self-signed* TLS certificate in PEM format, which is the typical format used to configure TLS server software.

Self-signed certificates are generally not trusted by client software such as web browsers. Therefore clients are likely to generate trust warnings when connecting to a server that has a self-signed certificate. Self-signed certificates are usually used only in development environments or apps deployed internally to an organization.

This resource is intended to be used in conjunction with a Terraform provider that has a resource that requires a TLS certificate, such as:

- `aws_iam_server_certificate` to register certificates for use with AWS *Elastic Load Balancer*, *Elastic Beanstalk*, *CloudFront* or *OpsWorks*.
- `heroku_cert` to register certificates for applications deployed on Heroku.

Example Usage

```
resource "tls_self_signed_cert" "example" {
  key_algorithm   = "ECDSA"
  private_key_pem = "${file(\"private_key.pem\")}"

  subject {
    common_name  = "example.com"
    organization = "ACME Examples, Inc"
  }

  validity_period_hours = 12

  allowed_uses = [
    "key_encipherment",
    "digital_signature",
    "server_auth",
  ]
}
```

Argument Reference

The following arguments are supported:

- `key_algorithm` - (Required) The name of the algorithm for the key provided in `private_key_pem`.
- `private_key_pem` - (Required) PEM-encoded private key data. This can be read from a separate file using the `file` interpolation function. If the certificate is being generated to be used for a throwaway development environment or other non-critical application, the `tls_private_key` resource can be used to generate a TLS private key from within Terraform. Only an irreversable secure hash of the private key will be stored in the Terraform state.
- `subject` - (Required) The subject for which a certificate is being requested. This is a nested configuration block whose structure matches the corresponding block for `tls_cert_request` (/docs/providers/tls/r/cert_request.html).
- `validity_period_hours` - (Required) The number of hours after initial issuing that the certificate will become invalid.

- `allowed_uses` - (Required) List of keywords each describing a use that is permitted for the issued certificate. The valid keywords are listed below.
- `dns_names` - (Optional) List of DNS names for which a certificate is being requested.
- `ip_addresses` - (Optional) List of IP addresses for which a certificate is being requested.
- `early_renewal_hours` - (Optional) If set, the resource will consider the certificate to have expired the given number of hours before its actual expiry time. This can be useful to deploy an updated certificate in advance of the expiration of the current certificate. Note however that the old certificate remains valid until its true expiration time, since this resource does not (and cannot) support certificate revocation. Note also that this advance update can only be performed should the Terraform configuration be applied during the early renewal period.
- `is_ca_certificate` - (Optional) Boolean controlling whether the CA flag will be set in the generated certificate. Defaults to `false`, meaning that the certificate does not represent a certificate authority.

The `allowed_uses` list accepts the following keywords, combining the set of flags defined by both Key Usage (https://tools.ietf.org/html/rfc5280#section-4.2.1.3) and Extended Key Usage (https://tools.ietf.org/html/rfc5280#section-4.2.1.12) in RFC5280 (https://tools.ietf.org/html/rfc5280):

- `digital_signature`
- `content_commitment`
- `key_encipherment`
- `data_encipherment`
- `key_agreement`
- `cert_signing`
- `crl_signing`
- `encipher_only`
- `decipher_only`
- `any_extended`
- `server_auth`
- `client_auth`
- `code_signing`
- `email_protection`
- `ipsec_end_system`
- `ipsec_tunnel`
- `ipsec_user`
- `timestamping`
- `ocsp_signing`
- `microsoft_server_gated_crypto`

- `netscape_server_gated_crypto`

Attributes Reference

The following attributes are exported:

- `cert_pem` - The certificate data in PEM format.
- `validity_start_time` - The time after which the certificate is valid, as an RFC3339 (https://tools.ietf.org/html/rfc3339) timestamp.
- `validity_end_time` - The time until which the certificate is invalid, as an RFC3339 (https://tools.ietf.org/html/rfc3339) timestamp.

Automatic Renewal

This resource considers its instances to have been deleted after either their validity periods ends or the early renewal period is reached. At this time, applying the Terraform configuration will cause a new certificate to be generated for the instance.

Therefore in a development environment with frequent deployments it may be convenient to set a relatively-short expiration time and use early renewal to automatically provision a new certificate when the current one is about to expire.

The creation of a new certificate may of course cause dependent resources to be updated or replaced, depending on the lifecycle rules applying to those resources.