General PKI Workload Description

We describe the PKI workload for different workflow cases:

1) A website achieves an SSL secure state from non secure state.

2) The process of browser identifying if an apparent SSL secure website is trusted or not.

3) In detail description of the process mentioned in point 2.

1. Scenario where a website achieves an SSL secure state
Actors: Website(site admin), Certificate Authority involved
Objects: SSL Certificate

We would elaborate the whole process to identify the labels involved and the queries made to identify the state of the actors.

a. A website exists with no SSL certificate installed.

b. The website(site admin) submits a certificate signing request to the certificate authority.

c. The certificate authority vets the the certificate signing request.

d. Once verified, the certificate authority generate an a signed X509 certificate for the website and sends it to the website for installation.

e. On receiving the certificate, the website(site admin) installs the certificate on the website.

f. The website achieves an SSL secure state.

2. Scenario where a browser identifies if an apparent SSL secure website is trusted.

Actors: Web browser, User
Objects: SSL Certificate

a. User tries to access a URL from the web browser. Considering only SSL secure URL in our model.

b. The web browser queries an SSL secure website and is presented with the installed SSL certificate chain.

c. The browser checks for every certificate in the certificate chain to be valid and genuine.

d. If every certificate in the certificate in the certificate chain is valid and genuine, the browser identifies the website as genuine and trusts it.

e. If the common name or DN of the certificate matches with the URL queried, it means that the certificate belongs to the URL queried.

f. Browser displays a visual cue(padlock) for secured connection on the website.

Now, lets try to elaborate the above discussed scenarios to identify the underlying steps involved in a path validation process.

The inputs to this process are: The certificate path to be validated, current date/time, Certificate policy OIDs and trust anchor of the certificate path.

The following steps are performed on every certificate in a certificate chain:

a. Check the public key algorithm and public key parameters of the certificate.

b. Check the current date and time to ensure that the certificate is not expired.

c. Check the status of certificate for revocation.

d. Check the name constraints so that the subject name is in the permissible subtrees and not the excluded subtrees(subtrees are as defined in RFC 5280)

e. Check the issuer name such that it should be the same as that of the subject name of the previous certificate in the certificate path.

f. Check the path length so that it is the same as is asserted by the certificate.

g. Key usage extension is checked to ensure signing of certificate.

h. Other checks like policy constraints and Certificate OIDs are checked so that they don’t violate any explicit policy requirements to avoid man in the middle attacks.

Once this procedure validates the above steps for every certificate in the certificate chain, we can assert that the path validation for this particular certificate chain was successful.

PKIX Workload

To define a workload we identified sets and some naming conventions for those sets.

E : Endpoints

We coined the term endpoint for an abstract entity at the other end of the certification process which is supposed to be the one the client(in this case the browser in question) should be actually talking to.

N : Names

$I \subseteq N$:Issuer Names

$S \subseteq N$: Service Names

I denotes issuer name and S denotes service name

B: Browsers

T - Times

$T_{now} \in T$

$T_{issue} \in T$

$T_{exp} \in T$

Times at three events are

T_{now}: Time T at the current moment

T_{issue}: Time at the issuance of the certificate

T_{exp}: Expiration time of the certificate

Certs: $I \times N \times T_{issue} \times T_{exp}$

This statement represents that a cert (certificate) is an entity consisting of some issuer , name and Tissue and Texp.

CA State:

To model the Certificate Authority state we proposed two sets containing certificates in issued state or revoked state.

Issued $\subseteq$ Cert

Revoked $\subseteq$ Cert

Website State:

To model the website state we agreed upon two sets that would let us perform the necessary operations.

Holds ⊆ E x Cert

This state is an abstract binding for an endpoint to a certificate that correctly represents the endpoint. The endpoint is the entity the browser wants to connect to.

Browser State:

Trusted ⊆ B x Cert

The browser in our model holds a set of trusted certificates. In the above representation of the state, it states that it contains entries with some browser with some certificate.

Operations:

```
issue(I, E ,N, Tissue, Texp)
    add <I, N, Tissue, Texp> = C (to issued set)
    add <E, C> (to holds set)
```

This operation issues a certificate. The actions to be taken are adding the certificate to the issued set and creating the holds binding

```
revoke (I, C)
    if C.I = I
        add C to the revoked
```

This operation revoked a certificate. It involves adding C to revoked set.

```
addRootCert(B, C)
    add <B, C> to trusted
```

Adds root certfiicate to browser’s trusted state.

```
removeRootCert(B, C)
    remove <B, C> from trusted
```

Removes root certificate from browsers trusted state.

Queries:

isAuthorized checks for two conditions:

1) Is the name in the certificate same as the service requested by the browser
2) is the certificate chain valid

isValid checks:

1) every certificate in the certificate chain is valid
2) In the certificate chain, the current certificate's Issuer is the same as the current certificate's parent(next as we traverse) in the chain

isRevoked checks if the certificate is not revoked

isTemporalValid checks if the certificate is temporally valid i.e. currentTime is between issued time and expiration time

certValid checks if a certificate is not revoked, is temporally valid and is in Issued set.

```
isAuthorized(B, E, S, $C^+$)
    if $C_0.N = S \wedge \langle E, C_0 \rangle \in$ Holds
        isValid (B, $C^+$)
    else
        false

isValid(B, $C^+$)
    for $C_i \in C_0 \dots C_{n-2}$
        if $C_i.I \neq C_{i+1}.N \vee$ !certValid(B, $C_i$)
            false
    if $\langle B, C_{n+1} \rangle \in$ Trusted $\wedge$ certValid(B , $C_{n-1}$)
        true
    else
        false

isRevoked(C)
    C $\in$ Revoked

isTemporalValid(C)
    $C.T_{issue} \leqq T_{now} \leqq C.T_{exp}$

certValid(C)
    C $\in$ Issued $\wedge$ !isRevoked(C) $\wedge$ isTemporalValid(C)
```

Certificate Pinning

Pinning is the process of associating a host with their *expected* X509. In other words we hard code in the client the certificate which is known to be used by the server.

How does it work?

When we try and reach a website we try and make calls to the server over SSL. If the server we hit is known then, we can stop to rely on the CA to validate the certificate provided by the web server. This can be done by pinning the certificate to a host (web server). Hence, when a host provides the user with a different certificate than what is pinned at the client side, then the user knows to not trust the host.

example: Say we pin a host(URL). The fingerprint of the certificate is stored at the client end. In addition to normal certificate verification when we try and reach the web server, we can use pinning to validate the certificate (basically acts a input to the validation method).

Say cert A has a validity for 1 year. Host = abc.com. When a user tries to access abc.com the normal procedure to determine if a certificate is valid proceeds. In addition to that we use pinning to validate. If a certificate changes before its validity (say after 5 months in this case) the validation fails and a warning could be issued to the user.

Points to consider:

- Certificates can be pinned to a host on the client side while development. For example, google Chrome comes bundled with certificates for www.google.com. These certificates are bundled in the client browser when they are developed.
- Certificates can also be pinned at first time use.
- One entity can have many certificates.
- Every certificate has a validity period. Once a certificate expires the browser has to be updated to load the new certificates. (Frequent issues of new certificates might be an issue).
- When we pin a certificate, we don't pin the CA's certificate; we pin the one at the end of the chain.
-

Adoption:

CA pinning: Mozilla work in progress

Dynamic Public Key Pinning: Google work in progress

References:

1. http://www.ietf.org/proceedings/82/slides/websec-1.pdf
2. http://tack.io/draft.html
3. https://www.imperialviolet.org/2011/05/04/pinning.html
4, http://tools.ietf.org/html/draft-evans-palmer-hsts-pinning-00 (They talk about un-pinning in this doc but wrt public keys instead of certificates)
http://tools.ietf.org/html/draft-ietf-websec-key-pinning-11

Static Public - Key Pinning

Notes:

- SPKI Fingerprint is defined as the output of a known cryptographic hash algorithm whose input is the DER-encoded ASN.
  ```
  SubjectPublicKeyInfo  ::=  SEQUENCE  {
       algorithm            AlgorithmIdentifier,
       subjectPublicKey     BIT STRING  }
  AlgorithmIdentifier  ::=  SEQUENCE  {
       algorithm            OBJECT IDENTIFIER,
       parameters           ANY DEFINED BY algorithm OPTIONAL  }
  ```
- Upon receipt of the Public-Key-Pins response header field, the UA notes the host as a Pinned Host, storing the Pins and their associated directives in non-volatile storage (for example, along with the HSTS metadata).
-
    1. The UA MUST note the Pins if and only if it received the Public- Key-Pins response header field over an error-free TLS connection.
    2. The UA MUST note the Pins if and only if the TLS connection was authenticated with a certificate chain containing at least one of the SPKI structures indicated by at least one of the given SPKI Fingerprints.
    3. The UA MUST note the Pins if and only if the given set of Pins
       contains at least one Pin that does NOT refer to an SPKI in the
       certificate chain.
- Validation:
    - When a UA connects to a Pinned Host, if the TLS connection has errors, the UA MUST terminate the connection without allowing the user to proceed anyway
    - A UA SHOULD perform Pin Validation whenever connecting to a Known Pinned Host, but MAY allow Pin Validation to be disabled for Hosts according to local policy. For example, a UA may disable Pin Validation for Pinned Hosts whose validated certificate chain terminates at a user-defined trust anchor, rather than a trust anchor built-in to the UA.
    - the UA will compute the SPKI Fingerprints for each certificate in the Pinned Host's validated certificate chain, using each supported hash algorithm for each certificate.
    - The UA will then check that the set of these SPKI Fingerprints intersects the set of SPKI Fingerprints in that Pinned Host's Pinning Metadata. If there is set intersection, the UA continues with the connection as normal. Otherwise, the UA MUST treat this Pin Failure as a non-recoverable error.

Terminology:

- Subject Public Key Info: This field is used to carry the public key and identify the algorithm with which the key is used. [1]
- Subject Public Key Info Fingerprint: The fingerprint is the SHA-1 or SHA-256 hash of the DER-encoded ASN.1 representation of the SubjectPublicKeyInfo (SPKI) field of the X.509 certificate. [2]
-

Endpoint State

Key ⊆ E × K x Cert Chain (each tuple accessible only by corresponding E)

- In each endpoint's local state, they have a private key K corresponding to a certificate Cert.

CA State

Issued ⊆ Cert

Browser State

Trusted ⊆ B x Cert (accessible only by B)

PinnedKeys ⊆ B x SubjectPublicKeyInfo fingerprint

//?Check (Pin validation = True)

issue(CA, endpoint, service name, pubkey, times)

> Have CA issue to endpoint a certificate binding service name to pubkey with for given time period. Modifies EndPoint state.

getpubkey(endpoint address)

> Do SSL/TLS handshake, returns pubkey and cert chain. After this call, browser believes it is talking to the owner of the private key corresponding to pubkey.

?chainValid(trusted CAs, chain)

> Do RFC 5280 path validation, return True if successful, False otherwise.

?certRevoked(cert)

> Checks if a certificate is revoked. Return True if revoked, False otherwise.
> We will have different implementations for CRLs and OCSP.
> Usage: After chainValid, we walk the chain and check revocation for each certificate. This could be via CRL OCSP.

// Adds the pin of an endpoint address to list of pinned addresses.

addPin(endpoint address)

Host with max-age directive (expiration date) comes bundled with the browser or compare the domain name with HSTS host domain name. When compared if there is a superdomain match or congruent match then the host must not be added again else add host.

// This function is to determine if the SPKI fingerprint provided by the known endpoint address is valid or not

?pinCheck(chain, metadata)

Compute SPKI fingerprint of every certificate in the certificate chain. Compare them with SPKI metadata. If matched then valid pin else invalid (pinCheckOutput).

// This function is to unpin an endpoint address from the list of known pins.

unPin(SKPI)

Max-age directive = 0 || max-age directive < current date → Un-piin

?isAuthorized(endpoint address, chain, pubkey, pinCheckOutput)

Return True if entity owning pubkey is allowed to represent the given endpoint address.

Usage: This is the top-level "call" that orchestrates the others. Chain is used as evidence to make the decision.

Orchestration (Website is pinned/not pinned):

1. Initial state/maintenance:
    a. Browser is configured with *trusted CAs*
    b. Browser is maintaining a set of CRLs
    c. Browser maintains a set of SKPI fingerprint metadata.
2. addPin(endpoint address) is called to pin a host in the browser.**(This function has to be called even when the website is not pinned. This is because it compares the domain name(endpoint address) in order to determine if the pin has to be added or not)**
3. User requests a secure webpage at *endpoint address*
4. Browser calls getpubkey(*endpoint address*) -> *pubkey*, *cert chain*
5. Browser calls isAuthorized(*endpoint address*, cert chain, pubkey)
    a. isAuthorized calls chainValid(*trusted CAs*, *cert chain*) to do 5280 validation.
    b. pinCheck(SPKI) is called to validate the pins.
    c. isAuthorized calls certRevoked(*cert* for *cert* in *cert chain*) to check for revoked certificates.
    d. isAuthorized calls unPin(*SPKI*) to check for unpinned hosts.
    e. Other schemes may do other stuff here, e.g. check CT logs.

References:

[1] http://www.ietf.org/rfc/rfc3280.txt
[2] http://tools.ietf.org/html/draft-evans-palmer-key-pinning-00
[3] http://tools.ietf.org/html/draft-ietf-websec-key-pinning-11

CRL Scheme

<u>Scheme Description:</u> Revoked certificates are maintained as Certificate revoked lists. These lists are stored in the browser and are broadcasted by the CA to the browsers regularly as well as on revocation of a certificate.

<u>Endpoint State</u>
Key ⊆ E × K x Cert Chain (each tuple accessible only by corresponding E)

- In each endpoint's local state, they have a private key K corresponding to a certificate Cert.

<u>CA State</u>
Issued ⊆ Cert

<u>Browser State</u>
Trusted ⊆ B x Cert (accessible only by B)

Note: We return temporary data, for "permanent" changes, we modify state.

issue(CA, endpoint, service name, pubkey, times)
    Have CA issue to endpoint a certificate binding service name to pubkey with for given time period. Modifies EndPoint state.

getpubkey(endpoint address)
    Do SSL/TLS handshake, returns pubkey and cert chain. After this call, browser believes it is talking to the owner of the private key corresponding to pubkey.

?chainValid(trusted CAs, chain)
    Do RFC 5280 path validation, return True if successful, False otherwise.
?certRevoked(cert)
    Checks if a certificate is revoked. Return True if revoked, False otherwise.
    We will have different implementations for CRLs and OCSP.
    Usage: After chainValid, we walk the chain and check revocation for each certificate. This could be via CRL OCSP.
?isAuthorized(endpoint address, chain, pubkey)
    Return True if entity owning pubkey is allowed to represent the given endpoint address.
    Usage: This is the top-level "call" that orchestrates the others. Chain is used as evidence to make the decision.

Orchestration (CRL version):

1. Initial state/maintenance:
    a. Browser is configured with *trusted CAs*

    b. Browser is maintaining a set of CRLs
2. User requests a secure webpage at *endpoint address*
3. Browser calls getpubkey(*endpoint address*) -> *pubkey*, *cert chain*
4. Browser calls isAuthorized(*endpoint address*, cert chain, pubkey)
    a. isAuthorized calls chainValid(*trusted CAs*, *cert chain*) to do 5280 validation.
    b. isAuthorized calls certRevoked(*cert* for *cert* in *cert chain*) to check for revoked certificates.
    c. Other schemes may do other stuff here, e.g. check CT logs.

OCSP Scheme

Scheme Description: The status of the certificate can be determined by querying the OCSP servers. This indicates if a certificate is revoked or not revoked.

Endpoint State

Key ⊆ E × K x Cert Chain (each tuple accessible only by corresponding E)

- In each endpoint's local state, they have a private key K corresponding to a certificate Cert.

CA State

Issued ⊆ Cert

Browser State

Trusted ⊆ B x Cert (accessible only by B)

OCSP Server State

Cert x Status

Note: We return temporary data, for "permanent" changes, we modify state.

issue(CA, endpoint, service name, pubkey, times)

Have CA issue to endpoint a certificate binding service name to pubkey with for given time period. Modifies EndPoint state.

getpubkey(endpoint address)

Do SSL/TLS handshake, returns pubkey and cert chain. After this call, browser believes it is talking to the owner of the private key corresponding to pubkey.

?chainValid(trusted CAs, chain)

Do RFC 5280 path validation, return True if successful, False otherwise.

?certRevoked(OCSPServerURL,cert)

Checks for the certificate status online with an OCSP server. The server returns the status of the certificate , say True for Revoked else False.

Usage: After chainValid, we walk the chain and check status of each certificate.

?isAuthorized(endpoint address, chain, pubkey)

Return True if entity owning pubkey is allowed to represent the given endpoint address.

Usage: This is the top-level "call" that orchestrates the others. Chain is used as evidence to make the decision.

Orchestration (OCSP version):

1. Initial state/maintenance:
    a. OCSP server has a state that maintains certificate and their status.
2. User requests a secure webpage at *endpoint address*
3. Browser calls getpubkey(*endpoint address*) -> *pubkey*, *cert chain*
4. Browser calls isAuthorized(*endpoint address*, cert chain, pubkey)
    a. isAuthorized calls chainValid(*trusted CAs*, *cert chain*) to do 5280 validation.
    b. isAuthorized calls certRevoked(OCSPServerURL,*cert* for *cert* in *cert chain*) to check for certificate status on the online OCSP server.

CT Scheme

Scheme Description: The authenticity of the certificate is determined by the presence in certificate log services. Certificate log services are network services that maintain the record of every certificate issued. A certificate is said to be conflicting or misbehaved if it is not accompanied by an log entries or if the log entries for the same certificate are conflicting with other log entries. Then, this certificate is marked to be revoked. Certificate Transparency does not handle revocation and hence depends on other scheme to handle revocation for itself.

Endpoint State
Key ⊆ E × K x Cert Chain (each tuple accessible only by corresponding E)

- In each endpoint's local state, they have a private key K corresponding to a certificate Cert.

CA State
Issued ⊆ Cert

Browser State
Trusted ⊆ B x Cert (accessible only by B)

Log State
Cert x log

Note: We return temporary data, for "permanent" changes, we modify state.

issue(CA, endpoint, service name, pubkey, times)
    Have CA issue to endpoint a certificate binding service name to pubkey with for given time period. Modifies EndPoint state.

getpubkey(endpoint address)
    Do SSL/TLS handshake, returns pubkey and cert chain. After this call, browser believes it is talking to the owner of the private key corresponding to pubkey.

?chainValid(trusted CAs, chain)
    Do RFC 5280 path validation, return True if successful, False otherwise.
?certRevoked(cert)
    Usage: After chainValid, we walk the chain and check validity of each certificate.
    We call isCertToBeRevoked(cert) to mark all the certificate that should be revoked.
    If ever isCertToBeRevoked(cert) return True, the function return False and reports the certificate for revocation.
?isCertToBeRevoked(cert)
    Usage:Checks for the audit proof entry for a certificate on online certificate logs .
    If the certificate is accompanied by audit proof entry in the certificate logs, then the

certificate is valid.
It also checks to see if a log misbehaves for a particular certificate where the audit proofs are not consistent with each other.
Return: If certificate audit proof is found return False else return True.

?isAuthorized(endpoint address, chain, pubkey)

Return True if entity owning pubkey is allowed to represent the given endpoint address.
Usage: This is the top-level “call” that orchestrates the others. Chain is used as evidence to make the decision.

Orchestration (CT version):

1. Initial state/maintenance:
    a. Certificate log services maintain certificate audit proofs for certificates issued and audited.
2. User requests a secure webpage at *endpoint address*
3. Browser calls getpubkey(*endpoint address*) -> *pubkey*, *cert chain*
4. Browser calls isAuthorized(*endpoint address*, cert chain, pubkey)
    a. isAuthorized calls chainValid(*trusted CAs*, *cert chain*) to do 5280 validation.
    b. isAuthorized calls certRevoked(*cert* for *cert* in *cert chain*) to check for certificate status on the certificate log service and revocation status by the CA.

Acknowledgements

This work was supported in part by the National Science Foundation through the awards 1228668 and 1228697. We thank and acknowledge contributions from students at Indiana University: Rohan Mathure, Veer Singh, and Rahul Sinha.