Attestation Protocols

Ken Goldman
kgoldman@us.ibm.com
December 7, 2017

1. Introduction 5
2. Provisioning Goals and Attack Model 6
3. Provisioning Process 7
3.1. Client Request 8
3.2. Server Challenge 9
3.3. Client Response 11
3.4. Server Acknowledge 12
3.5. Epilogue 13
4. Quote Goals and Attack Model 15
5. Quote Process 16
5.1. Client requests a nonce 17
5.2. Server supplies nonce and PCR selection 17
5.3. Client returns the quote data 17
5.4. Server requests an event log 18
5.5. Client returns the event log 18
5.6. Server acknowledge 19

1. Introduction

This paper describes the provisioning process for an attestation signing key, between the machine performing the attestation (the client) and the attestation verifier (the server).

It goes on to describe the attestation quote protocol.

2. Provisioning Goals and Attack Model

The server wants to install a client quote signing public key that it trusts to create valid attestation quote signatures.

The client, as an attacker, wants to convince the server to install an attestation key whose private part is under complete control of the client, so that the client can forge quote signatures.

The server trusts itself but does not trust the client software. The server contains TPM vendor root certificates that it can use to validate vendor provisioned TPM endorsement key (EK) certificates. These vendor root certificates are the root of trust for the provisioning process.

The process protects against

- client attacks on the protocol

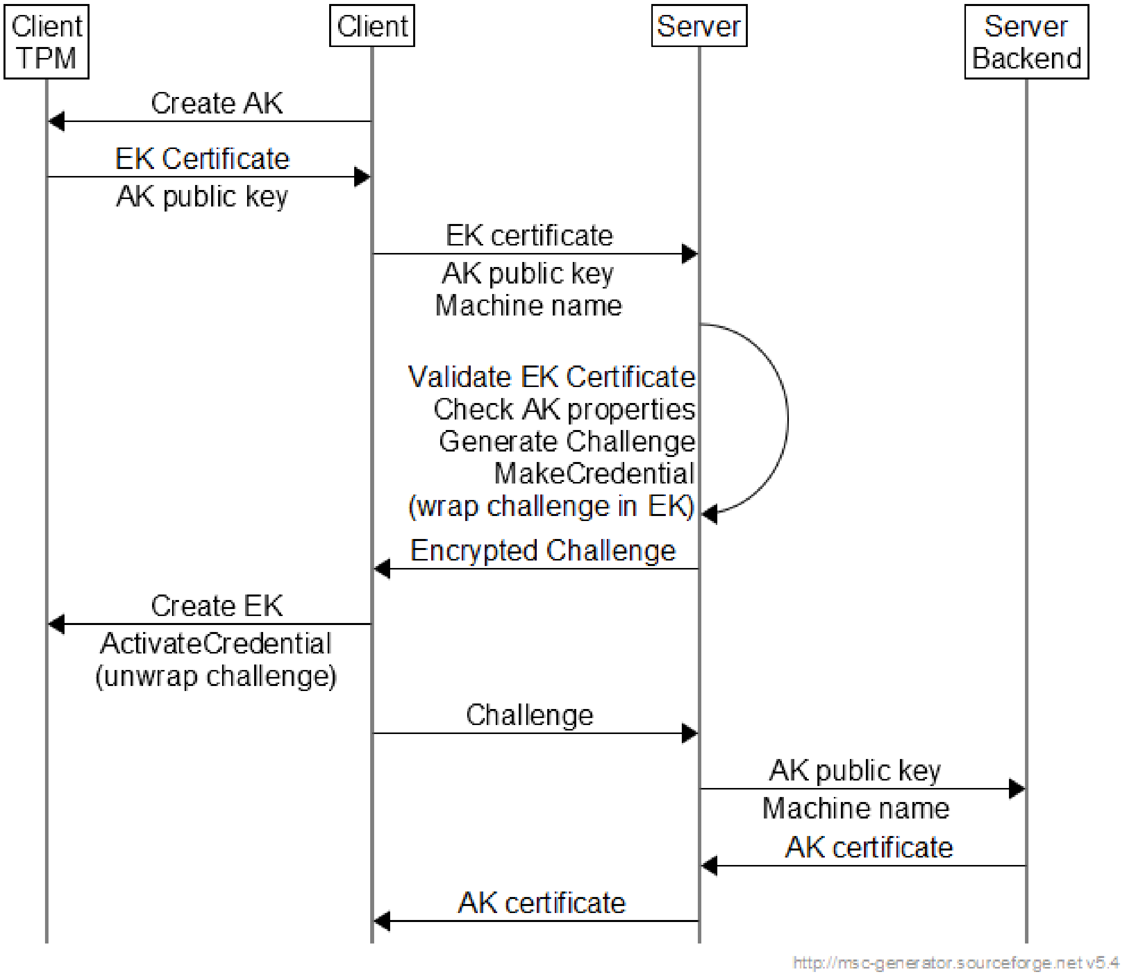
The process does not protect against

- the client providing an inaccurate host name
- a client that is not is authorized to be provisioned
- a client connecting to an incorrect server

3. Provisioning Process

The process consists of four steps:

- client request
- server challenge
- client response
- server certificate


Client TPM
Client
Server
Server Backend
Create AK
EK Certificate
AK public key
EK certificate
AK public key
Machine name
Validate EK Certificate
Check AK properties
Generate Challenge
MakeCredential
(wrap challenge in EK)
Encrypted Challenge
Create EK
ActivateCredential
(unwrap challenge)
Challenge
AK public key
Machine name
AK certificate
AK certificate
http://msc-generator.sourceforge.net v5.4

3.1. Client Request

1. The client creates its SRK primary storage key if it does not already exist.

   The TPM2_CreatePrimary() command generates a repeatable key pair when the identical template is used.

2. The client creates an attestation signing key under (encrypted with, wrapped by) the SRK.

3. The client reads the TPM vendor EK (endorsement key) certificate from TPM NV space.

4. The client sends the enrollment request to the server. The request consists of:

   - command - enrollrequest
   - hostname - client hostname
   - ekcert - EK certificate (X.509 DER format)
   - public - attestation key public part (TPMT_PUBLIC structure)

3.2. Server Challenge

The server receives the enrollment request, with a host name, EK certificate, and attestation public key. The client claims that the certificate and key come from an authentic TPM. The server trusts neither claim.

1. The server verifies that the host name has not already been successfully enrolled.

2. The server validates the EK certificate against its list of TPM vendor root certificates.

   If the certificate is valid, the server trusts that the certificate came from **an** authentic TPM, but not that it came from the **client's** TPM.

3. The server extracts the EK public key from the EK certificate.

4. The server validates the attestation public key properties: fixedTPM, fixedParent, sensitiveDataOrigin, sign, restricted, not decrypt, RSASSA algorithm, SHA-256, and RSA 2048-bit with the default exponent.

   These are the properties required of an attestation key: it was generated on a TPM, cannot be duplicated off the TPM, and it is restricted to sign only TPM generated data such as a quote.

   The server does not trust that this key came from the client's TPM, as it has received only a public part.

5. The server generates a random challenge.

6. The server loads (TPM2_LoadExternal) the public attestation key, using its TPM to calculate the Name. The Name is a hash of the public area.

7. The server loads the client EK public key, extracted from the EK certificate.

8. The server runs TPM2_MakeCredential(), supplying the EK handle, the challenge, and the attestation key Name.

   TPM2_Makecredential() links together the challenge and the attestation key Name, then encrypts the result with the EK public key. This becomes the server challenge to the client.

   Steps 6, 7 and 8 use no TPM secrets. The calculations could be performed completely in software. However, they are complex. It is easier for the implementation to use a TPM (perhaps a software TPM) than to rewrite and maintain another version.

9. The server stores the hostname and certificate in its "machines" database table.

   However, it marks the row invalid, since the server still does not know whether the EK certificate or the attestation key came from the client's TPM.

10. The server sends the response to the enrollment request:

    - response - enrollrequest
    - credentialblob - the make credential output
    - secret - challenge encrypted with the client EK public key

3.3. Client Response

The client receives the challenge, the server response to the enrollment request.

1. The client (re)creates the EK using either the default EK template or the EK template and nonce from the TPM NV.

2. The client loads its previously saved attestation key.

3. The client runs the TPM2_ActivateCredential() command, specifying the credentialBlob, the encrypted secret, the EK handle, and the attestation key handle.

   Use of the EK requires a policy session with a policy secret against the endorsement authorization.

4. The client TPM validates the authorization: the EK policy for the EK and an empty password for the attestation key.

5. The client TPM (simplified) validates the integrity of the credentialBlob against the EK.

6. The client TPM validates that the Name of the loaded attestation key matches that in the credentialBlob.

   This check prevents the client from sending an attestation key to the server different from the one generated by the TPM.

7. The client TPM then decrypts secret using the EK private key to recover the challenge.

   This step proves that the client was using an authentic TPM to generate the attestation key.

8. The client sends a command to the server, requesting enrollment of the certificate.

   - command - enrollcert
   - hostname - the client host name
   - challenge - the decrypted challenge

3.4. Server Acknowledge

1. The server matches the challenge certificate to the challenge that the server generated.

   This proves that the client could decrypt the challenge. The client could only do that if it had the EK private key (known to be from an authentic TPM) and an attestation key with the server-validated properties (because the client TPM matches the Name).

   The match is important. It is not enough to detect a valid attestation public key, since the client could try to install a counterfeit.

2. The server constructs an X.509 certificate for the attestation public key, and signs it with its privacy CA key. It uses the client hostname as the subject CN - common name.

3. The server stores the certificate in the database.

4. The server sends the certificate to the client.

   - response - enrollcert
   - akcert - certificate in PEM format

After the server response, the client saves the attestation key public and private part in the filesystem for later use when signing quotes.

It optionally stores the attestation key certificate.

3.5. Epilogue

A careful reader may have observed that the server could have stored a raw attestation public key. The server never walks the certificate chain back to its privacy CA root.

However, there are some advantages to this design.

First, an X.509 certificate is a convenient way for the server to store a public key. It permits standard signature verification, while a proprietary public key format would require extra code.

More interesting, the client and server now have an X.509 certificate for a TPM signing key. This opens up other use cases, using the server privacy CA root certificate as a root of trust.

- The enrollment server and the attestation server can be separate. The enrollment sever sends the certificate to the client, and the client sends it to the attestation server.
- The client or server can send the certificate to another attestation server, avoiding the need to run this protocol more than once.
- The client can send the certificate to a recipient, and then use the signing key for applications other than attestation.
- The client can locally use the key to certify other TPM keys, with a certificate chain back to the server privacy CA root.

4. Quote Goals and Attack Model

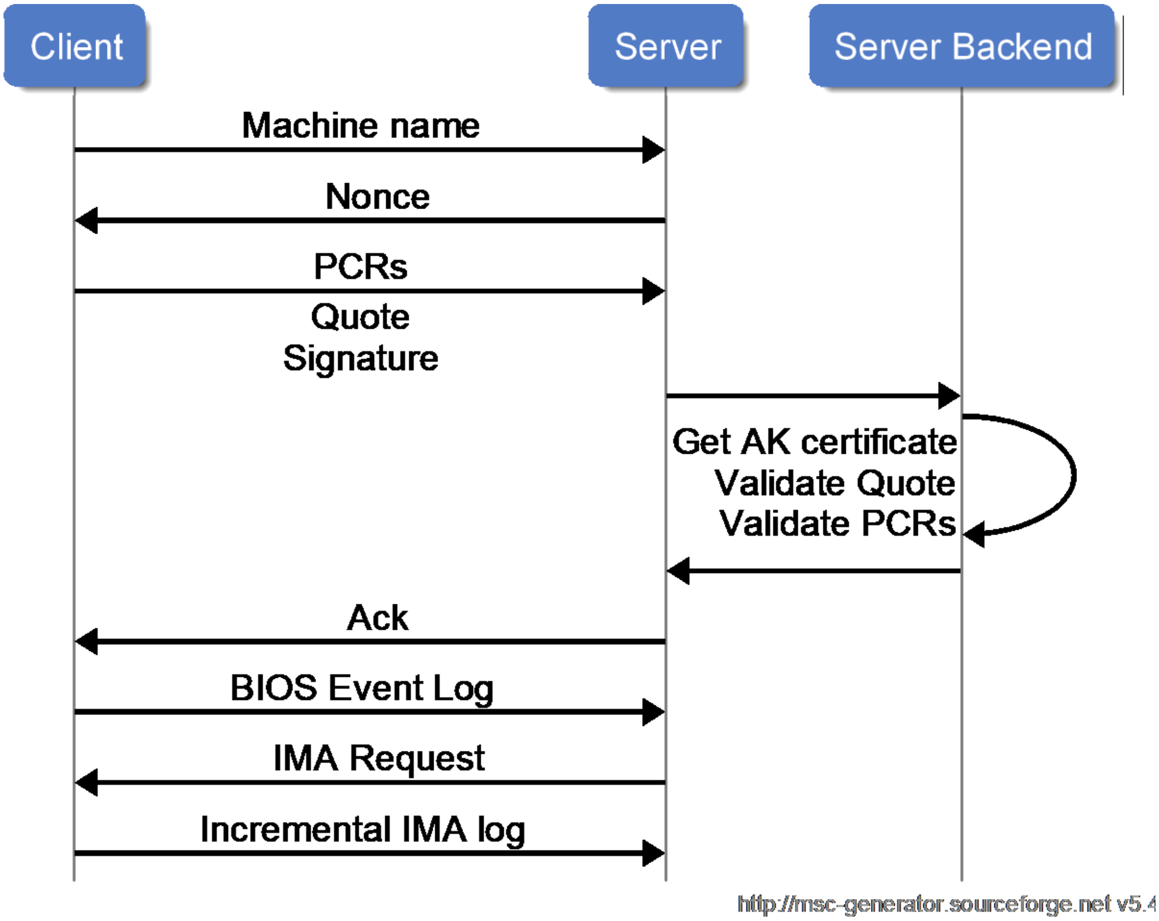
A quote is essentially a signature over a client event log. There are two levels of indirection:

1. Event log entries are hashed into the client PCRs.
2. PCRs are hashed into the data that is signed by the attestation key.

The server uses the quote signature to validate that the client event log is authentic and current. It can then use the event log entries to establish trust in the client.

The client, as an attacker, wants to convince the server that a tampered event log is authentic, or that a replay of a previous quote is current.

5. Quote Process


Client
Server
Server Backend
Machine name
Nonce
PCRs
Quote
Signature
Get AK certificate
Validate Quote
Validate PCRs
Ack
BIOS Event Log
IMA Request
Incremental IMA log
http://msc-generator.sourceforge.net v5.4

The process consists of six steps:

- Client requests a nonce
- Server supplies a nonce and PCR selection
- Client returns the quote data
- Server requests an event log
- Client returns the event log
- Server acknowledge

5.1. Client requests a nonce

1. The client sends the nonce request to the server. The request consists of:

    - command - nonce
    - hostname - client hostname
    - user ID - the client account that generated the request (untrusted, for reference)

5.2. Server supplies nonce and PCR selection

1. The server responds with a nonce and a bitmap of PCRs that the client should quote.

    The PCR selection is currently hard coded to "all PCRs". There is little performance benefit to quoting fewer PCRs. The server can ignore those not of interest.

    - response - nonce
    - nonce - a 32 byte nonce
    - pcrselect - all PCRs

5.3. Client returns the quote data

1. The client runs the TPM2_Load() command to load its attestation key

2. The client runs the TPM2_Quote() command, supplying the nonce and PCR selection, and signing with the loaded attestation key.

3. The client runs the TPM2_PCR_Read() command several times to read the selected PCRs.

4. The client sends the quote to the server.

    - command - quote
    - hostname - the client host name
    - pcr0 - pcr23
    - quote data
    - quote signature
    - client boot time

The boot time permits a future optimization, where the server may not need to request the entire event log if the client has not rebooted. This is more useful for IMA event logs, because:

- The IMA log is far larger than the firmware event log.
- The IMA log is likely to change on every boot, because the event order changes.

5.4. Server requests an event log

The server validates the quote, and then requests the event log.

1. The server retrieves the provisioned client attestation key X.509 certificate.

2. This certificate is used to verify the signature on the quote data.

3. The server reconstructs the quote data PCR digest from the PCRs. It matches the result to that received from the client.

   The server now trusts that the PCR values sent by the client are authentic.

4. The server matches its copy of the nonce to that in the quote data.

   The server now trusts that the quote is fresh, not a replay of a previous quote.

5. The server sends a response to the client:

   - response - quote

A future optimization can indicate whether the client should send an event log. If the PCRs have not changed, the server does not need the current event log. This is the typical case for firmware event logs, but will be important for IMA logs. The IMA logs are also larger, making the complexity of this optimization worthwhile.

5.5. Client returns the event log

The client receives the quote response, indicating that the quote was valid. The client next sends the event log.

As a future optimization, if the client did not reboot, the server can request an incremental event log. This is not likely to be worthwhile for firmware logs, but can be useful for the larger IMA logs.

1. The client sends a command to the server, requesting processing of the event log.

- command - biosentry
- hostname - the client host name
- nonce - the client nonce
- eventn - the event log entries

5.6. Server acknowledge

The server processes the event log.

1. The server matches the nonce against the nonce the client used for the quote.

   The server uses the nonce as a sort of one time password. The client echoes the quote nonce with the event log and the server checks for a match. This prevents a rogue client from masquerading as a client and causing mischief by sending an incorrect event log. It assumes that the nonce is a random value that cannot be guessed by the rogue.

   This becomes redundant if the client maintains a stateful connection to the server through the process, or if the client uses an authenticated connection. The current design permits an untrusted, stateless connection.

2. The server walks the event log, reconstructing PCR values. At each step, it checks for a PCR match. When all PCRs match, the server is done processing.

   There may be more entries in the event log than were used for the quote. The server ignores entries after a match.

   If a PCR first matches, but further event log entries cause a PCR to no longer match, the server notes an invalid log. There may be more entries at the end of a log, after all PCRs match, but not before then.

3. The server sends a final acknowledgement to the client.

   - response - biosentry