Secure Messages

DESY

Owen Synge
Secure messages
Hepix Spring 2011

HELMHOLTZ ASSOCIATION

DESY

Whats In common ?

> Whats in common

- Dcache
- Postfix
- Yum
- Open stack
- Apel accounting
- BDII
- Castor
- SGE
- Apt
    - Many other applications too.

> Asynchronous communication

- Messages send from one instance/node to another.

Why talk about Signing?

- Not used much in HEP.
  - Is going to be used more now.
    - EMI message group have projects coming.
- Basic technique for security.
- Has unique benefits and unique costs.
- Fits well with Message Queues.
  - Open source solutions now maturing.
  - Allows simple scripts to scale easily.
    - Admins can/do write scripts.
- We all should know about this technology.

Signing and Encryption in HEP

- Old technology.
  - Mostly know its use with email.
  - Is used in CA messages (eg CRL), rpm and deb packages.
- Relatively new in HEP.
  - Eg SynCat, SSM are only examples I know.
- X509 use is less common outside HEP.
  - Most products use PGP (eg rpm deb's etc).
    - I looked into X509 so talk will only cover X509.
- Supported in standard SSL.
  - You need more to support chains of trust etc.

Benefits of Signing and Encryption

- SSL Connections not required for data integrity.
  - Content is secured with a hash that is encrypted with private key.
- Allows message processing asynchronously.
  - Allowing use of message queues.
    - Allows admins to use scripts to scale services.
- Allows messages to be verified later.
  - So data bases can contain nothing but verified messages.
    - Removing fear of corrupted/cracked DB's.
- No delegation needed.

Disadvantages of Signing and Encryption compared with SSL.

- Messages are not interactive.
  - Cannot start bidirectional communication with a message only a service.
- Messages may need to include an reference number.
  - So you know that no intermediate message has been lost.
- Transport may need to be secure.
  - Preventing Man in the middle blocking new messages.
  - So whats the point? (we already said this)
- No common libraries for use in HEP.
  - Syncat and HVWG have signing libraries.
    - Recommend you contact authors and EMI message group if intending to use this technology.

Disadvantages of Only Signing messages

- Unencrypted Messages are readable by all.
  - Unless you use a secure transfer like SSL.
- Need to verify senders certificate.
  - If not known in advance.
    - Makes things simpler if one user.
  - For admins within a site this is enough.
- Need to manage ACL's
  - If more than one Cert accepted.

Disadvantages of Encryption and Signing messages.

- Only can be decrypted by one cert.
  - Do we distribute “service certs”?
    - Used in Glite accounting.
  - Limits use as point to cert communication
    - Much like SSL based communication.
  - Example.
    - Glite accountancy uses Secure STOMP to encrypt messages from sites to destination.

So how do I use all this info?

- Recommend using a AMQP message queue.
  - EMI messaging group recommendations.
    - JSON, SMIME.
    - Other technology is similar (XML signatures etc), but would have to learn more tools to demo this to you all.
- Recommend signing messages.
  - Most admin scripts need security not privacy.
  - Deployment of encryption requires key deployment.
- EPEL for SL6 (and SL5) contains AMQP servers.
  - Also includes AMQP libraries.
  - RabbitMQ Promises to be a scalable message queue.
    - We will cover this later.

How do I sign with SMIME?

- Supported in OpenSSL
  - on command line

```
echo fred > message
openssl smime -sign -in message \
 -out signed-message \
 -signer ~/.globus/usercert.pem  \
 -inkey  ~/.globus/userkey.pem -text
```

```
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha1"; boundary="----
0F358CF91058578108BCB73677C6F049"

This is an S/MIME signed message

------0F358CF91058578108BCB73677C6F049
Content-Type: text/plain

fred

------0F358CF91058578108BCB73677C6F049
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIHdAYJKoZIhvcNAQcCoIIHZTCCB2ECAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb3
DQEHAaCCBSUwggUhMIIECaADAgECAgIz7DANBgkqhkiG9w0BAQUFADA2MQswCQYD
VQQGEwJERTETMBEGA1UEChMKR2VybWFuR3JpZDESMBAGA1UEAxMJR3JpZEthLUNB
MB4XDTExMDExMDE1MDMxN1oXDTEyMDIwOTE1MDMxN1owRjELMAkGA1UEBhMCREUx
EzARBgNVBAoTCkdlcmlhbkdyaWQxDTALBgNVBAsTBERFUlkxEzARBgNVBAMTCk93
ZW4gU3luZ2UwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCkgbPFZrVL
pmwf7GKBBFkwTK5V7RmlupsU3Z3FqdfMnJGn2NrrnHIhthUTCTq4WbLIZTbOEh0n
JqZgZBvYcwJV4V9pais4YlsEug+JLMbB9hZ6e2XgdjXWgLqz6vBSIf6KXi4KhCxe
a4FylvIk7OtY+bg0mg5IFHib6uP7fXhFKdBEapoi+B05wpluBMA+2DBdSt+rjzA8
SwiHUuan60VIyJAxammyOe3IKSpwyBXkQ10XjIhIpoSavqYXJboFOVzUcqxawdbX
Con2W8QfiwFKYupohG/VTusDXFT2MP4k+KxG3/rTTPWUDJme7VUPv3+CTcEO+z4v
X8/XhI44oAXlAgMBAAGjggInMIICIzAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB/wQE
AwIE8DAdBgNVHQ4EFgQUGakUy66kgvulNBIf18WBXjGolqYwXgYDVR0jBFcwVYAU
xnXJKKzRC/w8/7mlHtNfO4BiEjShOqQ4MDYxCzAJBgNVBAYTAkFRMRMwEQYDVQQK
EwpHZXJtYW5HcmlkMRIwEAYDVQQDEwlHcmlkS2EtQ0GCAQAwHQYDVR0RBBYwFIES
b3dlbi5TeW5nZUBkZXN5LmRlMB8GA1UdEgQYMBaBFGdyaWRrYS1jYUBpd3IuZnpr
LmRlMDUGA1UdHwQuMCwwKqAooCaGJGh0dHA6Ly9ncmlkLmZ6ay5kZS9jYS9ncmlk
a2EtY3JsLmRlcjAaBgNVHSAEEzARMA8GDSsGAQQBlDarLAEBAQUwEQYJYIZIAYb4
QgEBBAQDAgWgME4GCWCGSAGG+EIBDQRBFj9DZXJ0aWZpY2F0ZSBpc3N1ZWQgdW5k
ZXIgQ1AvQ1BTIHYuIDEuNSBhdCBodHRwOi8vZ3JpZC5memsuZGUvY2EwJAYJYIZI
AYb4QgECBBcWFWh0dHA6Ly9ncmlkLmZ6ay5kZS9jYTAzBglghkgBhvhCAQgEJhYk
aHR0cDovL2dyaWQuZnprLmRlL2NhL2dyaWRrYS1jcHMucGRmMDMGCWCGSAGG+EIB
AwQmFiRodHRwOi8vZ3JpZC5memsuZGUvY2EvZ3JpZGthLWNybC5kZXIwDQYJKoZI
hvcNAQEFBQADggEBAMbn91TOQ6r4D/aKwgIFXiXe40B7iccz/P5pCFSi1R6IC3KH
Ui4s/f9iAGl9rA21h8QAaRaJ/h1OQNlgMZbc9jDCWcqxr8wQTYAQDiBkspLT68ZO
5xVFRiq3HjkkhwnFfFzsNSiLFYZTRjChPluclYG3TEvSg8dz9Lv/IEJxE5C5lZ2d
e3CSu0vcD0DESiu/sVqPOOHi8NL/59U2ine3z23Y+piCabQCxjT0inT2MmR8UNDF
ij2JJYxlt56U/SQCEe0304w3x1jIg8vcpm4dfh+L2IjJ9hVfEeLaCyhv9Wjbmu5O
vk0yLjcEZ7b4RKeo7djVYh+5kCWJYCr/W6uGW44xggIXMIICEwIBATA8MDYxCzAJ
BgNVBAYTAkRFMRMwEQYDVQQKEwpHZXJtYW5HcmlkMRIwEAYDVQQDEwlHcmlkS2Et
Q0ECAjPsMAkGBSsOAwIaBQCggbEwGAYJKoZIhvcNAQkDMQsGCSqGSIb3DQEHATAc
BgkqhkiG9w0BCQUxDxcNMTEwNDI3MTAxNjM3WjAjBgkqhkiG9w0BCQQxFgQUsIAV
z7GO6RuaGTJx7/8IXrALf+AwUgYJKoZIhvcNAQkPMUUwQzAKBggqhkiG9w0DBzAO
BggqhkiG9w0DAgICAIAwDQYIKoZIhvcNAwICAUAwBwYFKw4DAgcwDQYIKoZIhvcN
AwICASgwDQYJKoZIhvcNAQEBBQAEggEAPCyl+wzV8upYDNStxWlflAzH92EiavCr
qJmy93xWr0Y4tL3uR8OnD24PIiAoo0Mx7Gu+wJHixZTX+CB0XdAjAFXvlKA9UTJX
dl5NPoKvifqMepG3UUe4FMcM/+lFcN/aTtUYQnh5GaFkXyKW5mZBuNyzYHJ4qvP/
gT21Wi8V/kVwejluDSuUhGCraA2lRtaNUM/QT1w3fJfMbOVyWh5Yd5nQl+0LysSq
7P390yVNGAvXhIlzMgB14IFO9gXdsGyVloxz2SfftV0ICx8werbvEYiqJECT28Jf
UpxL6gOZS4MCAPadhiSLMwwcRV1zXDzroC67S3hmRj66dbp9vJ4APQ==

------0F358CF91058578108BCB73677C6F049--
```


DESY

How do I verify a SMIME signature?

> To get the DN and CA of your signature.

```
openssl smime -in your_signed.msg \
 -pk7out | openssl pkcs7 -print_certs
```

> To Verify the message against the CA cert.

```
openssl smime -in your_signed.msg \
 -CAfile /etc/grid-security/certificates/dd4b34ea.0 \
 -verify 1> /dev/null
```

> Also need to consider

- CA name spaces
- Certificate Revocation Lists
- Access control lists.

I made a Library to Simplify processing For HEPIX VWG.

Message Queues for ease of use.

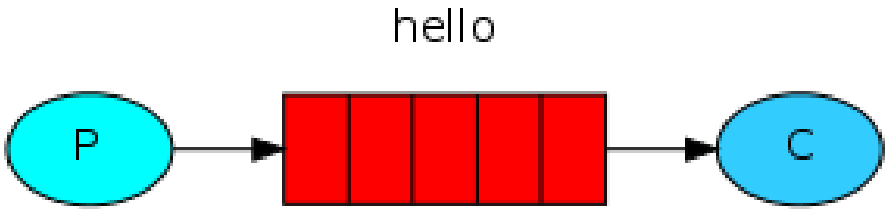
- Messages queues can provide.
  - Scalability and Clustering.
  - Synchronization.
  - Error Handling.
  - Persistence.
  - Standardized communication platform.
- What does the message queue application creator provide.
  - Only write your business logic.
  - AMQP client code doing what I want.
  - No need to write high performance code.

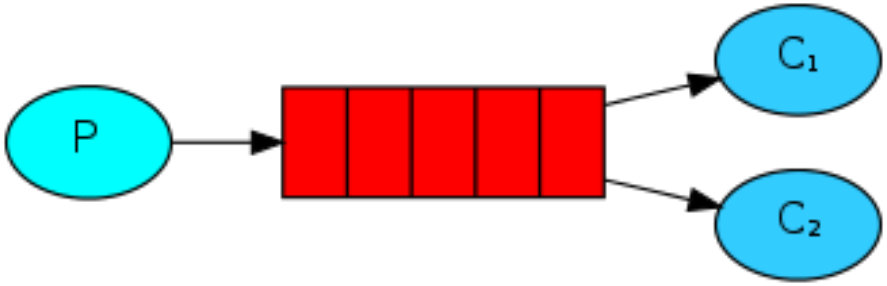
You think admins will write services?

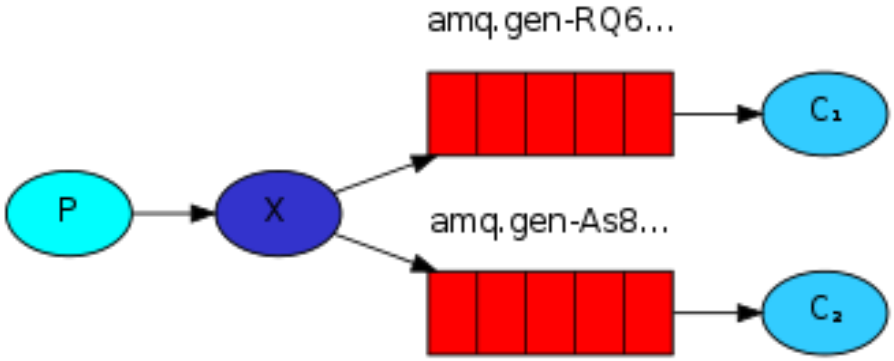
- Admins do this all the time.
  - Examples.
    - Running applications on selected nodes.
    - Draining worker node jobs.
    - Services to see out of date packages.
    - Shutting down 20% of my cluster due to cooling issues.
- Currently use ssh for security.
  - How about error recovery?
  - How about synchronization?
- What have I done with this?
  - Extended Hudson to use re use clean VM's.
    - zero issues with signing and message queues.
  - Plan to make further use of this for DESY virtualisation group.

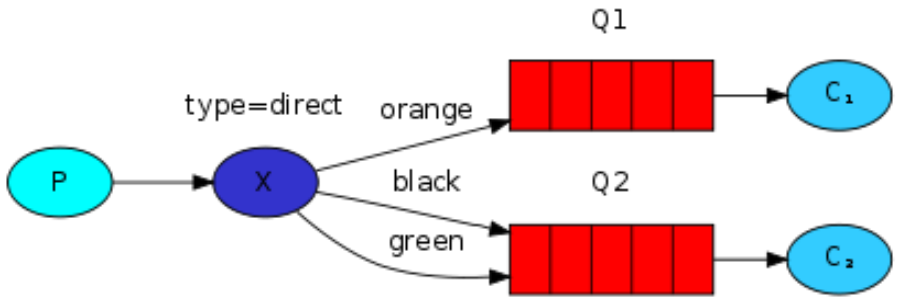
Message Queue delivery models.

Pictures from RabbitMQ tutorial

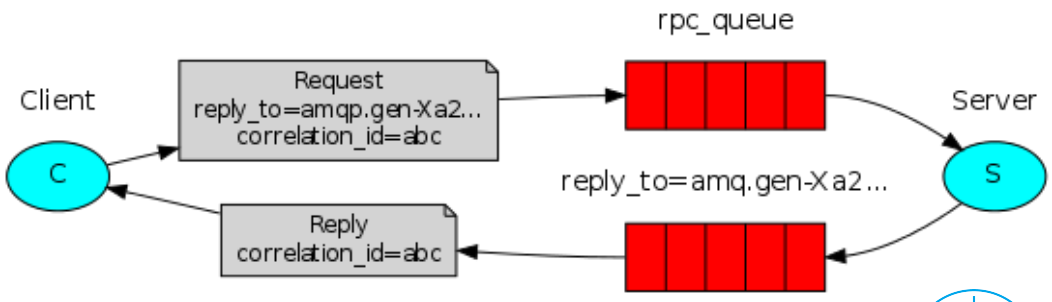








Simple consuming of messages.
Cluster of nodes consuming messages.
Publish and subscribe.
Routing message delivery.
Simulating RPC with messages.



Message queue delivery models.

- Multiple producers, one consumer.
  - Worker node reporting needed.
    - What packages need an upgrade.
- Multiple consumers.
  - Cluster of workers doing expensive tasks.
    - Install this VM image on the node.
- Message Routing.
  - This message must be delivered to “Blue” WN's.
    - Power off any worker node with no jobs.

AMQP standard for servers and clients.

- Message Orientated Middle-ware standard.
- AMQP is supported in many languages.
  - Python (I have tested this)
  - Erlang/Java/C++/Perl
  - Sorry not aware of bash!
- My examples will be in Python and RabbitMQ.

Python Message Queue send example

> How to send an AMQP message in python.

- Simple amqplib example of routed delivery.
- Uses routing key to send to clients.

```
from amqplib import client_0_8 as amqp
exchange_name = "my_exchange_name"
routing_key = "my_routing_key"
msg = "My signed message to send"
conn = amqp.Connection(host="localhost:5672", userid="guest",
       password="guest", virtual_host="/", insist=False)
chan = conn.channel()
msg = amqp.Message(msg_body)
msg.properties["delivery_mode"] = 2
chan.basic_publish(msg,exchange=exchange_name,routing_key=rou
ting_key)
chan.close()
conn.close()
```

Python Message Queue receive Example

> How to recive an AMQP message in python.

- Simple amqplib example.
- Creates two queues with routing key to send messages from first queue to second.

```
from amqplib import client_0_8 as amqp

exchange_name = "my_exchange_name"
routing_key = "my_routing_key"

def recv_callback(msg):
    print 'Received: ' + msg.body + ' from channel #' + str(msg.channel.channel_id)
    chan.basic_ack(msg.delivery_tag)

conn = amqp.Connection(host="localhost:5672", userid="guest", password="guest",
virtual_host="/", insist=False)
chan = conn.channel()
chan.queue_declare(queue="po_box", durable=True, exclusive=False, auto_delete=False)
chan.exchange_declare(exchange=exchange_name, type="direct", durable=False, auto_delete=False)
chan.queue_bind(queue=routing_key, exchange=exchange_name, routing_key=routing_key)
chan.basic_consume(queue=routing_key, no_ack=False, callback=recv_callback,
          consumer_tag="testtag2")
while True:
    chan.wait()
chan.basic_cancel("testtag")
chan.close()
conn.close()
```

Signing a message with Python and M2Crypto

> A small example of signing a message with x509 using python.

- Encryption is not very different code.

```
from M2Crypto import BIO, Rand, SMIME

content = "My message"
smime_context = SMIME.SMIME()
smime_context.load_key(signer_key,signer_cert)
buf = BIO.MemoryBuffer(content)
p7 = smime_context.sign(buf, SMIME.PKCS7_DETACHED)
buf = BIO.MemoryBuffer(content)
out = BIO.MemoryBuffer()
smime_context.write(out, p7, buf)
message_signed_with_x509 = str(out.read())
f = open(outfile, 'w')
f.write(message_signed_with_x509 )
f.close()
```

Verifying an x509 message using M2Cryto and python

> Simplified example of verification of a signed message.

- Notice this does not parse the CRL's or CA Name Space and signing policies.
- Decryption is very similar code.

```
from M2Crypto import SMIME, X509

s = SMIME.SMIME()
x509c = X509.load_cert('/etc/grid-security/certificates/dd4b34ea.0')
sk = X509.X509_Stack()
sk.push(x509c)
s.set_x509_stack(sk)
st = X509.X509_Store()
st.load_info('/etc/grid-security/certificates/dd4b34ea.0')
#st.load_info('usercert.pem')
s.set_x509_store(st)
p7, data = SMIME.smime_load_pkcs7('message_file_name')
v = s.verify(p7,data)
print v
```

Summary

- Open Source Messaging Servers are being used in large systems
  - For example Open Stack is based on RabbitMQ.
    - Open Stack data centers are believed to be very large.
- HEP Applications are being developed with signatures and/or encryption.
  - Admins should be aware of this.
- X509 Signatures and Encryption are easy to use but have pros and cons.
- X509 Signatures and Encryption are natural partners with Messaging.
- The difficult code involves handling multiple CA's and Certificates.
  - Authorization, verification signing policies/CA Name Space.
- For admins wanting to securely control remote machines at their sites little new code must be made for signing or encryption, with a messaging server.
  - Easier with a limited number of Certificates (No complex authorization)
  - Allows scalability in services with out writing your own server.

DESY

Resources

- EMI messaging guidelines.
  - https://twiki.cern.ch/twiki/bin/view/EMI/EMIMessagingGuidelines
- RabbitMQ an AMQP server. (erlang implementation in EPEL 6)
  - Http://www.rabbitmq.com/
- A simple non threaded Python AMQP library “amqplib”
  - http://code.google.com/p/py-amqplib/
- Python M2Crypto library wrapping open SSL.
  - http://chandlerproject.org/bin/view/Projects/MeTooCrypto