Quarkslab on

iMessage Privacy

HITB, Kuala Lumpur, oct. 2013

@pod2g (pod2g@quarkslab.com)
gg (gg@quarkslab.com)

Presentations

- **Quarkslab** is a research company specialized in cutting edge solutions to complex security problems. We provide innovative, efficient and practical solutions based on profound knowledge and years of experience in the field.
- **gg**: security researcher, cryptography R.E. specialist. Joined Quarkslab in 2012
- **@pod2g**: security researcher, long background in Apple product security. Joined Quarkslab in 2013

Plan

1. THE CONTEXT

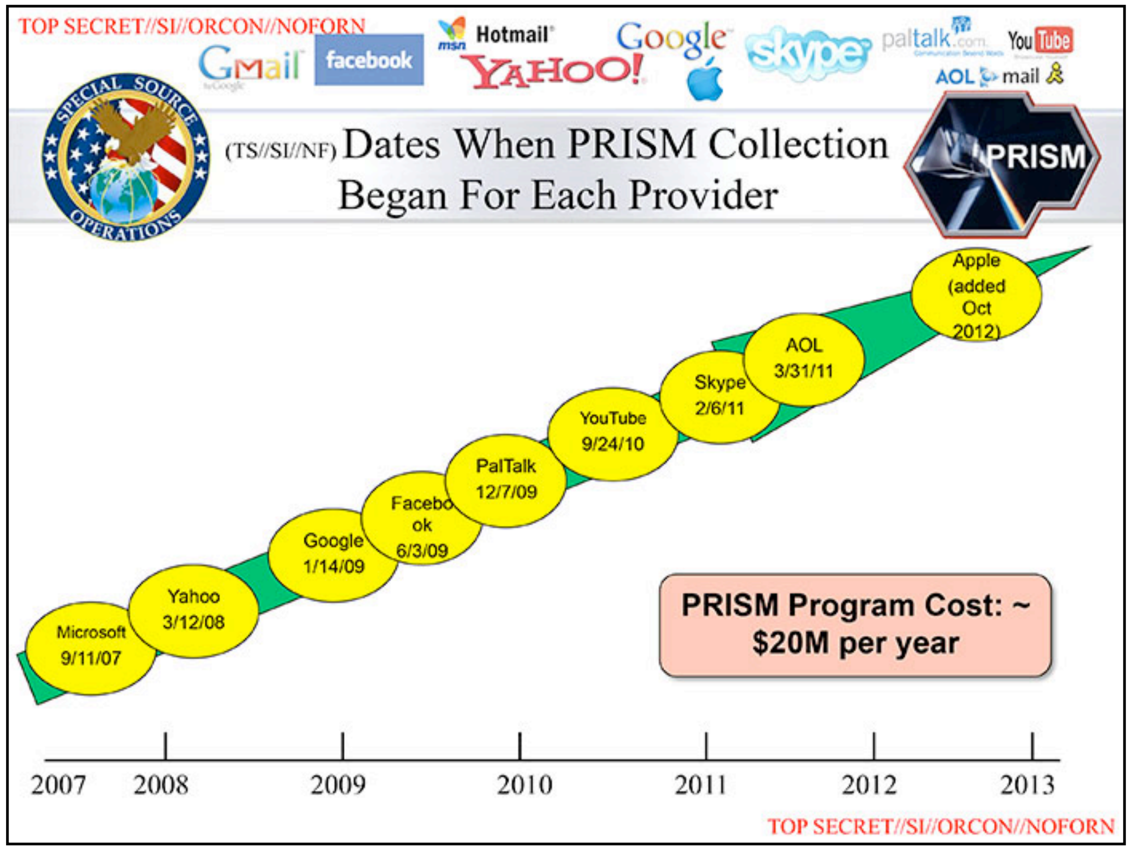
NSA, PRISM, Apple

NSA’s PRISM (US-984XN)

- American supervision program
- Mass surveillance data mining
- Based on alliances with american firms
- Can collect texts, emails, photos, etc.
- Foreigners are also potential targets
- Program was leaked by Edward Snowden

Is Apple included?

- Washington Post have leaked PRISM presentation slides that are said to be coming from the NSA
- Looking at them, Apple joined in oct 2012
- Slides talk of « data collection », which sounds like a transparent process



Apple publicly says:

« Two weeks ago, when technology companies were accused of indiscriminately sharing customer data with government agencies, Apple issued a clear response: We first heard of the government's "Prism" program when news organizations asked us about it on June 6. We do not provide any government agency with direct access to our servers, and any government agency requesting customer content must get a court order. »

Source: https://www.apple.com/apples-commitment-to-customer-privacy/

What about iMessages?

« Apple has always placed a priority on protecting our customers' personal data, and we don't collect or maintain a mountain of personal details about our customers in the first place. There are certain categories of information which we do not provide to law enforcement or any other group because we choose not to retain it.

For example, conversations which take place over iMessage and FaceTime are protected by end-to-end encryption so no one but the sender and receiver can see or read them. Apple cannot decrypt that data. »

Source: https://www.apple.com/apples-commitment-to-customer-privacy/

Media facts :-)

- Edward Snowden is said to have used iMessages to hide from the NSA :-)
- DEA Thinks iMessage Encryption is Too Tough (DEA leaked document, CNET)

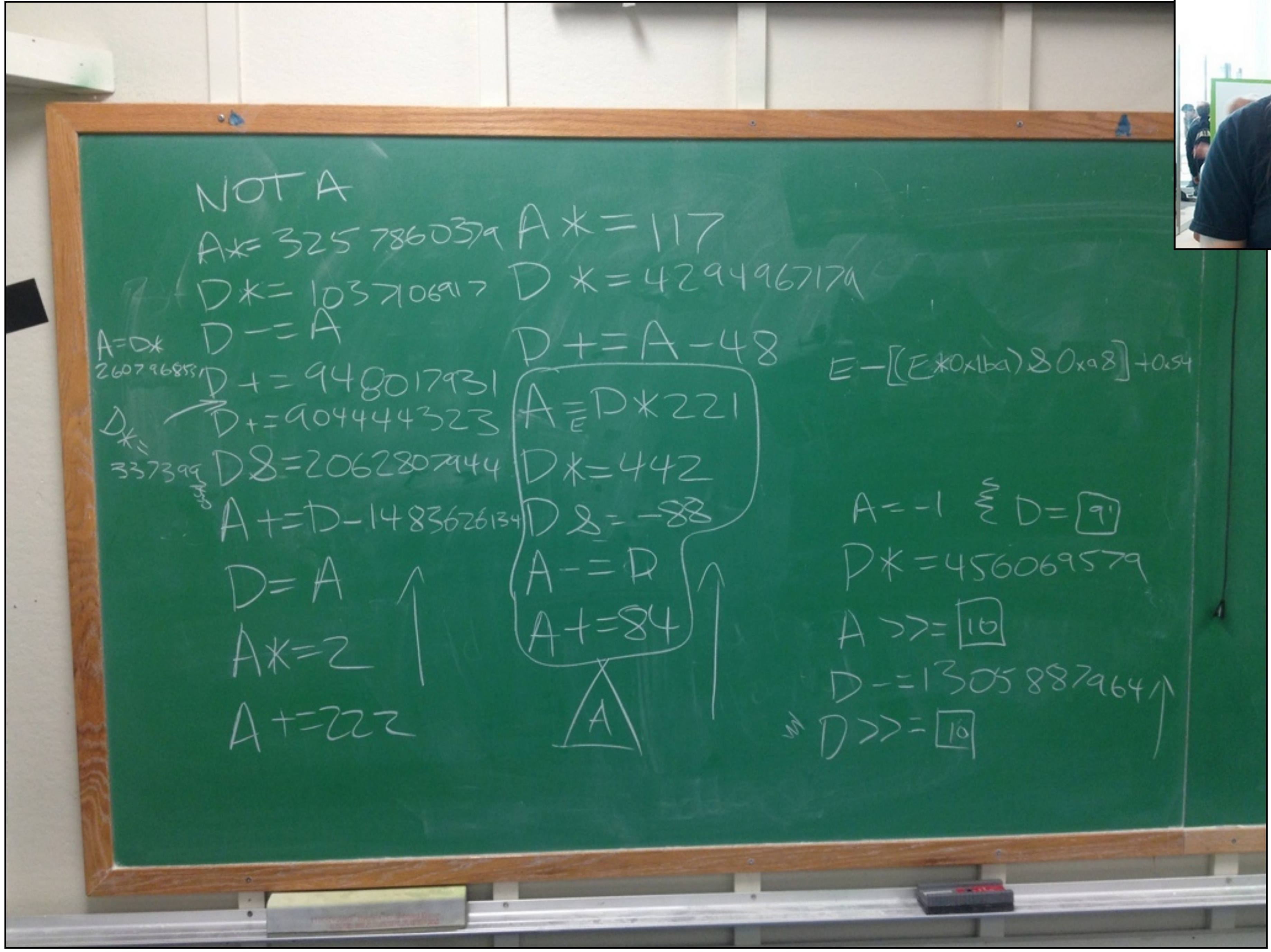
Real facts

- PUSH client (SSL server authenticates clients) certificate is 1024 bit
- iMessage encryption key is 1280 bit
- iMessage ECDSA signing key is 256 bit
- Keys are generated with the opensource Security framework
- No certificate pinning for both PUSH and iMessage servers (while Apple does it for SecureBoot and Developer certificates...)

iMessage authentication

- Heavily obfuscated. State-of-the-art white box cryptography
- Challenge-response kind
- Prevents from creating a 3rd party iMessage client on another platform
- AppleID password thrown in plaintext (not hashed) over SSL

Blackboards are not dead!

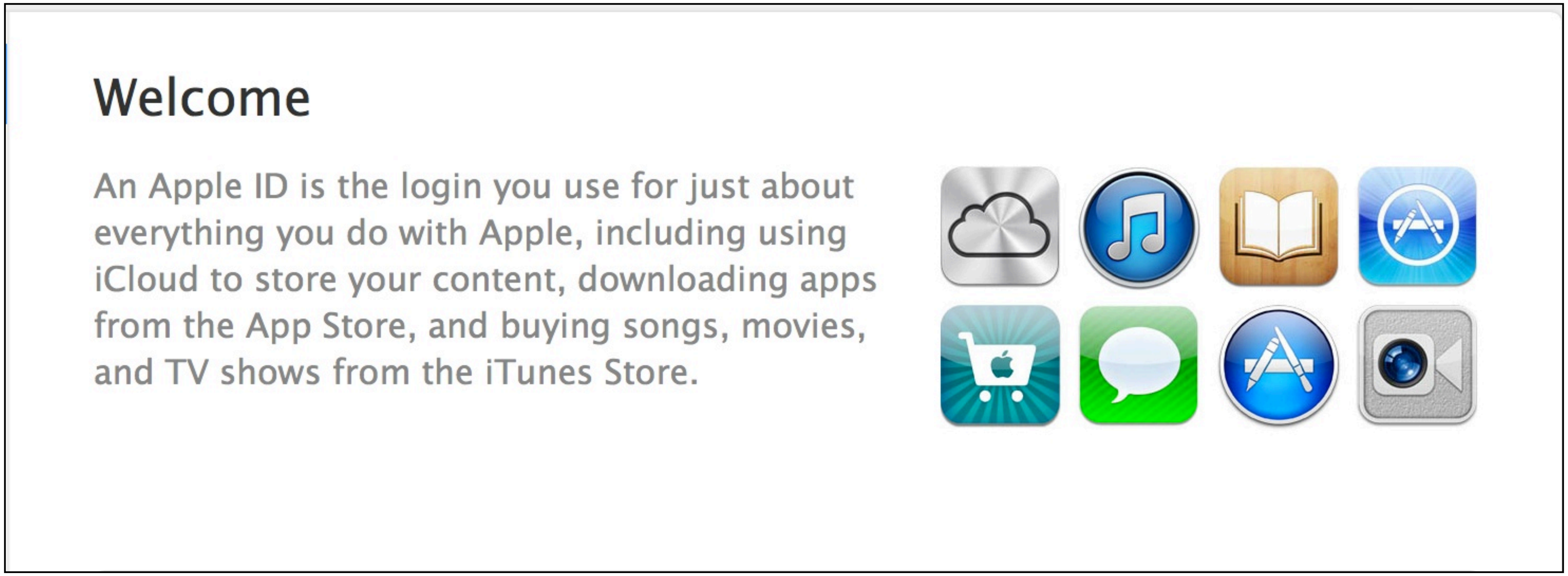


@saurik reversing imagent's white-box cryptography

First weaknesses

- Due to the lack of certificate pinning, adding a fake CA to the user keychain or obtaining a rogue (but verified) certificate for the domain leads to:
  - the leakage of the AppleID password
  - the accessibility to more sensitive Apple services
  - the possibility of impersonating Apple PUSH and iMessage servers (we'll see it later on)
- This can be done by the entity you think of, but also your company MDM administrator or hacker « friend » using a simple Mobile Configuration file

What is an AppleID?

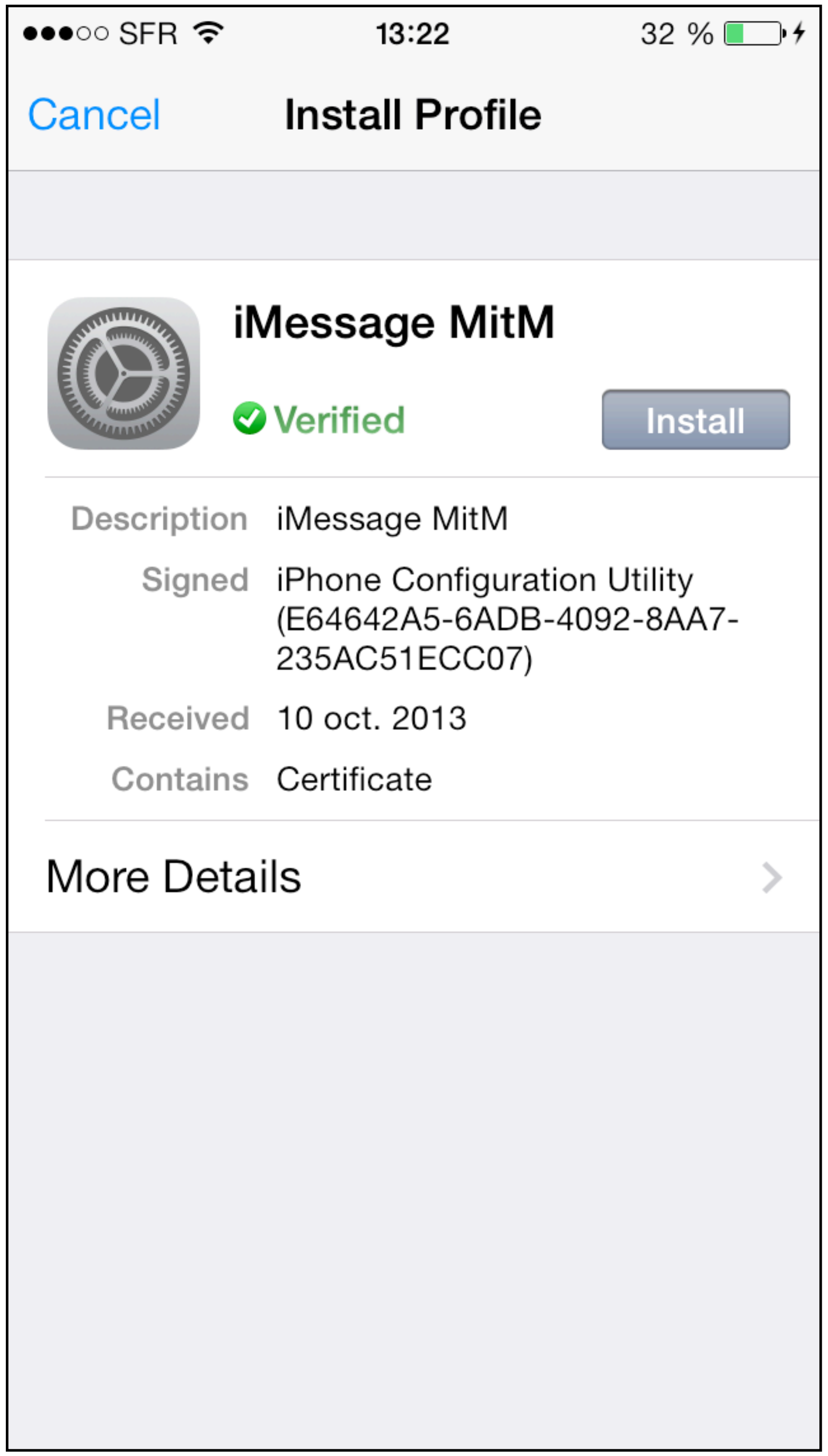


Welcome

An Apple ID is the login you use for just about everything you do with Apple, including using iCloud to store your content, downloading apps from the App Store, and buying songs, movies, and TV shows from the iTunes Store.

A really personal and sensitive information

Isn't it dodgy?



iMessage MitM
Verified

Description iMessage MitM
Signed iPhone Configuration Utility (E64642A5-6ADB-4092-8AA7-235AC51ECC07)
Received 10 oct. 2013
Contains Certificate

- The « Verified » state only depends on if the device has been plugged one time to a machine running iPhone Configuration Utility which silently adds a signing certificate to the device
- MDM administrators can do this transparently to enrolled devices

II. THE PROTOCOL

PUSH, iMessage

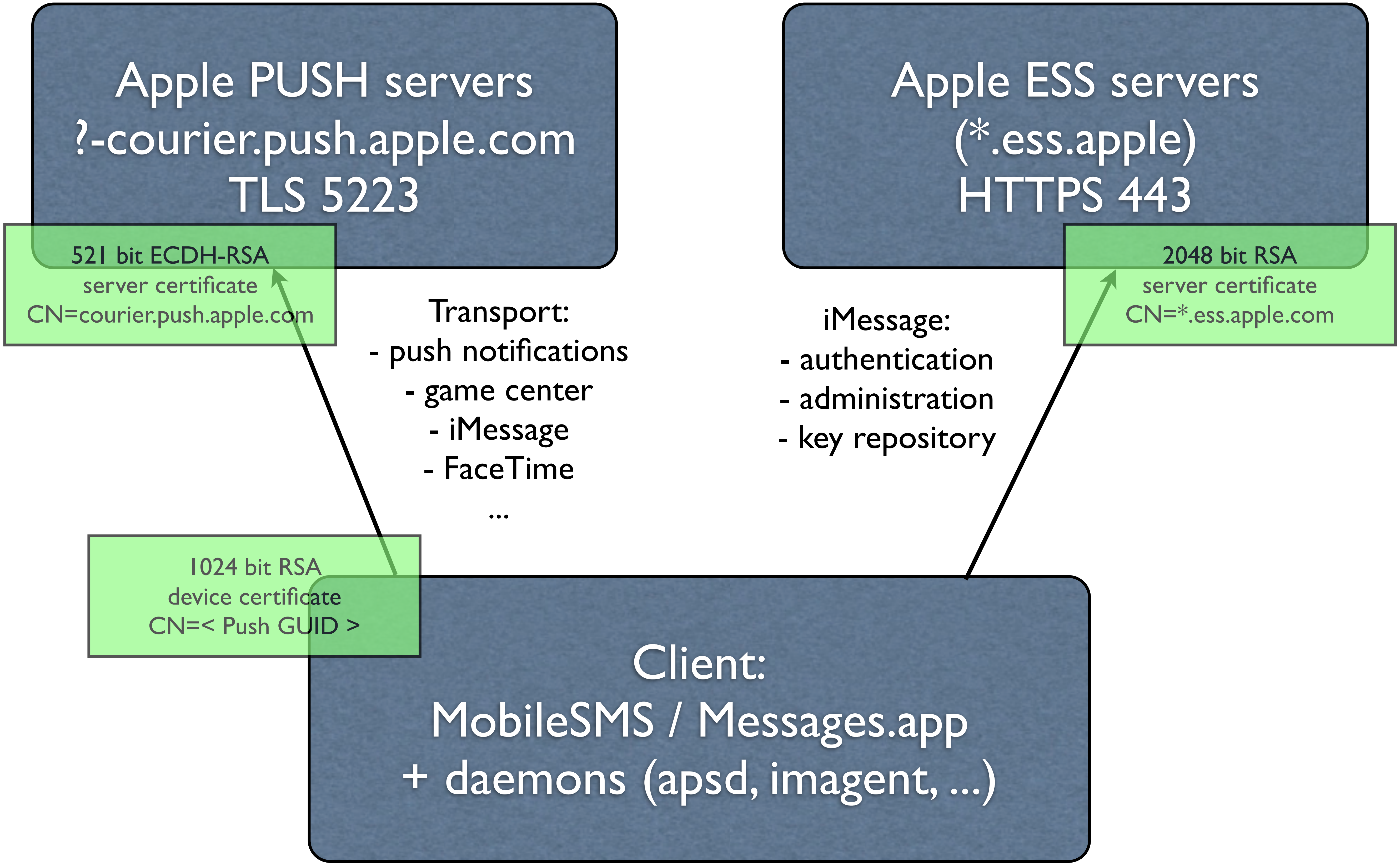
II.1. THE BIG PICTURE

Protocols and servers involved

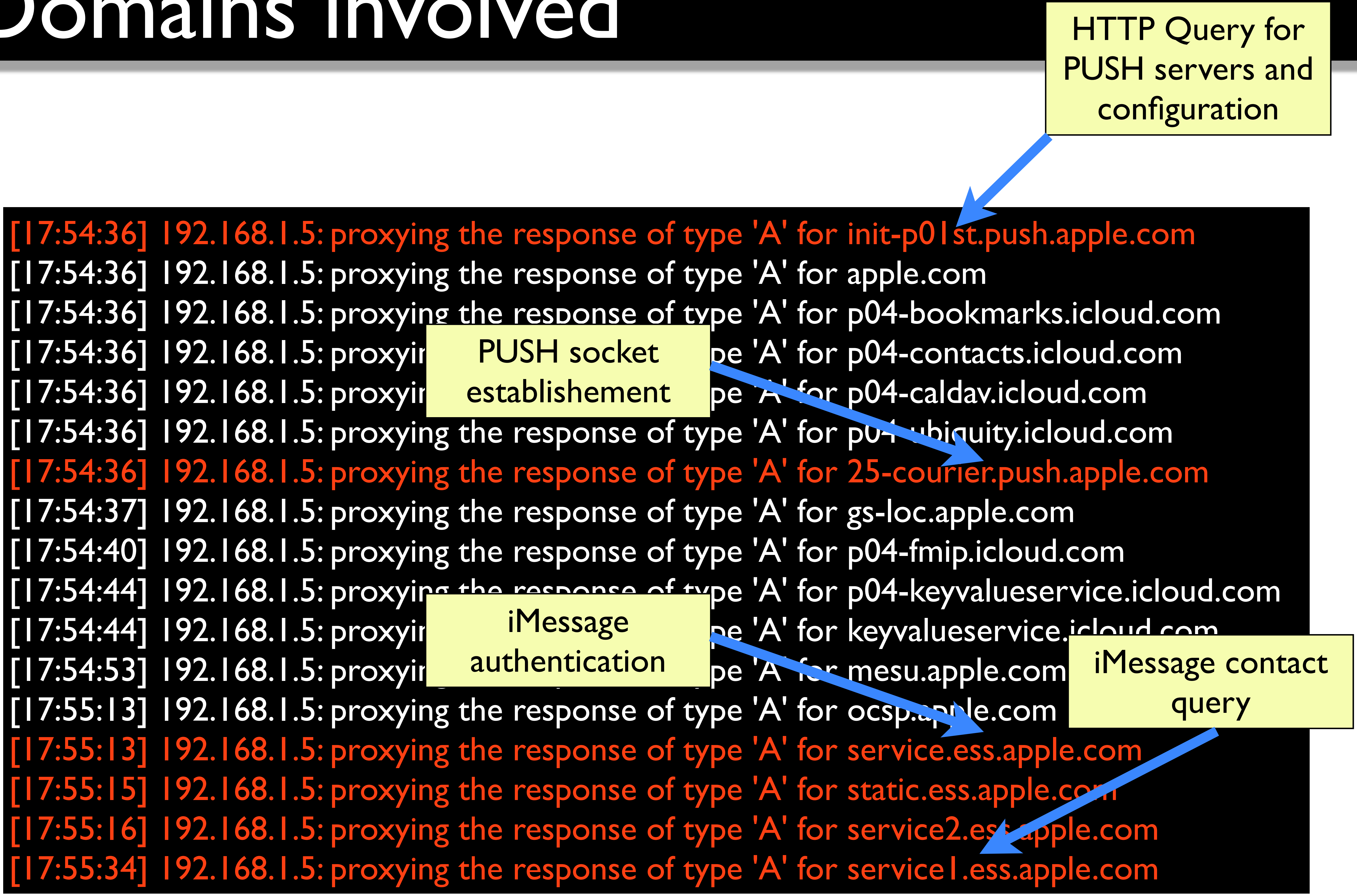
QUARKSLAB INNOVATIVE SECURITY

Two channels

- iMessages are transmitted over the PUSH protocol (TLS, server port 5223)
    - domain: rand(0,255)-courier.push.apple.com
    - client certificate is authenticated
- The rest of the protocol (authentication, administration, key and contact queries) runs over HTTPS to other servers
    - domain: *.ess.apple.com


Apple PUSH servers
?-courier.push.apple.com
TLS 5223
521 bit ECDH-RSA
server certificate
CN=courier.push.apple.com
Apple ESS servers
(*.ess.apple)
HTTPS 443
2048 bit RSA
server certificate
CN=*.ess.apple.com
Transport:
- push notifications
- game center
- iMessage
- FaceTime
...
iMessage:
- authentication
- administration
- key repository
1024 bit RSA
device certificate
CN=< Push GUID >
Client:
MobileSMS / Messages.app
+ daemons (apsd, imagent, ...)

Domains involved


HTTP Query for PUSH servers and configuration
PUSH socket establishement
iMessage authentication
iMessage contact query

```
[17:54:36] 192.168.1.5: proxying the response of type 'A' for init-p01st.push.apple.com
[17:54:36] 192.168.1.5: proxying the response of type 'A' for apple.com
[17:54:36] 192.168.1.5: proxying the response of type 'A' for p04-bookmarks.icloud.com
[17:54:36] 192.168.1.5: proxyi                             pe 'A' for p04-contacts.icloud.com
[17:54:36] 192.168.1.5: proxyi                             pe 'A' for p04-caldav.icloud.com
[17:54:36] 192.168.1.5: proxying the response of type 'A' for p04-ubiquity.icloud.com
[17:54:36] 192.168.1.5: proxying the response of type 'A' for 25-courier.push.apple.com
[17:54:37] 192.168.1.5: proxying the response of type 'A' for gs-loc.apple.com
[17:54:40] 192.168.1.5: proxying the response of type 'A' for p04-fmip.icloud.com
[17:54:44] 192.168.1.5: proxying the response of type 'A' for p04-keyvalueservice.icloud.com
[17:54:44] 192.168.1.5: proxyi                             pe 'A' for keyvalueservice.icloud.com
[17:54:53] 192.168.1.5: proxyi                             pe 'A' for mesu.apple.com
[17:55:13] 192.168.1.5: proxying the response of type 'A' for ocsp.apple.com
[17:55:13] 192.168.1.5: proxying the response of type 'A' for service.ess.apple.com
[17:55:15] 192.168.1.5: proxying the response of type 'A' for static.ess.apple.com
[17:55:16] 192.168.1.5: proxying the response of type 'A' for service2.ess.apple.com
[17:55:34] 192.168.1.5: proxying the response of type 'A' for service1.ess.apple.com
```

II.2. THE PUSH LAYER

History, details, and man-in-the-middle

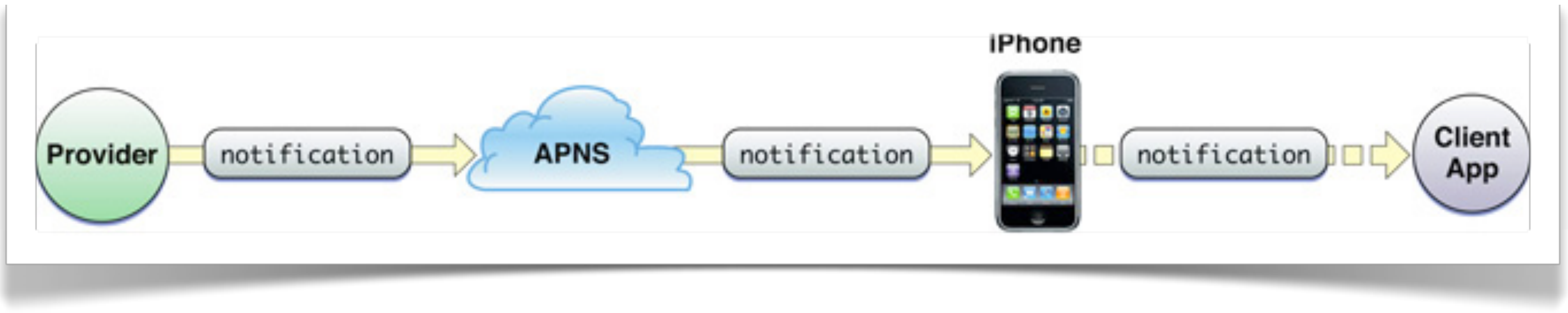
QUARKSLAB INNOVATIVE SECURITY

Introduction

- Apple Push Notification Service (APNS)
- Service created by Apple Inc. in 2009
- Enhance the user experience with notifications like sounds, text alerts, etc.
- Available as an API
- Better than PULL for battery life

How it works

- Based on push technology
- Maintain an open IP connection to forward notifications from the servers of third party applications to Apple devices



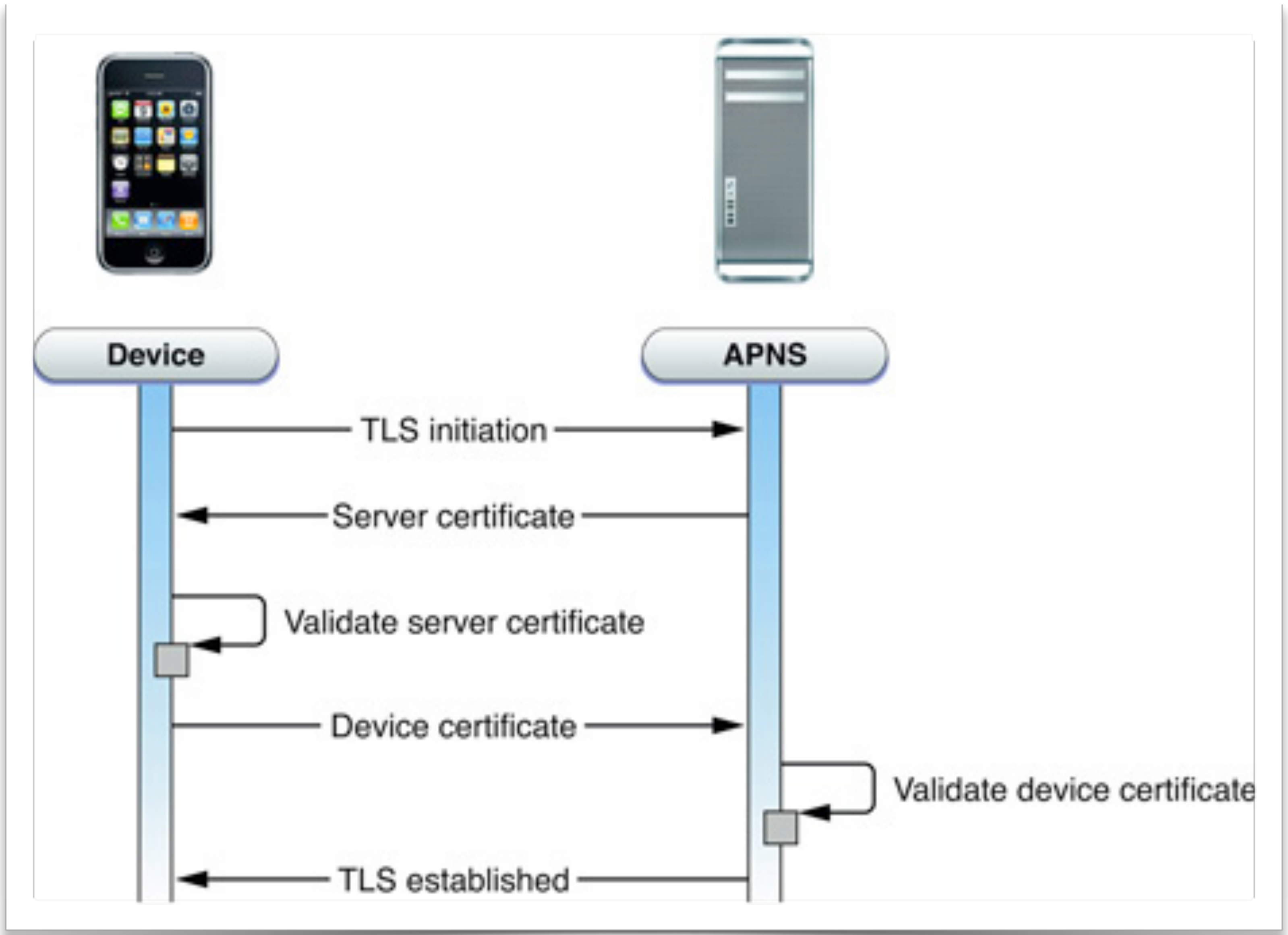
PUSH Client

- Device communicates with the PUSH server
- Distant port : 5223 (TCP)
- Traffic encrypted with TLS
- Requires a Push-Token
- Requires a Push-Certificate

PUSH device certificate

- Generated on device APN activation
- Certificate request sent to *albert.apple.com*
- Signed by *Apple Iphone Device CA*
- Used to establish PUSH TLS communication

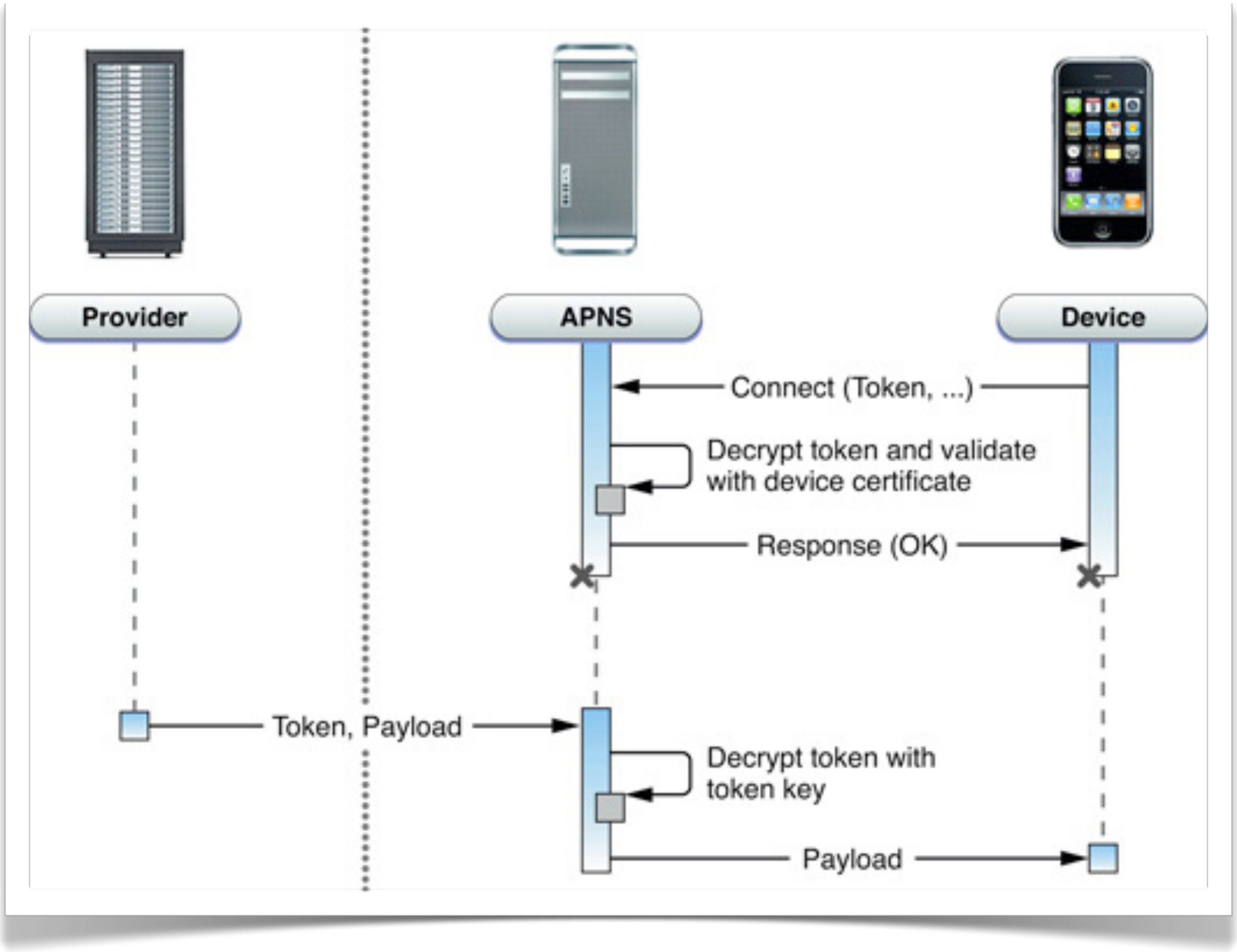
Mutual authentication



Push-Token

- 256-bit binary string
- Opaque, server-side generated
- Identifier to route notifications to devices
- Shared with providers

Token usage


Provider
APNS
Device
Connect (Token, ...)
Decrypt token and validate with device certificate
Response (OK)
Token, Payload
Decrypt token with token key
Payload

MitM: PushProxy - 1

- Catch PUSH communications
- Decode notifications in a readable form
- Provide APIs for handling and sending
- More info: https://github.com/meeee/pushproxy

MitM: PushProxy - 2

How to:

- Generate a Root CA and add it to the keychain
- Create and sign all the required certificates with it (APNS server, HTTPS bag server)
- Extract device TLS private key (a.k.a. device certificate)
- Edit hosts file to redirect DNS queries to PushProxy (or use a rogue DNS server)

Outgoing iMessage notification

0a	>> Message Type
XX XX XX XX	>> Next Length
04 00 04 XX XX XX XX	>> Identifier (4 bytes)
01 00 14 XX XX	>> Topic Hash (20 bytes)
02 00 20 XX XX	>> Push-Token (32 bytes)
03 XX XX ...	>> iMessage payload

II.3. IMESSAGE IDs

Tokens, keys, URIs, directory

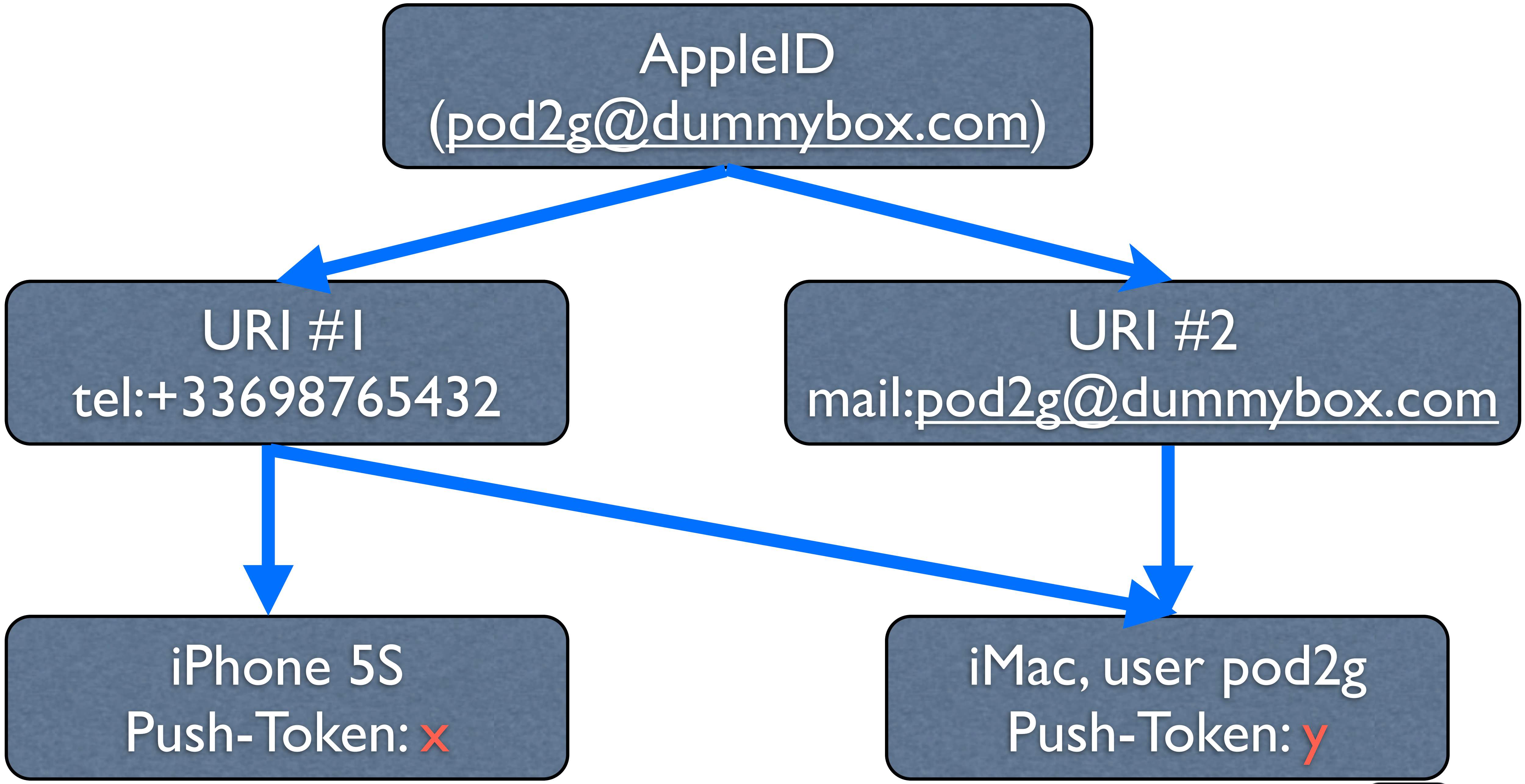
Definitions

- AppleID: Apple identifier of a person (or a legal entity). Most people have a single AppleID that they use on multiple Apple devices and services
- URI: recipient identifier, either a phone number or email address
- Push-Token: token identifying an authenticated device
- For OS X, a « device » is a user account set-up to receive iMessages

Organization - I

- An Apple account (AppleID) can be linked to multiple URIs
- Email URIs have to be verified
- A phone URI can only be added with an iPhone and the corresponding legit SIM card
- The same URI can’t be attached to multiple AppleIDs
- A URI can be linked to multiple devices (Push-Tokens)
- On each device the user can decide which URI to handle

Organization - 2



URI & iMessage

- Sender inputs the recipient’s URI to start the communication
- All iMessages are encrypted and signed using asymmetric cryptography
- Thus, there has to be a key directory
- iMessage client retrieves recipient’s public keys by querying Apple’s ESS server

An example of contact query

Here is what is sent:

```
GET /WebObjects/QueryService.woa/wa/query?uri=tel:+33123456789
Host: service1.ess.apple.com
Content-Type: application/x-apple-plist
x-id-cert: [Provision Certificate]
x-id-nonce: [Random Nonce with Timestamp]
x-id-sig: [Query Signed with Provision Cert.]
```

Response

Here is what we get, for each associated device:
(XML plist converted to JSON for clarity)

```
{
  'push-token': [PushToken]
  ‘client-data’:
  {
    'show-peer-errors': True,
    'public-message-identity-version': 1.0,
    'public-message-identity-key': [Public Keys Buffer]
  }
}
```

Response analysis

- The public keys buffer contains:
  - An ECDSA public key (256-bit): to verify messages issued by the remote device
  - A RSA public key (1280-bit): to encrypt messages for the remote device
- Push-Token will help to route messages
- We can now send and encrypt messages to a given URI (and all devices associated with)!

II.4. THE IM PAYLOAD

Description, goodies

An iMessage is a bplist

- The iMessage payload as seen earlier in the PUSH Protocol section is a binary plist
- Binary plist (a.k.a. bplist) is an Apple standard property list file
- A bplist stores serialized objects
- Objects can be of type NSString, NSNumber, NSDate, NSData, NSArray and NSDictionary
- Serializes an NSDictionary as the root object

iMessage bplist - 1

D: True

E: ‘pair’

P: <variable length binary data> *(iMessage payload, deflate compressed)*

U: <128bit binary data> *(iMessage UID)*

c: 100

i: <32bit integer> *(messageId, same as in PUSH header)*

iMessage bplist - 2

sP: mailto:pod2g@dummybox.com *(emitter URI)*

t: <256bit binary data> *(emitter Push-Token)*

tP: mailto:dhillon@dummybox.com *(recipient URI)*

ua: [Mac OS X,10.8.5,12F37,MacBookPro10,2]
(emitter os and hardware version)

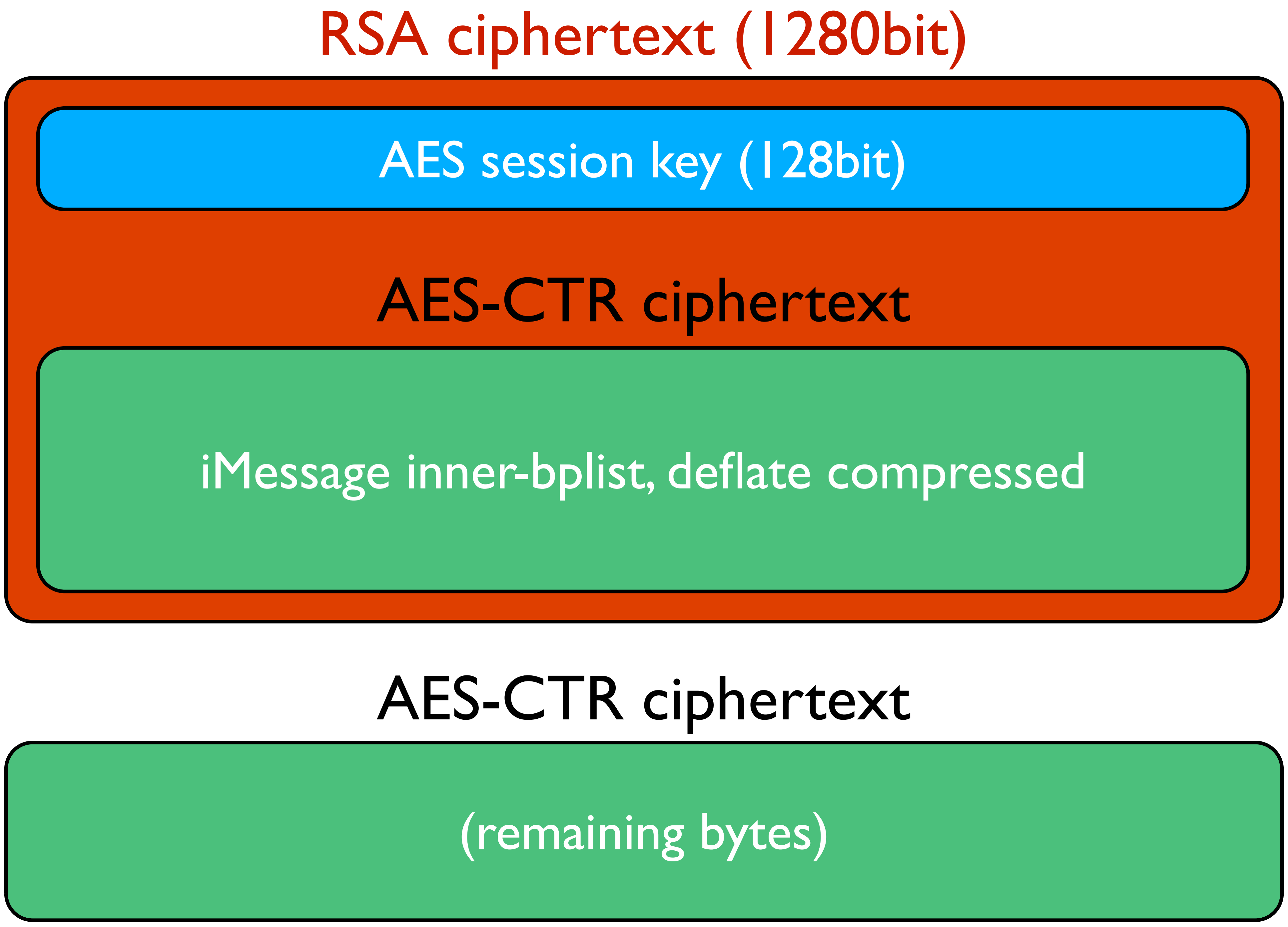
v: 1

iMessage payload, inflated

byte	**0x02**	version?
short	**ciphertext length**	
data	**ciphertext**	RSA / AES-CTR data - ciphered with the RSA public key of the recipient
byte	**signature length**	
data	**signature**	ECDSA signature of <ciphertext> - computed with the ECDSA private key of the emitter

Why did they decided to deflate ciphered data?

iMessage ciphertext



iMessage inner-bplist (inflated)

p: array of URIs in the discussion group

t: iMessage text (for iOS)

v: version (1)

x: iMessage html (attachments, and style - for OS X)

iMessage attachments

- Attachments are encrypted using AES
- They are uploaded to the iCloud storage, in a dedicated space
- URL of the attachment and the required AES key to decipher it are included in the special tag <FILE> added to the HTML message body
- `<FILE name="`***`<name>`***`" width="`***`<width>`***`" height="`***`<height>`***`" datasize="`***`<size>`***`" mime-type="`***`<mime type>`***`" uti-type="`***`<uti type>`***`" mmcs-owner="`***`<identifier>`***`" mmcs-url="`***`<URL>`***`" mmcs-signature-hex="`***`<signature>`***`" file-size="`***`<size>`***`" decryption-key="`***`<key>`***`">`

Spoofing URIs

- The existence of URIs in the discussion group are not fully verified:
  - The real recipient URI has to be in the list
  - Other URIs are not checked
  - Phone number URIs can be text
- The result is a spoofing kind of vulnerability

DEMO #1

Conference chat with a surprise guest :)

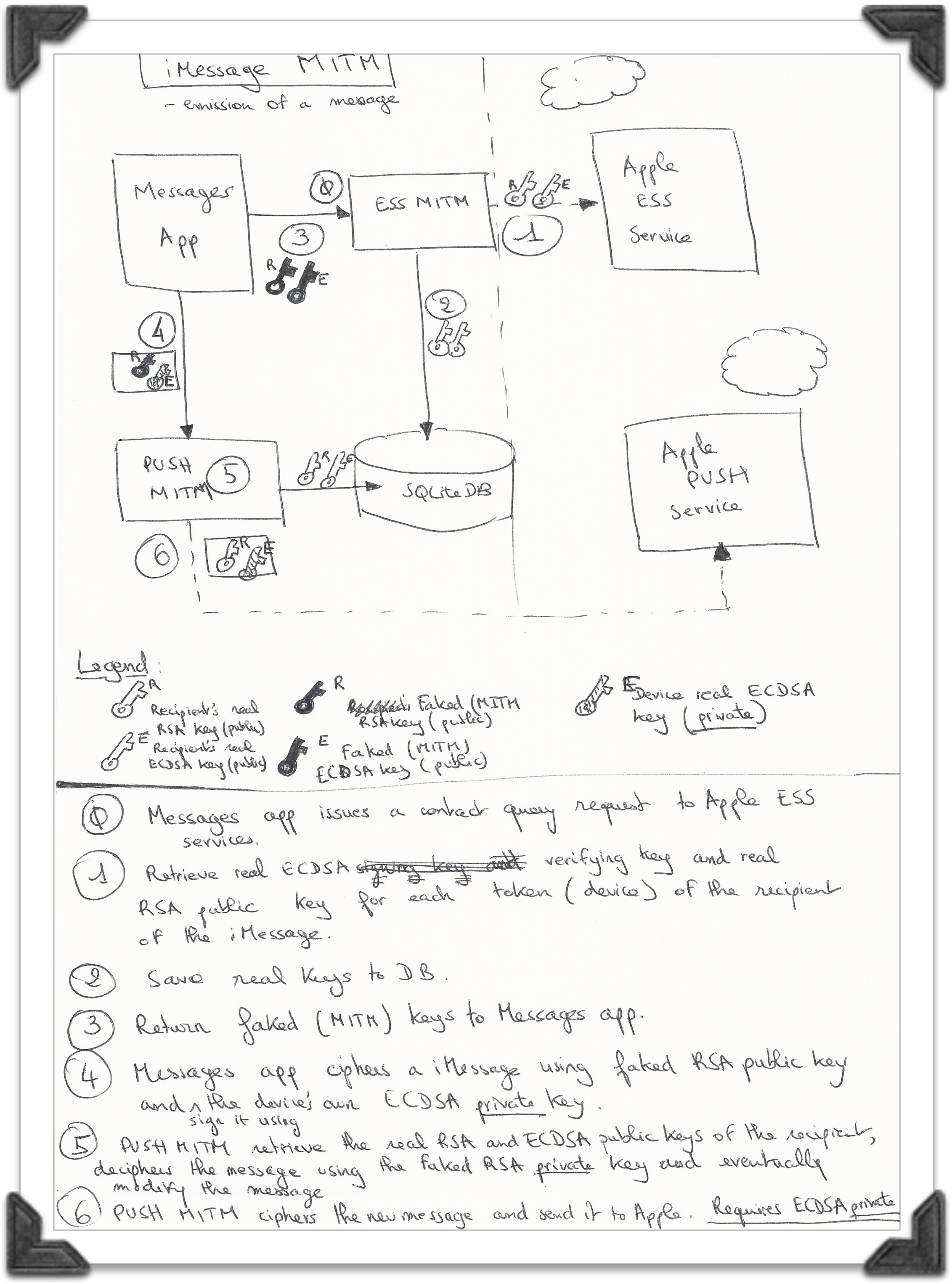
III. MITM ATTACKS

iMessage interception and forgery

III.1. INTRODUCTION

Requirements, network tricks, and software

Original Quarkslab document



Legend:

A Recipient's real RSA key (public)

E Recipient's real ECDSA key (public)

R ~~[illegible]~~ Faked (MITM RSA key (public)

E Faked (MITM) ECDSA key (public)

E Device real ECDSA key (private)

0 Messages app issues a contact query request to Apple ESS services.

1 Retrieve real ECDSA ~~signing key and~~ verifying key and real RSA public key for each token (device) of the recipient of the iMessage.

2 Save real keys to DB.

3 Return faked (MITM) keys to Messages app.

4 Messages app ciphers a iMessage using faked RSA public key and sign it using the device's own ECDSA private key.

5 PUSH MITM retrieve the real RSA and ECDSA public keys of the recipient, deciphers the message using the faked RSA private key and eventually modify the message

6 PUSH MITM ciphers the new message and send it to Apple. Requires ECDSA private

Requirements - DNS

- To achieve a man-in-the-middle attack, the DNS requests of the victims have at least to pass through a machine / network you control
- The point is to rogue responses for domains *service1.ess.apple.com* and **.push.apple.com*
- Next slide shows possible network tricks to have a chance to forge DNS responses

Some network tricks

- Use ARP poisoning to route all ethernet packets of the victim to your box
- Have access to physical cables, to the gateway, or any network component in the route to the DNS server
- Create a rogue (open) wifi network or 3G network with the same name and emit stronger
- Announce Apple's routes with BGP, and reroute the traffic to your own equipments (unlikely)
- Hack the DNS servers using DNS cache poisoning and other DNS related vulnerabilities (unlikely)

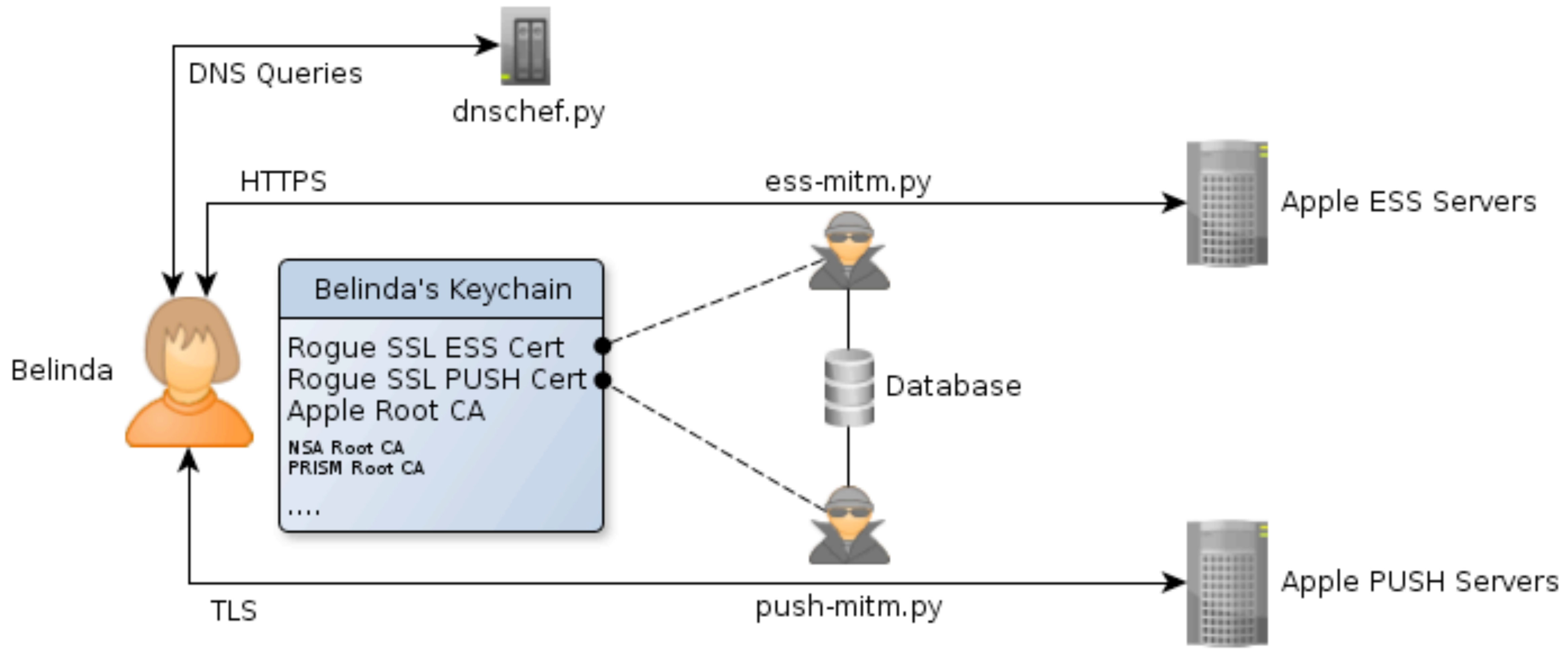
Requirements - software

- PushProxy with Quarkslab's *imessage-mitm.py* handler to intercept iMessages
- Quarkslab's *ess-mitm.py* to intercept and modify Apple ESS responses
- A DNS proxy software. We used *dnschef* in our tests: http://thesprawl.org/projects/dnschef/
- Python 2.7

Requirements - SSL

- Rogue servers, either PUSH or ESS, have to serve valid SSL certificates from the point of view of the victim(s)
- Either add these certificates or their root CA to the victim's KeyChain
- Find a flow in Apple certificate verification (unlikely?)
- Have the user install a configuration profile (or be a MDM administrator in the company and push it)
- Have a trusted root sign your rogue certs (NSA?)

Picture


DNS Queries
dnschef.py
HTTPS
ess-mitm.py
Apple ESS Servers
Belinda
Belinda's Keychain
Rogue SSL ESS Cert
Rogue SSL PUSH Cert
Apple Root CA
NSA Root CA
PRISM Root CA
....
Database
Apple PUSH Servers
TLS
push-mitm.py

III.2. ONE-SIDED MITM

Prerequisites, theory, demo

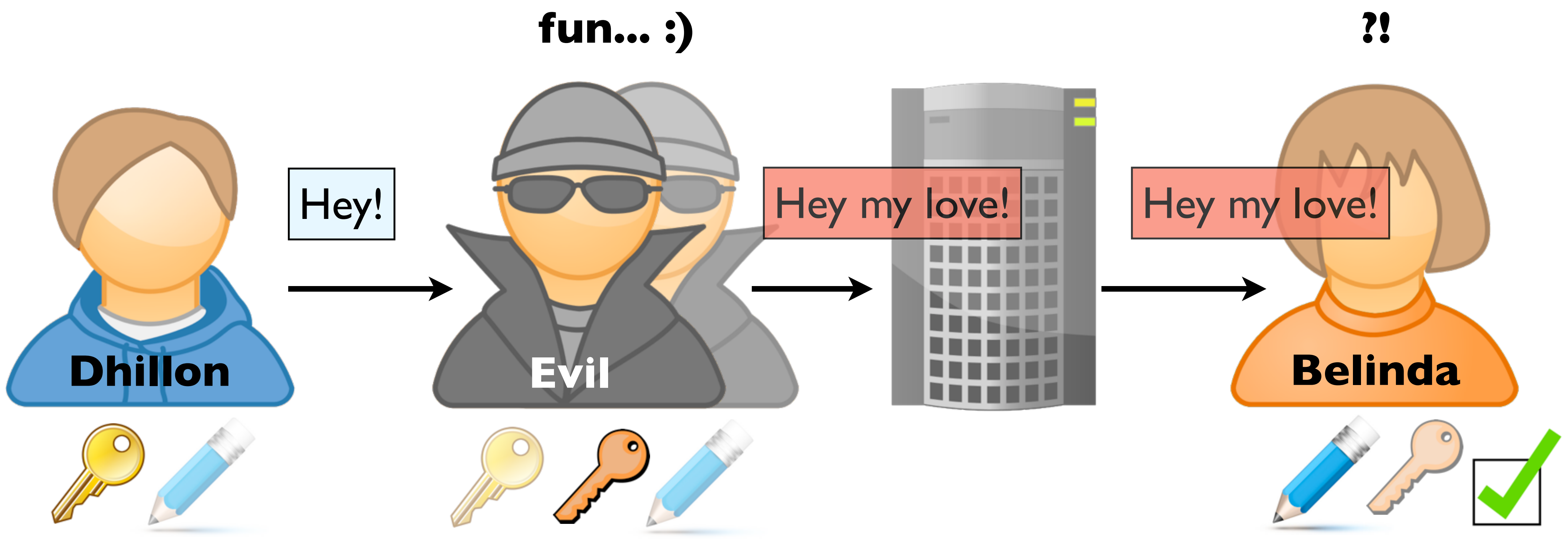
Principle & limitations

- Idea: proxify victim's contact requests to Apple's ESS server in order to exchange every public key found with a new one. « Evil's » in the figure to come
- The victim's PUSH communication is also proxyfied and is utilized to eavesdrop iMessages in real time, and possibly modify them
- The biggest limitation to this approach is that the victim's private keys (PUSH device, iMessage RSA, iMessage ECDSA) are needed

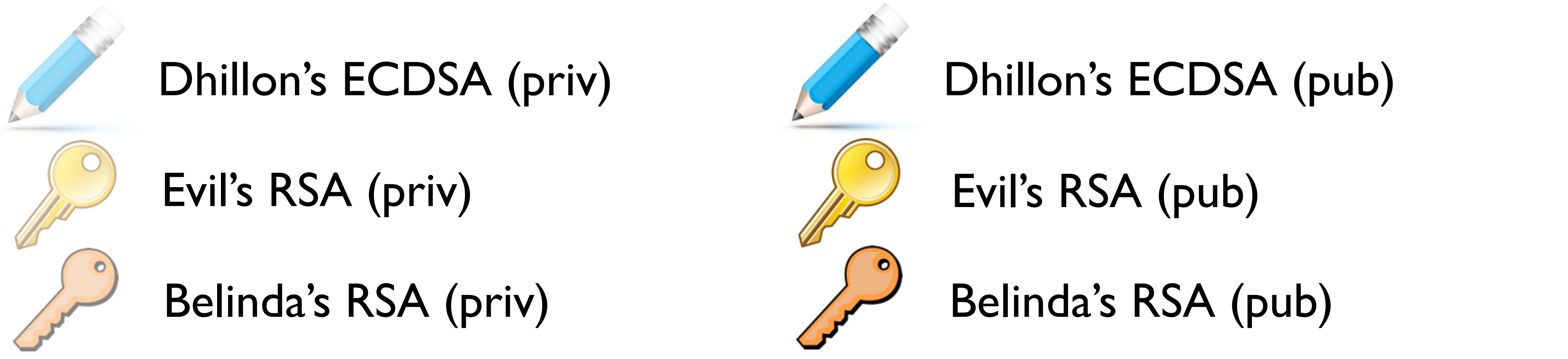
Requirements

- Network / DNS control
- Verified PUSH and ESS certificates
- Victim’s PUSH Device private key
- Victim’s iMessage RSA & ECDSA private keys

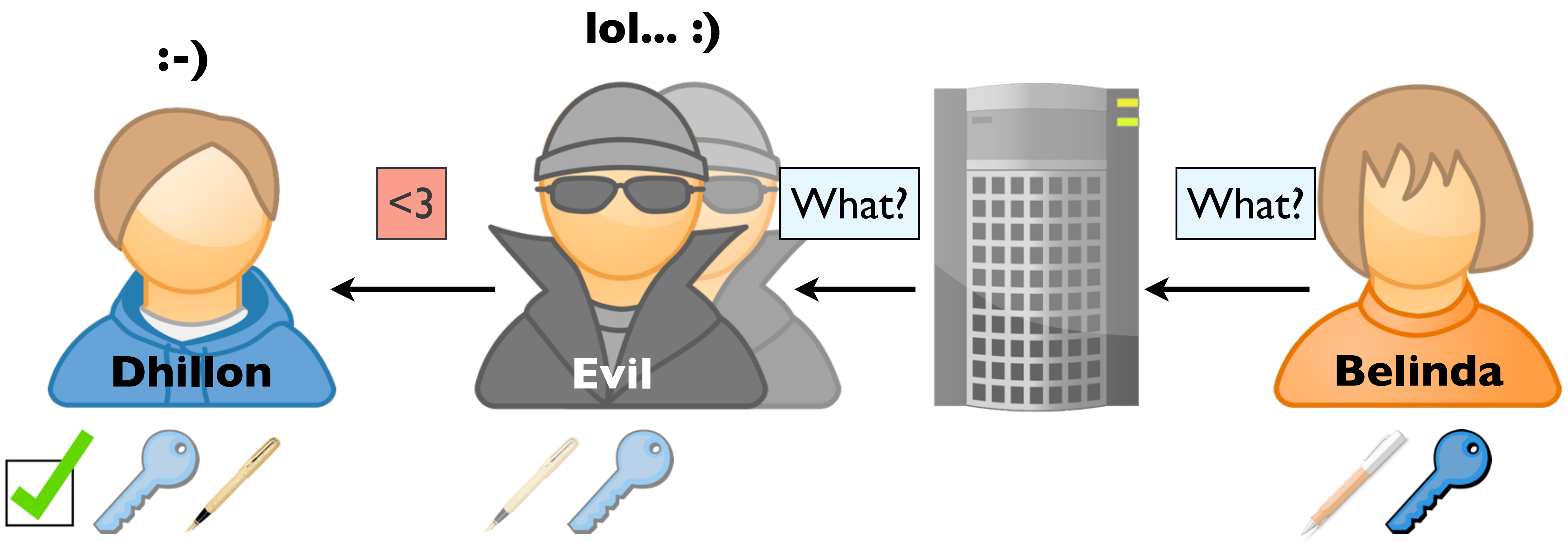
iMessage emission MitM



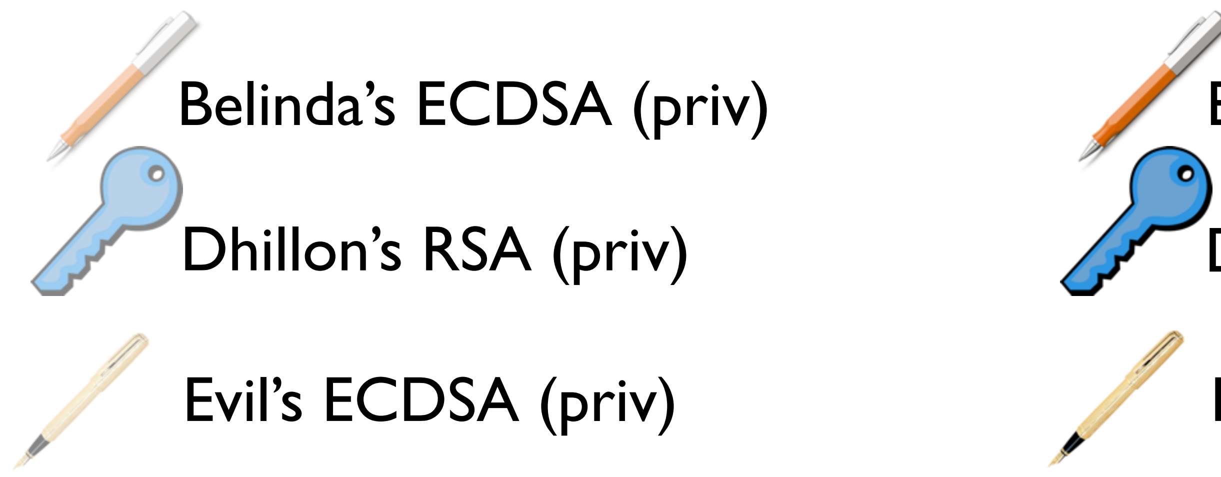
Evil presented his key to Dhillon instead of Belinda's.
He owns Dhillon's ECDSA. He can read and forge Dhillon's messages.



iMessage reception MitM



Evil owns Dhillon's RSA private key. He can read his messages.
Evil presented his ECDSA key instead of Belinda's. He can forge messages.



DEMO #2

Intercepting iMessages OTA

III.3. TWO-SIDED MITM

Prerequisites, theory

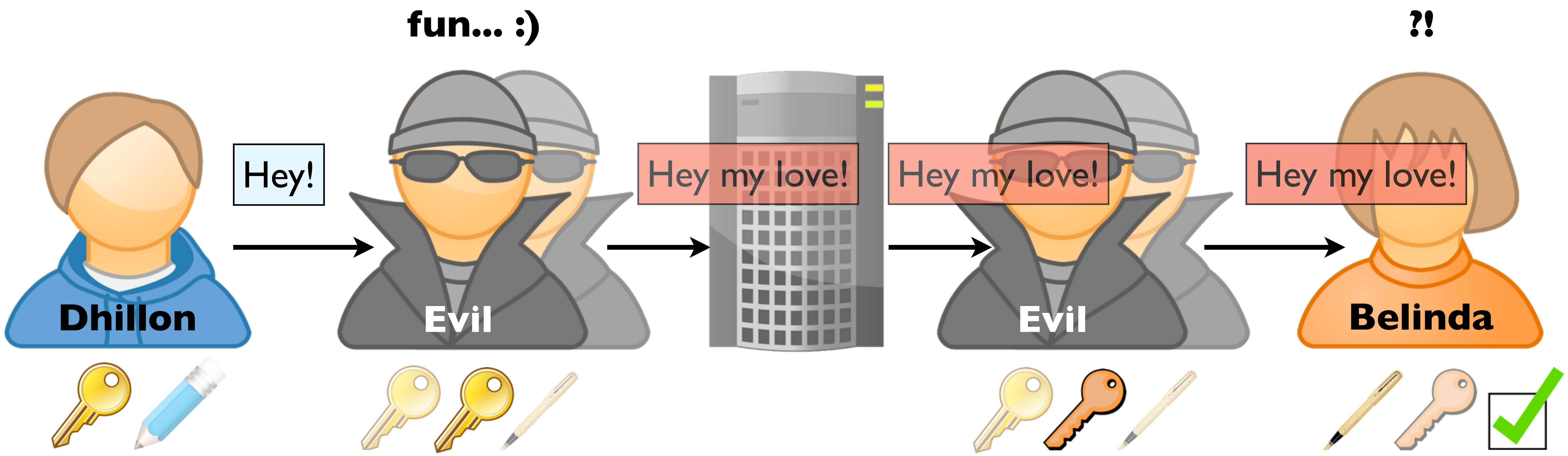
Principle & limitations

- Idea: proxify all victims contact requests to Apple's ESS server in order to exchange every public key found with a new one. « Evil's » in the figure to come
- Victims' PUSH communications are also proxyfied and are utilized to eavesdrop iMessages in real time, and possibly modify them
- The two-sided implementation is unpractical in terms of network control requirements and the PUSH device private keys of the victims are needed

Requirements

- (Great network / DNS control) * *N victims*
- (Verified PUSH and ESS certificates) * *N victims*
- (Victim’s PUSH Device private key) * *N victims*

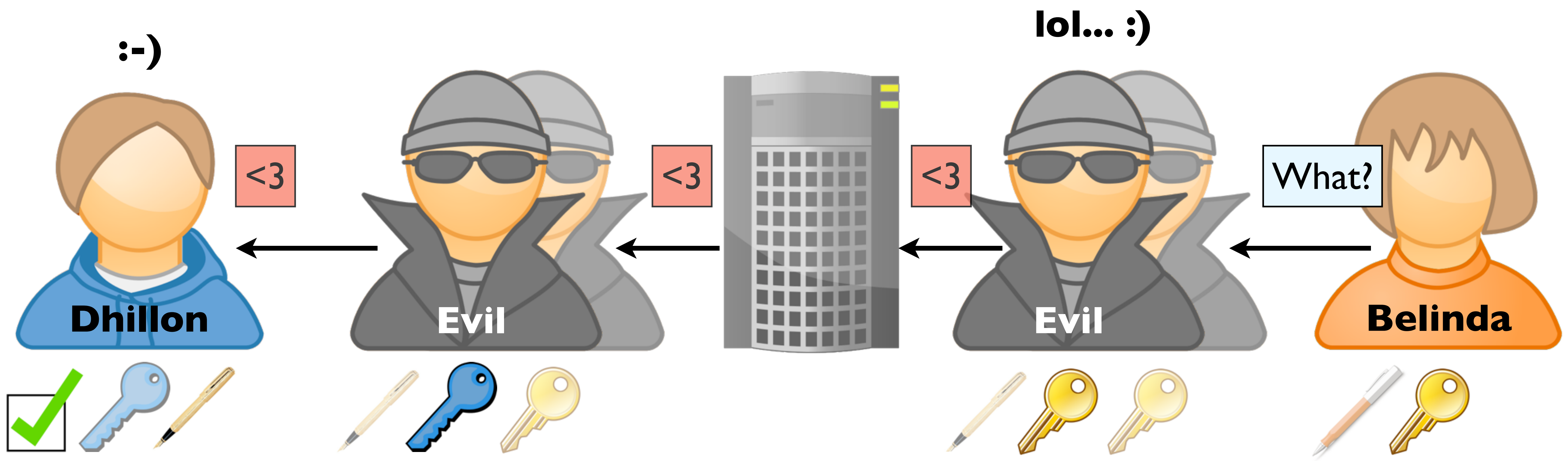
iMessage emission MitM



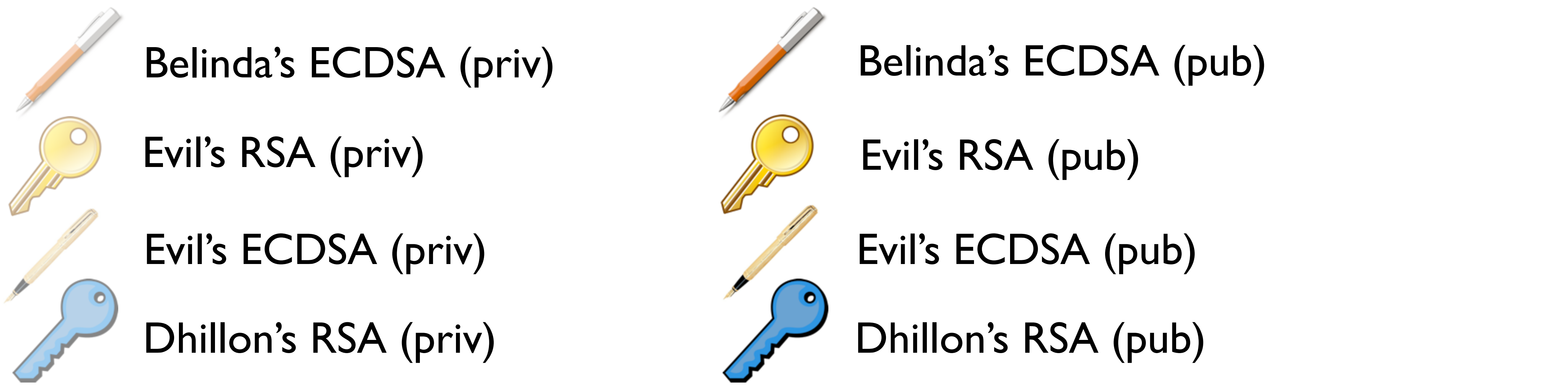
Evil presented his key to Dhillon instead of Belinda's.
Evil presented his key to Belinda instead of Dhillon's.
He can read and forge Dhillon's messages without any of his private keys.

Dhillon's ECDSA (priv)	Dhillon's ECDSA (pub)
Evil's RSA (priv)	Evil's RSA (pub)
Evil's ECDSA (priv)	Evil's ECDSA (pub)
Belinda's RSA (priv)	Belinda's RSA (pub)

iMessage reception MitM



Evil presented his key to Dhillon instead of Belinda's.
Evil presented his key to Belinda instead of Dhillon's.
He can read and forge Belinda's messages without any of her private keys.



III.4. APPLE BYPASS

Prerequisites, theory

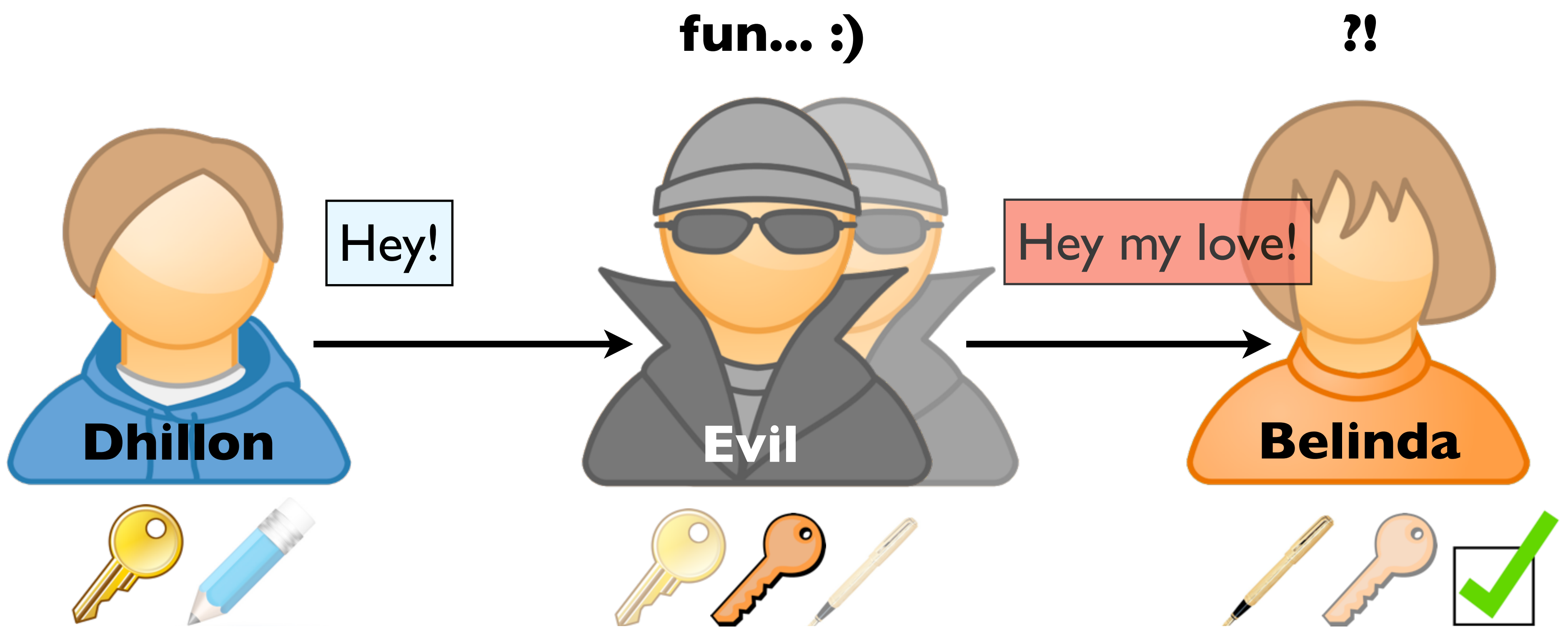
Principle & limitations

- Basically the same as previous one, except that real Apple’s servers are never used as a transport
- Same limitations as the classical two-sided implementation: great network control is required, but absolutely no victims’ private keys are needed

Requirements

- (Great network / DNS control) * *N victims*
- (Verified PUSH and ESS certificates) * *N victims*
- *Any idea who have access to these requirements?*
  - *Multiple possible answers would work ;-)*

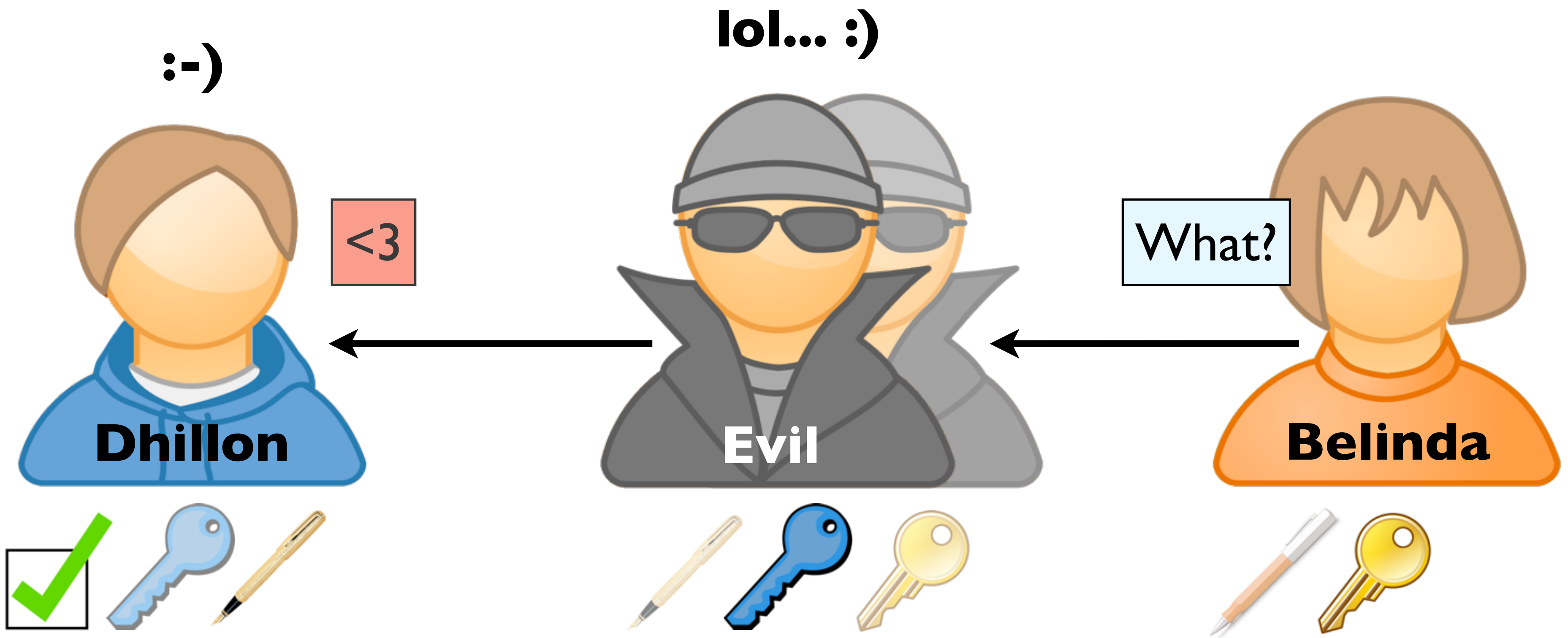
iMessage emission MitM



Evil presented his key to Dhillon instead of Belinda's.
Evil presented his key to Belinda instead of Dhillon's.
He can read and forge Dhillon's messages without any of his private keys.

✎	Dhillon's ECDSA (priv)	✎	Dhillon's ECDSA (pub)
🔑	Evil's RSA (priv)	🔑	Evil's RSA (pub)
✒	Evil's ECDSA (priv)	✒	Evil's ECDSA (pub)
🔑	Belinda's RSA (priv)	🔑	Belinda's RSA (pub)

iMessage reception MitM



Evil presented his key to Dhillon instead of Belinda's.
Evil presented his key to Belinda instead of Dhillon's.
He can read and forge Belinda's messages without any of her private keys.

🖊	Belinda's ECDSA (priv)	🖊	Belinda's ECDSA (pub)
🔑	Evil's RSA (priv)	🔑	Evil's RSA (pub)
🖊	Evil's ECDSA (priv)	🖊	Evil's ECDSA (pub)
🔑	Dhillon's RSA (priv)	🔑	Dhillon's RSA (pub)

III.5. BEING APPLE

Any requirements?

Requirements?

- Apple has full control over the ESS public key directory, no need to hijack anything
- Swapping keys is transparent for the user, they are never shown anywhere in Messages.app / MobileSMS

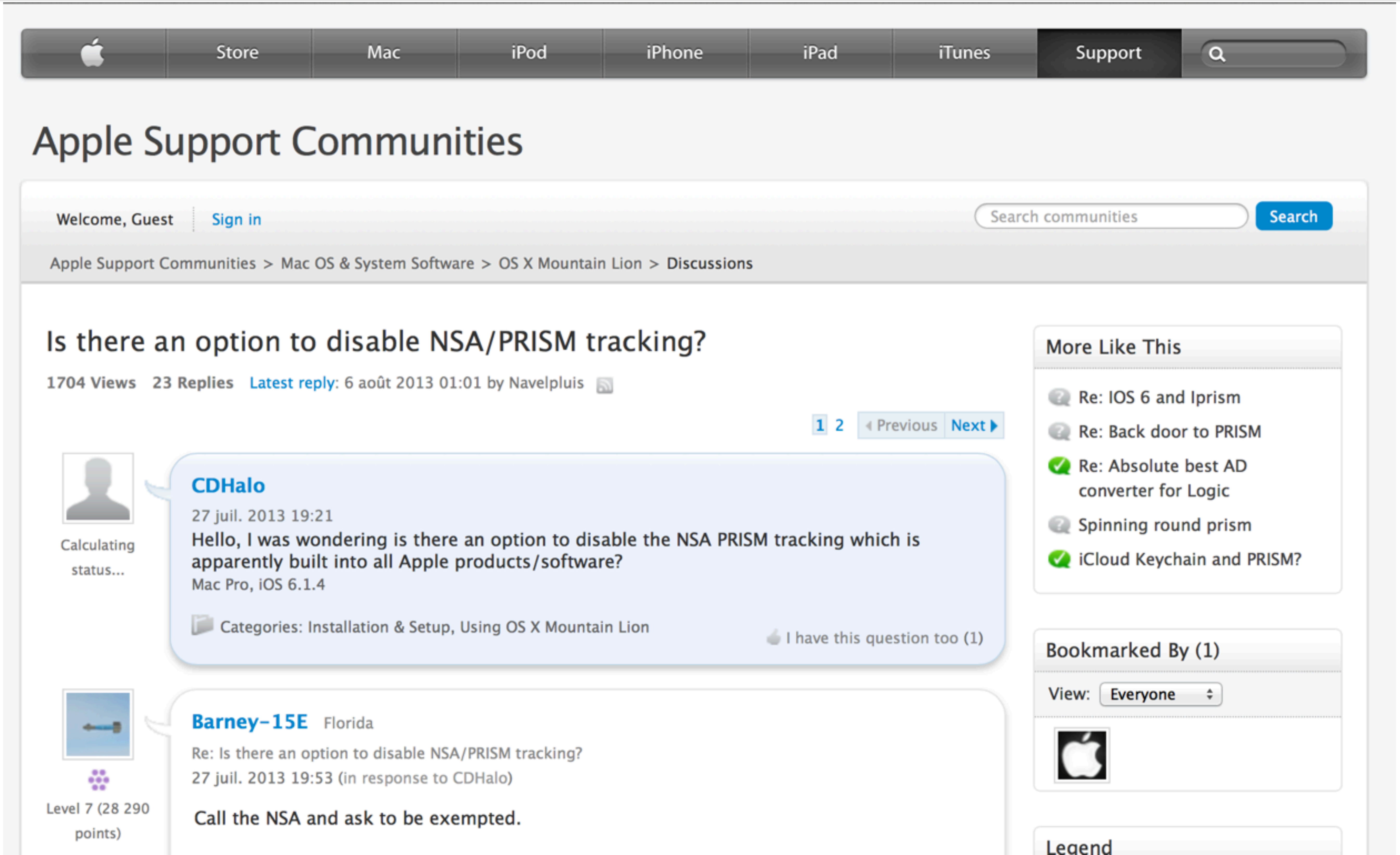
IV. COUNTERMEASURES

Let me alone!

Why is it technically possible?

- Public keys are only cached in the client app memory, and have a life time of only 30 minutes approximately
  - A new iPhone added to an AppleID has to be able to receive iMessages quick
- The lack of certificate pinning adds agencies with strong network capabilities and root CA control to the list of possible spies
- The average user cannot see which public keys are being used by the client app

Simple solution #1 :-)



Apple Support Communities

Welcome, Guest | Sign in

Search communities | Search

Apple Support Communities > Mac OS & System Software > OS X Mountain Lion > Discussions

Is there an option to disable NSA/PRISM tracking?

1704 Views 23 Replies Latest reply: 6 août 2013 01:01 by Navelpluis

1 2 Previous Next

Calculating status...

CDHalo

27 juil. 2013 19:21

Hello, I was wondering is there an option to disable the NSA PRISM tracking which is apparently built into all Apple products/software?

Mac Pro, iOS 6.1.4

Categories: Installation & Setup, Using OS X Mountain Lion

I have this question too (1)

Barney-15E Florida

Re: Is there an option to disable NSA/PRISM tracking?

27 juil. 2013 19:53 (in response to CDHalo)

Call the NSA and ask to be exempted.

Level 7 (28 290 points)

More Like This

- Re: IOS 6 and Iprism
- Re: Back door to PRISM
- ✓ Re: Absolute best AD converter for Logic
- Spinning round prism
- ✓ iCloud Keychain and PRISM?

Bookmarked By (1)

View: Everyone

Legend

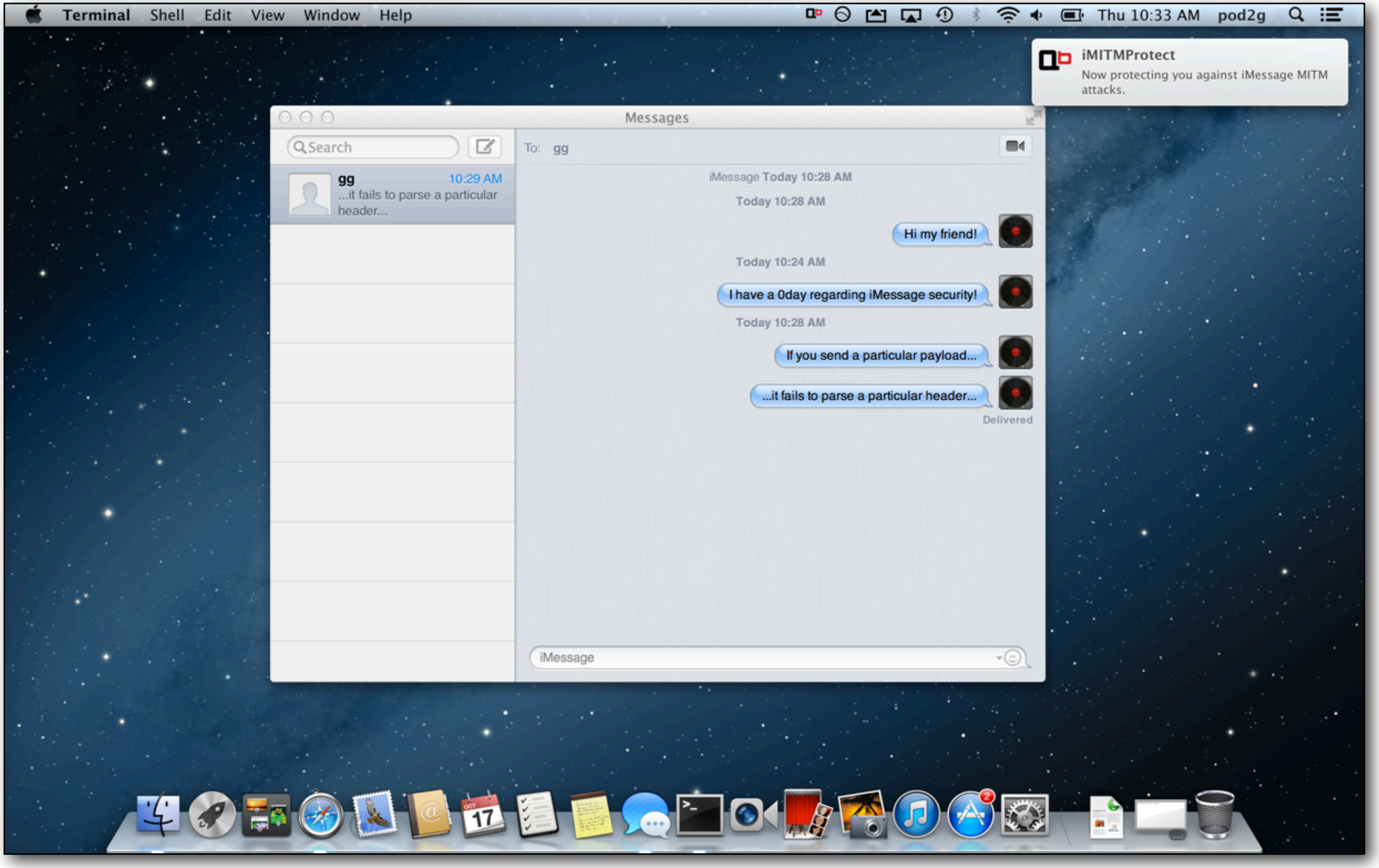
Presenting iMITMProtect - 1

- OS X Version ready
  - Simple installer
- iOS Version on the way
  - Will come as a Cydia package
- Open-source
- http://www.github.com/quarkslab/iMITMProtect

Presenting iMITMProtect - 2

- Hooks *imagent* service
- Contact requests to Apple's ESS servers are recorded to a per-user (OS X) database
- If public keys (RSA / ECDSA) change for a particular token (which should be impossible), a notification is thrown, and the recorded keys are served instead to the client app
- User can list the key database and export them

MitM detection



iMITMProtect
Now protecting you against iMessage MITM attacks.

gg 10:29 AM
...it fails to parse a particular header...

iMessage Today 10:28 AM

Today 10:28 AM

Hi my friend!

Today 10:24 AM

I have a 0day regarding iMessage security!

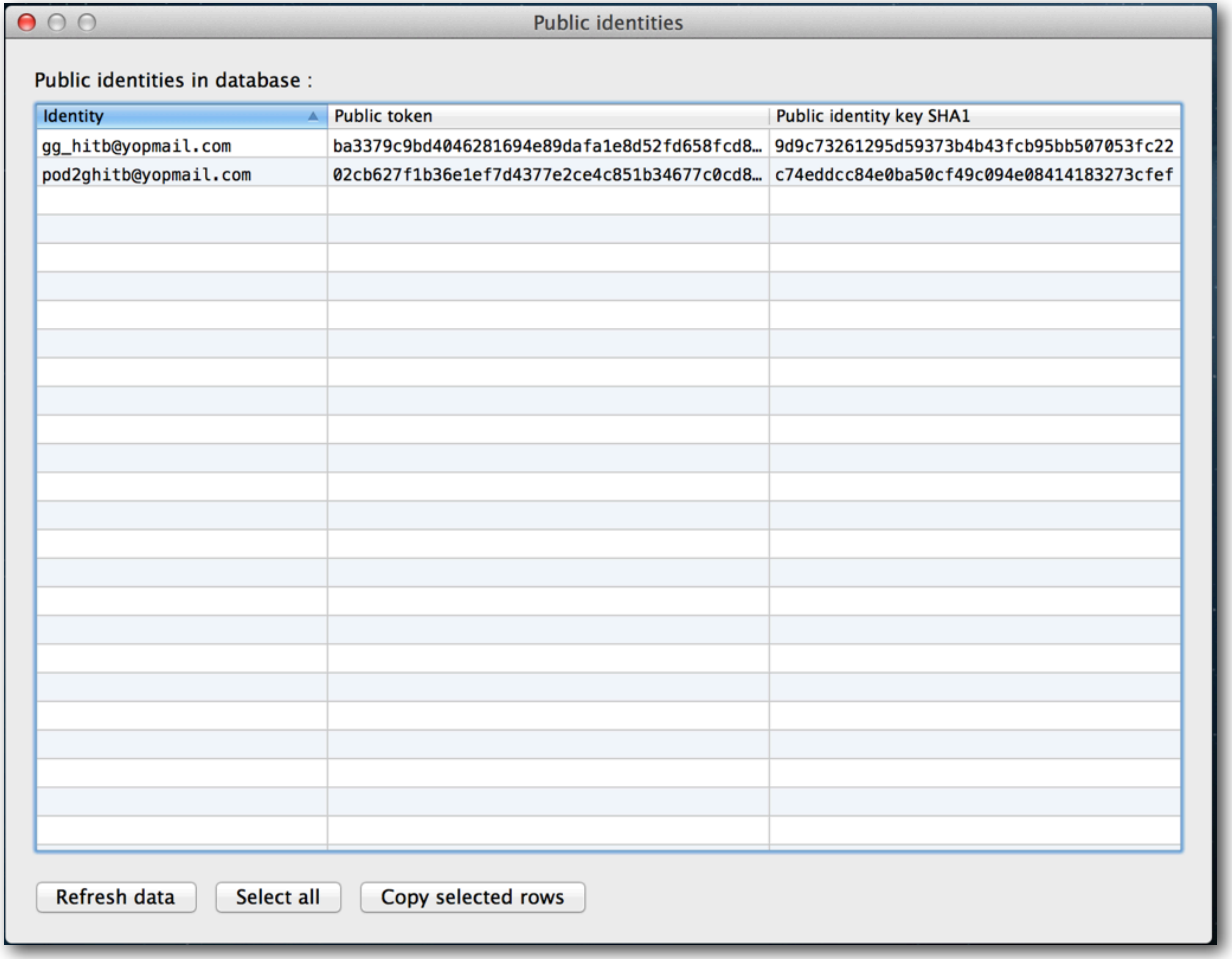
Today 10:28 AM

If you send a particular payload...

...it fails to parse a particular header...

Delivered

Public key database



Public identities in database :

Identity	Public token	Public identity key SHA1
gg_hitb@yopmail.com	ba3379c9bd4046281694e89dafa1e8d52fd658fcd8…	9d9c73261295d59373b4b43fcb95bb507053fc22
pod2ghitb@yopmail.com	02cb627f1b36e1ef7d4377e2ce4c851b34677c0cd8…	c74eddcc84e0ba50cf49c094e08414183273cfef

Features we are working on

- Full database administration: import / add keys, remove specific keys
- Compare rows with a public key directory (where sensible informations would be hashed)
- P2P GPG encryption, with a cleaner PKI

V. CONCLUSION

Final thoughts

QUARKSLAB
INNOVATIVE SECURITY

Now, what do you think?

« Apple has always placed a priority on protecting our customers' personal data, and we don't collect or maintain a mountain of personal details about our customers in the first place. There are certain categories of information which we do not provide to law enforcement or any other group because we choose not to retain it.

For example, conversations which take place over iMessage and FaceTime are protected by end-to-end encryption so no one but the sender and receiver can see or read them. Apple cannot decrypt that data. Similarly, we do not store data related to customers' location, Map searches or Siri requests in any identifiable form. »

Source: https://www.apple.com/apples-commitment-to-customer-privacy/

What is a secure protocol?

- Using strong cryptographic principles
- Implemented in a binary on which absolutely no obfuscation was applied
- Fully documented
- Frequently analyzed by security researchers and crypto-analysts
- With a transparent, and administrable PKI

Shall we continue to use iM?

- MITM attacks on iMessage are unpractical to the average hacker, and the privacy of iMessage is good enough for the average user
- If the informations being exchanged are sensitive to the point that you don't want any government agencies to look into them, don't
- If you are working on Apple 0days, you may also want to avoid this communication channel :-)
- Apple, make a more transparent PKI and document the protocol, and it could be considered the most practical and secure real-time messaging system available

Questions ?

QUARKSLAB
INNOVATIVE SECURITY

www.quarkslab.com

contact@quarkslab.com | @quarkslab.com