GMPLS Network Control Plane Enabling Quantum Encryption in End-to-End Services

Alejandro Aguado, Víctor López, Jesús Martínez-Mateo, Momtchil Peev, Diego López and Vicente Martin
Outline

• Introduction
• Secure Channel Creation
• QKD node architecture
• PCE/GMPLS extensions to enable automatic provisioning
• Experimental validation
• Conclusions
Introduction

• **Quantum key distribution** (QKD) is a novel technology that can be seen as a *synchronized source of symmetric keys* in two separated domains that is immune to any algorithmic cryptanalysis.

• On the other hand, network services are *increasingly requesting more flexibility and network resources*.

• One of the biggest demands is to **increase the level of security** for the transmission between remote premises.

• In this work, we propose a **node architecture** and define **protocol requirements** in a GMPLS environment to provide **QKD-enhanced security in end-to-end services**.
Introduction

Alice → Encrypt → Eve → Encrypt → Bob

Key exchange
Channel Creation
Message encryption
Message Exchange
Introduction: Quantum Key Distribution

Ingredients:
- Qubit transmitter (typically photons), Alice.
- Single qubit receivers, Bob.
- Quantum channel (capable of transmitting qubits from Alice to Bob, in our case fibre).
- Classical channel (public, but authenticated).

Main steps:
- Raw key exchange:
  - Qubit transmission
  - Sifting (basis reconciliation)
- Key post-processing:
  - Information reconciliation
  - Error verification
  - Privacy amplification
Introduction: Quantum Key Distribution

- **QKD** technology can be regarded as **two sources of synchronized random numbers** that are separated physically.
- A correct implementation will deliver keys of the **highest security**
- It can be **mathematically proven to be secure** (in principle, an information theoretic secure (ITS) primitive)

LIMITATIONS

- QKD has some limitations that do not affect the **conventional cryptosystems**, usually based on computational complexity.
- Any kind of **amplifiers or active components** that can modify the state of these signals **must be bypassed**.
- This sets a **limit to the maximum distance** (or absorptions) that a QKD protocol can tolerate, well suited to be used within a metropolitan area or with links of **up to 150 km**
Secure channel creation

- **Key exchange**
  - Exchange Secure Keys / Quantum Channel
  - QKD Box
  - ETSI Proxy

- **Channel Creation**
  - Lightpath creation / Control Plane
  - GMPLS Agent

- **Encryptor**
  - Include Keys in the encryption card
  - Message encryption

- **Message Exchange**
  - OXC

- **PCE**
  - ... 

- **Alice**
  - QKD Box
  - ETSI Proxy

- **Bob**
  - QKD Box
  - ETSI Proxy

- **Eve**
  - QKD Box
  - ETSI Proxy
Example of QKD-enabled network node architecture

Desired capabilities:
• Access to QKD-generated keys.
• Encryption in upstream services (Data encryptor, security module, etc.).
• Switching/Routing.
• Control plane interface enabling automation.
Definition of requirements in terms of parameters

- Parameters required to be exchanged (point-to-point encryption):
  - **Session ID (key_handle):** Initially set as 0, session ID gets the value of the first Key handle extracted by the source agent in the initial setup. The source agent will be in charge of updates (future work).
  - **Key length:** Length of the key to be used for the encryption.
  - **Destination:** It defines the other peer (encryptor/decryptor) to synchronise with. Currently defined by an IP address.
  - **Encryption Layer:** Layer where encryption is performed.
  - **Refresh type and value:** Type of refresh to be done for a key (time/traffic/etc) and the value to be considered as a threshold.
  - **Algorithm:** Encryption algorithm to be used.
Distributed GMPLS Control

• Majority of the commercial deployments of optical core and transport networks are based on GMPLS.
• GMPLS was standardized by IETF in CCAMP WG
• Fundamental protocols:
  • RSVP-TE: responsible of setting up end-to-end quality-enabled connections
  • OSPF-TE: dissemination of the topology and traffic engineering (TE) information, enabling routing
  • LMP (Link Management Protocol): is responsible of links management
Path Computation Element

• GMPLS is complemented with a logically centralized element, the PCE

- PCE learns the TE DB listening the IGP.
- Active Stateful PCE can request to create a path using PCInitiate.
- The node set-up the connection using RSVP Path, Resv.

Telefonica Netphony release open source PCE implementation and GMPLS control plane.
GMPLS+PCE Architecture

Proposed workflow: Case “Node starts”

4 metrics:
- Key length
- Layer of encryption
- Refresh type / value
- Enc_Alg

GMPLS case:
- PCRequest including metric for inline encryption.
- PCReply including new ERO subobjects for key management
- RSVP including the same ERO
- RSVP QE ERO subobject detected by node 1. Key_handle unset (=0), it gets a new key and key_handle, and adds the key_handle as sessionID to be used by node 5
- Node 5 gets the sessionID and extracts the required key.
- The rest is standard RSVP
Experimental validation

DockerNet

Node:
etsiA

Type: LC
Img: ubuntu:14.04
ext 10.2.2.11
inter 11.2.2.1
sync 11.1.1.11

https://github.com/alexaguado/DockerNet
Experimental validation
OSPF for Quantum encryption capabilities

Quantum Encryption support (bit 7): capable
Experimental validation PCEP

10.1.1.1 10.1.1.200 PCEP 160 Path Computation Request (PCReq)
10.1.1.200 10.1.1.1 PCEP 268 Path Computation Reply (PCRep)
10.1.1.1 10.1.1.200 PCEP 272 Path Computation LSP State Report (PCRpt)

Path Computation Element communication Protocol
  ▶ Path Computation Request (PCReq) Header
  ▶ RP object
  ▶ END-POINT object
  ▶ BANDWIDTH object
  ▶ METRIC object
  ▶ METRIC object
  ▶ METRIC object
  ▶ OBJECTIVE FUNCTION object (OF)

Path Computation Element communication Protocol
  ▶ Path Computation Reply (PCRep) Header
  ▶ RP object
  ▶ BANDWIDTH object
  ▶ METRIC object
  ▶ METRIC object
  ▶ OBJECTIVE FUNCTION object (OF)

New QE
ERO subobject
Experimental validation RSVP (signalling)

Node 4 QE ERO subobject. (before node 2)
Type: 0x67
Value: "00..00" (64 bytes)
KeyLength: 32
Enc_layer: 2
RefType: 0xfd
RefValue: 60
Alg: 10 (TBD)

Node 4 QE ERO subobject. (before node 2)
Type: 0x67
Value: "4a0e...052f" (64 bytes)
KeyLength: 32
Enc_layer: 2
RefType: 0xfd
RefValue: 60
Alg: 10 (TBD)
Conclusions

• We propose a node architecture and define protocol requirements in a GMPLS environment to provide QKD-enhanced security in end-to-end services.

• This is the first work to propose, implement and validate extensions in a PCE/GMPLS architecture to use this technology.

• The work is done with Open Source tools using Netphony and DockerNet.

• As future work, the authors will explore this approach in OpenFlow or Netconf.
THANK YOU!!!

Alejandro Aguado, Víctor López, Jesús Martínez-Mateo, Momtchil Peev, Diego López and Vicente Martin
Appendix A
ETSI GS QKD 004 V1.1.1
for remote apps and IDQ3P
ETSI IDQ Proxy

• ETSI GS QKD 004 V1.1.1 defines an API to be used by applications which are running within the same server as the Key Manager.
• In order to justify the use of this standard, we have developed a proxy that implements ETSI GS QKD 004 V1.1.1-based messages to communicate with external applications.
• These messages are mapped to IDQ3P requests.
• Additional Sync messages have been implemented as well.
• This interface allows to use a single identifier (key_handle) that can be used to extract multiple keys.
Modules / Messages

ALICE

APP

ETSI / IDQ Proxy

IDQ System

BOB

APP

ETSI / IDQ Proxy

IDQ System

Sync messages:
Session Opened/closed
Block Session, Update Key

Quantum channel
Error correction
Distillation...

App messages
Send_key_handle()

ETSI GS QKD 004 V1.1.1 msgs
QKD_{OPEN, CLOSE, GET_KEY, CONNECT_NONBLOCK, CONNECT_BLOCKING}

ETSI GS QKD 004 V1.1.1 msgs
QKD_{OPEN, CLOSE, GET_KEY, CONNECT_NONBLOCK, CONNECT_BLOCKING}
Example OPEN & CONNECT

ALICE

IDQ System

ETSI / IDQ Proxy

APP

Key_handle

Send_key_handle()

QKD_CONNECT_NONBLOCK()

ACK

BOB

ETSI / IDQ Proxy

APP

QKD_OPEN(Key_handle)

ACK

QKD_CONNECT_NONBLOCK()

ACK

SYNC_OPEN(key_handle)

QKD_OPEN()
Example GET_KEY

GET_KEY()
KeyID, Key

SYNC_KEY(key_ids)

ACK

Update_Key()

ACK

QKD_GET_KEY(key_handle)

KEY

QKD_GET_KEY(Key_handle)

SYNC_BLOCK(key_handle)
Example CLOSE

- ALICE
  - IDQ System
  - ETSI / IDQ Proxy
  - APP
  - QKD_CLOSE()
  - SYNC_CLOSE(key_handle)
  - ACK
  - Send_close()????
  - ACK
  - QKD_CLOSE(Key_handle)
  - ACK

- BOB
  - ETSI / IDQ Proxy
  - APP
  - ACK