# A QKD protocol and its integration into a software library

Thomas Prévost - Bruno Martin - Olivier Alibart

# Agenda

- Brief context introduction
- QKD: advantages and challenges (from IT PoV)
- Protocol design for unlimited distance and participants
- Formal proofs
- Randomness and security *(with Yoann PELET)*
- Implementing a user-friendly software library
- Conclusion: a video call secured by QKD

# Context and personal introduction

- Thomas Prévost, IT engineer from Polytech


- PhD thesis, supervised by Bruno Martin (I3S)

- Co-supervised by Olivier Alibart

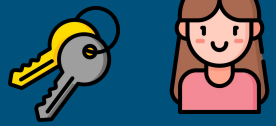
# QKD: advantages (from IT PoV)

How to transmit a secret to someone I never met before ?

We should encrypt messages…

But how to transmit the encryption key between participants?

# Current solution: public key cryptography



*Public and private key*



Public key

*Generates random symmetric key*

Symmetric key encrypted using public key

*Decrypts symmetric key using private key*

*Share a common symetric key*

# Public key drawbacks

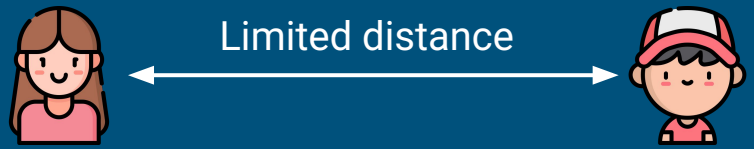
- Symmetric key size limitation
- Potentially vulnerable to future attacks (for example quantum algorithms)


“Harvest now, decrypt later” attack

# QKD advantages

- Unlimited key size
- Perfect forward-secrecy: encryption is **broken now or never**

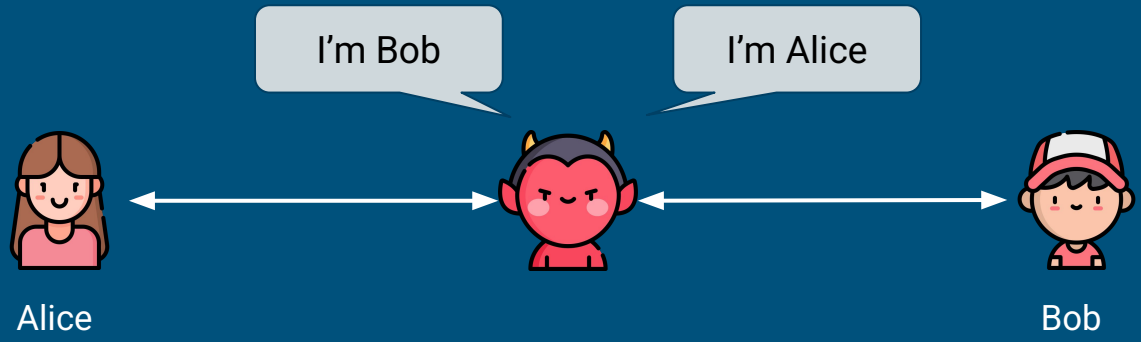
# Main QKD challenges




Very expensive: mostly suited for cross-datacenter communication

# The authentication problem

QKD remains vulnerable to Man-In-The-Middle (MITM) attack



# How can you be sure of someone’s identity?

# Solution: use information you already know

# Solution: use authentication authority

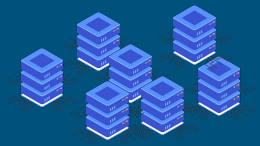
# Authentication

- There is no perfect authentication process as it exists for encryption
- You must adapt your method regarding your constraints
- Protocol are designed using some assumptions, these must be chosen properly

# Protocol design: constraints

- Routing the secret through nodes in case of no direct QKD link
- From 2 to n x n users on the same channel

?



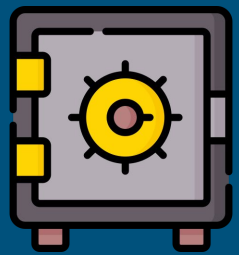
?

# Introducing a new primitive: shared secrets

Employees

There must be at least 2 over 3 employees to open the safe

Manager

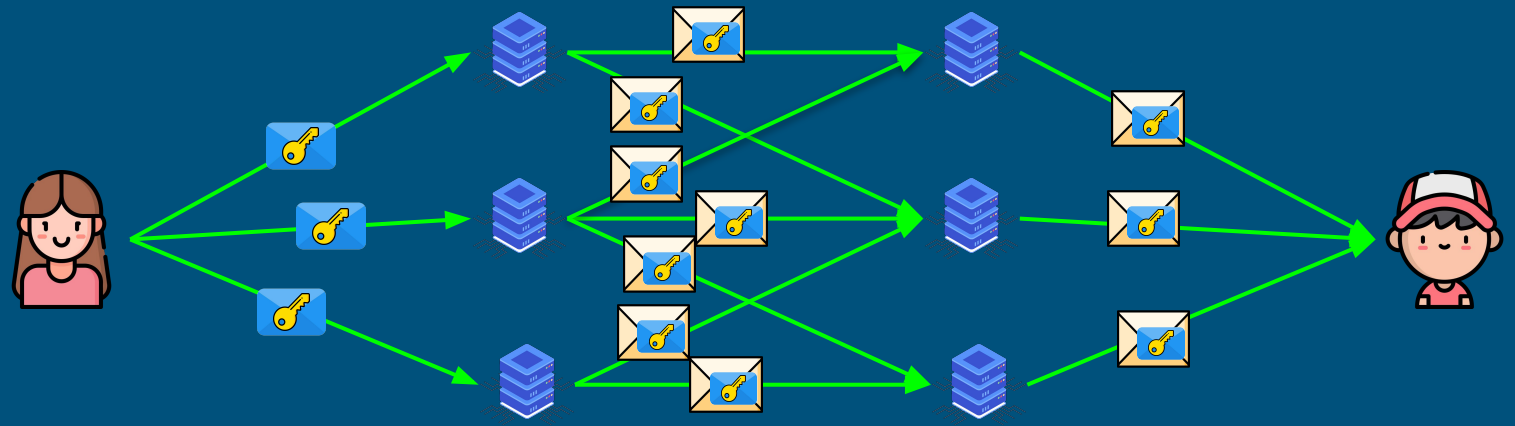


# Introducing a new primitive: shared secrets

Shamir's Shared Secret Scheme:
Let *n* be the number of share, *k* the decrypting threshold
Let *P* be a random polynom, *deg(P) = k-1 (a + b.x + c.x^2 …)*
*secret = P(0)*
*shares = {P(1), P(2), …, P(n)}*

It is possible to reconstruct secret *P(0)* from **k** shares using **Lagrangian interpolation**

# Our protocol: recursive secret sharing between intermediate nodes



QKD independent secured channel

Key / secret

Shamir share, threshold = 51%

# How to be sure that my protocol is flawless?

**The public key is never authenticated, a Man-In-The-Middle attack is possible!!!**

*Public and private key*

Public key

Symmetric key encrypted using public key

*Generates random symmetric key*

*Decrypts symmetric key using private key*

*Share a common symetric key*

# Solution: formal security provers

- Takes abstract description of cryptographic primitives and protocol:

  *fun senc(bitstring, key): bitstring.*

  *reduc forall m: bitstring, k: key; sdec(senc(m, k), k) = m.*

  *process*

  *out(public_channel, senc(my_secret));*

- Translates protocol into constraints set

# Solution: formal security provers

- Tries to infer all possible attacks over the protocol
- Possible replies:
  - unsafe (with the attack)
  - cannot be proven
  - safe
- Soundness: The prover cannot reply “safe” if an attack exists
- So we are **100% certain that our protocol is secure**!

# Randomness validation of symmetric key

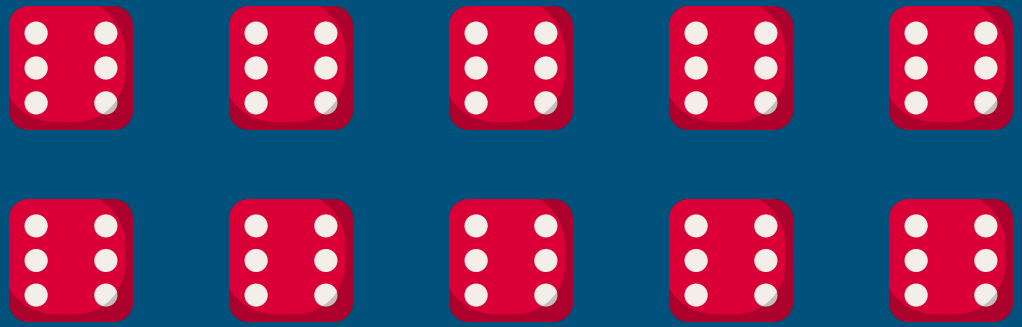
2 notions are hidden behind “randomness”:

- Initial source of entropy, should be unknown from the attacker (quantum entropy source is well suited).

- Final distribution, must “look like” random. This is what we are testing.

# What is random? Let’s play with dice

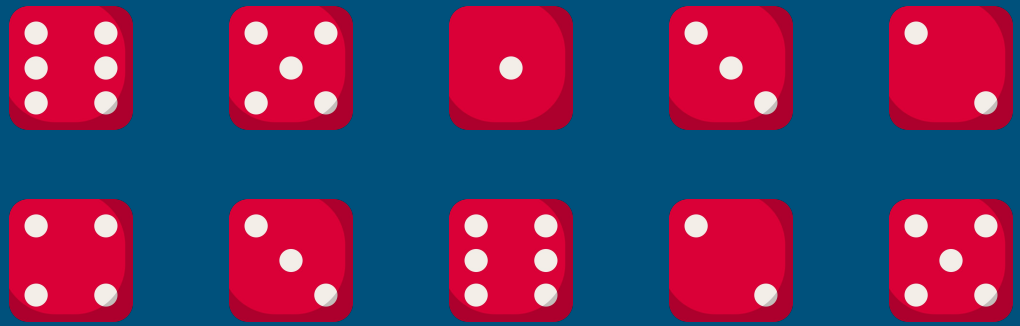
Let’s throw 10 dice:



*Would you trust me if I told you my dice were fair?*

# What is random? Let’s play with dice

Restart the experiment



*And now would you consider my dice random?*

**WHY?**

# Randomness distribution validation

- There is no way to prove that an output distribution satisfies randomness requirements with 100% certainty
- What we can do is “statistical tests” over a large range of data, and verify that the output bits “look random”
- Some tests are better than other. We used:
  - dieharder
  - NIST
  - testU01

# Randomness validation of QKD output bits

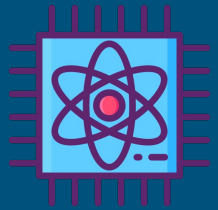
Does QKD generator validate statistical tests?

No

So how could we extract cryptographic keys?

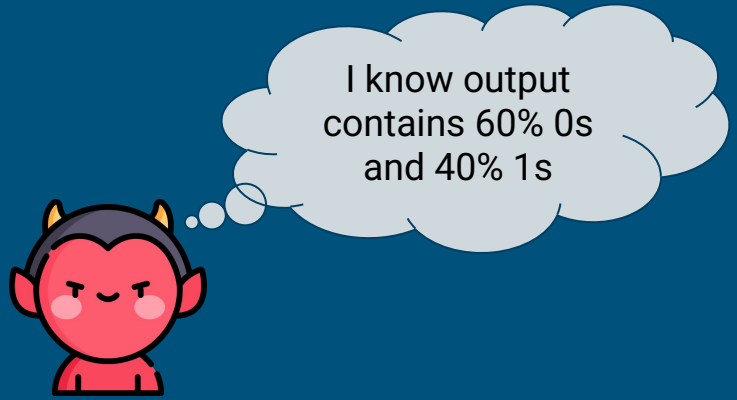
*Thanks to Yoann PELET*

# Privacy Amplification and min-entropy



Quantum generator

n bits =
0 1 0 0 0 1 1 0 1 0




- **Privacy Amplification** is a deterministic algorithm that extract uniform distribution from output bits. It is run by both participants after QKD is finished.
- It can extract m < n random bits, due to attacker biasis knowledge. This is called **min-entropy**.

# Randomness validation of PA output bits

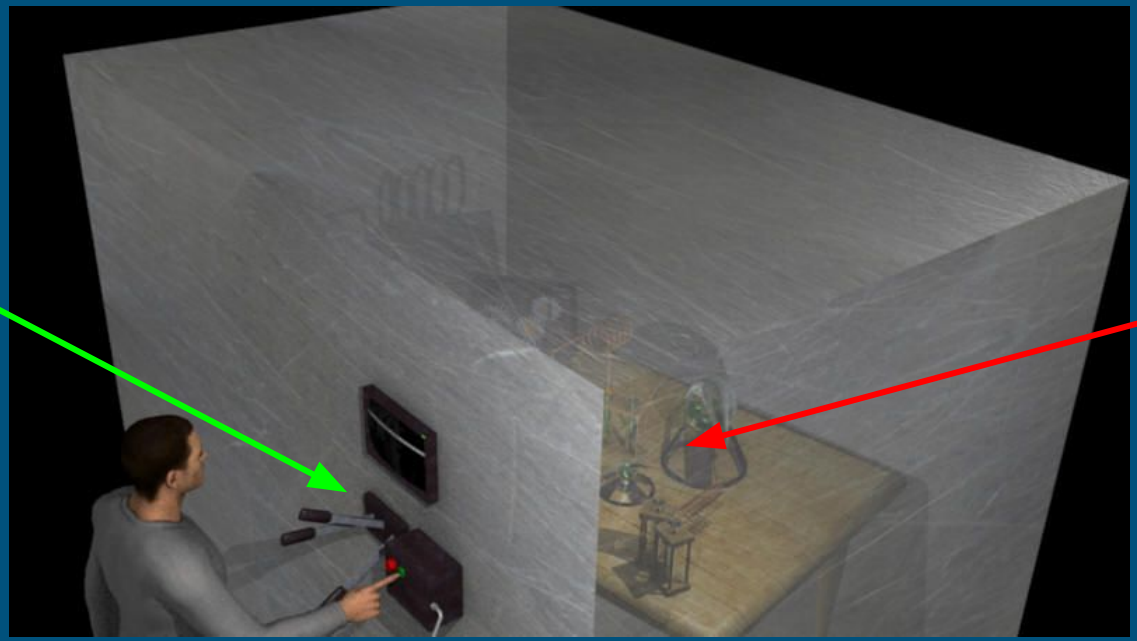
Does Privacy Amplification generator validate statistical tests?

Yes

We can use the PA output bits as cryptographic keys

# Implementing a user-friendly software library

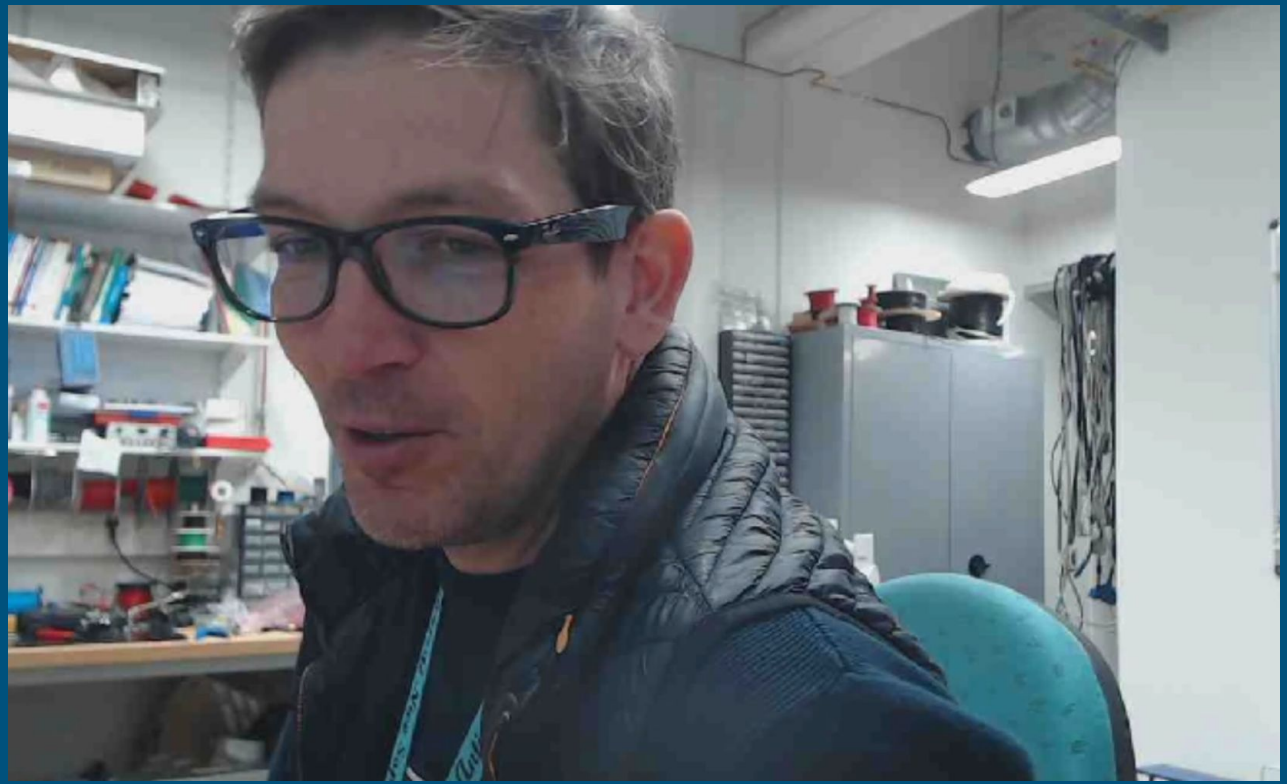
A good encapsulation



# Implementing a user-friendly software library

- Layer over SSL/TLS (HTTPS)
- Backwards compatibility with classic HTTPS
- Followed ETSI GS QKD 014 v1.1.1
- Target: RFC (Request For Comments), ie Internet standard

# Conclusion: a video call secured by QKD



# Thanks

Do you have questions?