QKD, qCrypto and LDPC

01
Quantum Key Distribution

A description

02
Open Source Solutions

03
qCrypto

SpeQtral's software stack

04
LDPC

Protocol procedure & simulation results

05
What's next for qCrypto

Taking qCrypto forward

Basics of Encryption

Alice

Bob

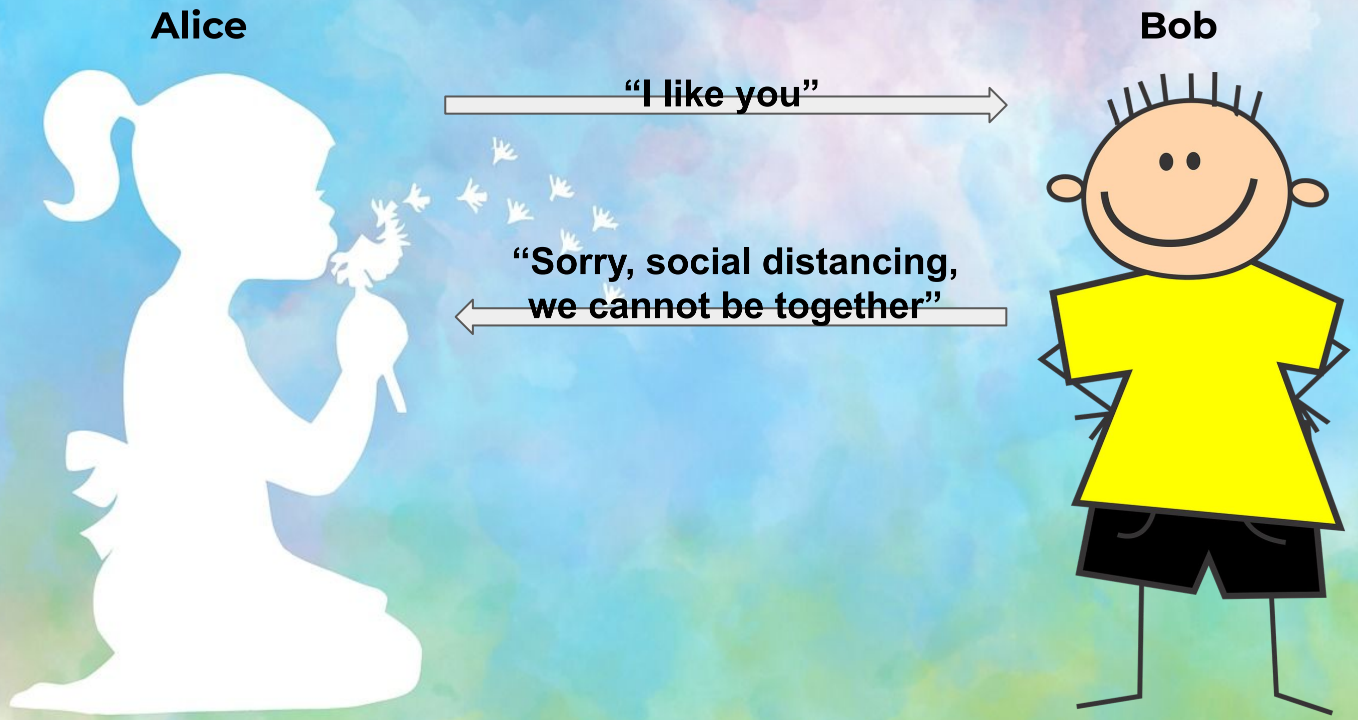
QKD

Why?

QKD

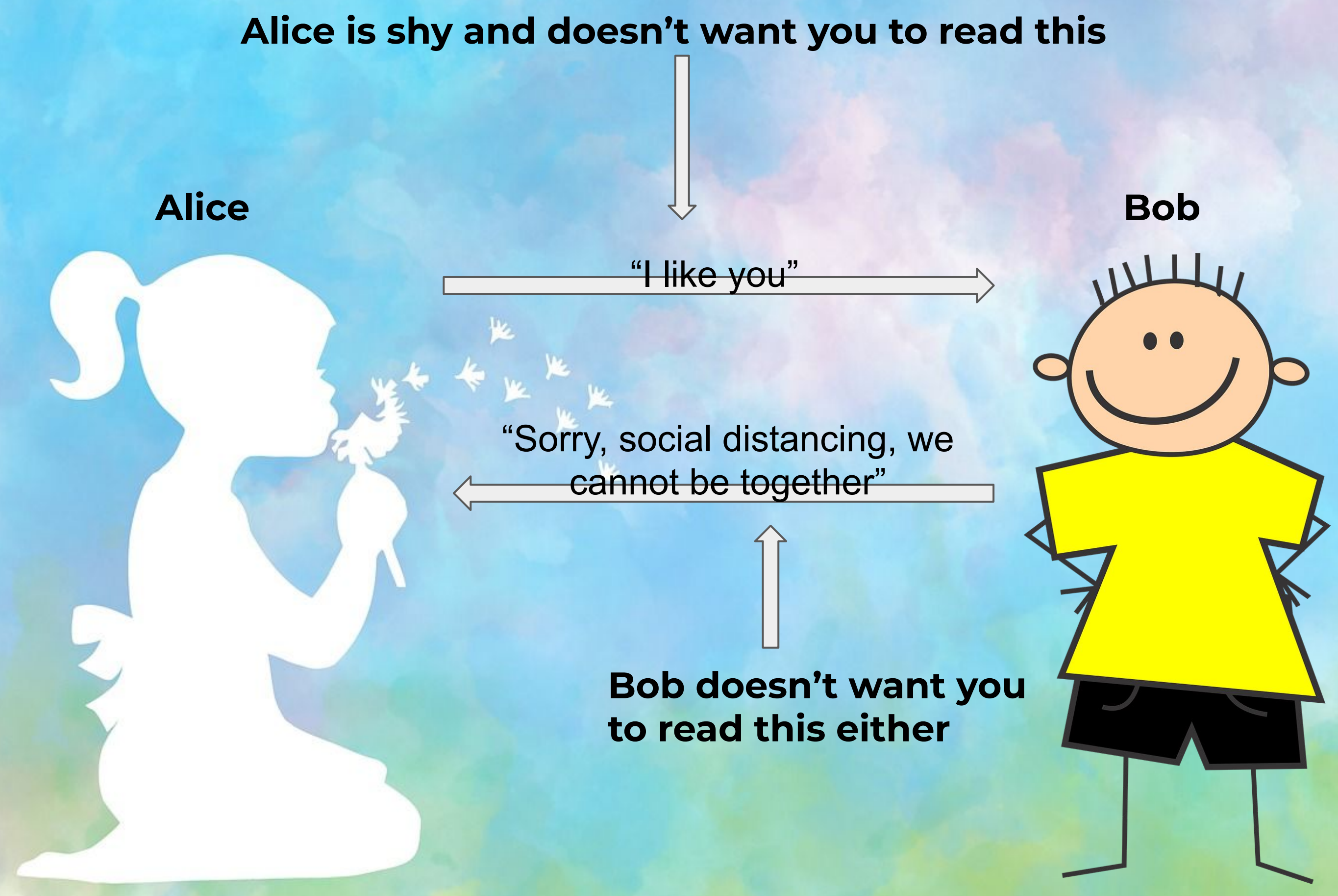
Why?

Alice & Bob want to talk.

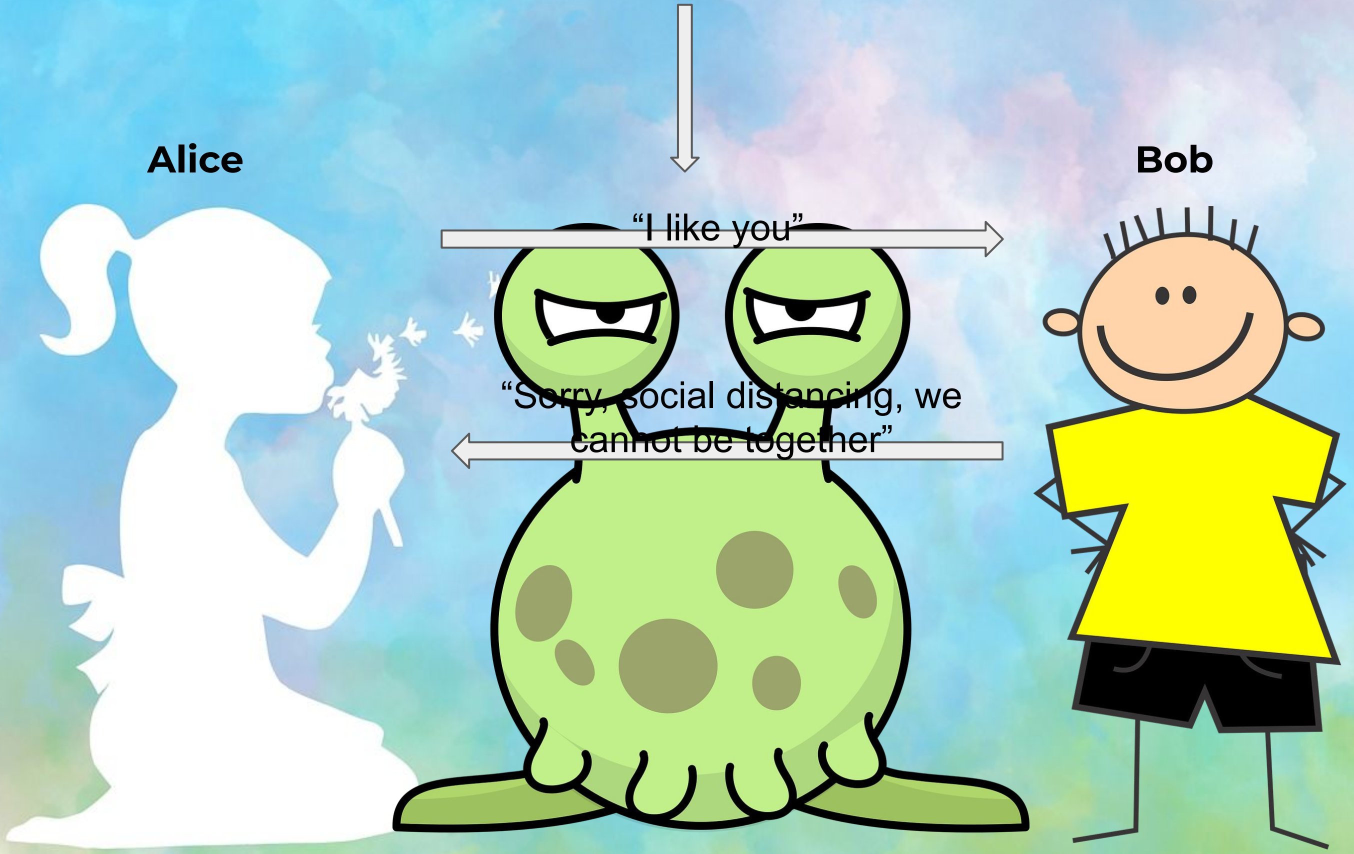


QKD

Why?


Alice is shy and doesn't want you to read this
Alice
Bob
"I like you"
"Sorry, social distancing, we cannot be together"
Bob doesn't want you to read this either

And there may be people who *really* want to know what they are talking about... We call these people Eve

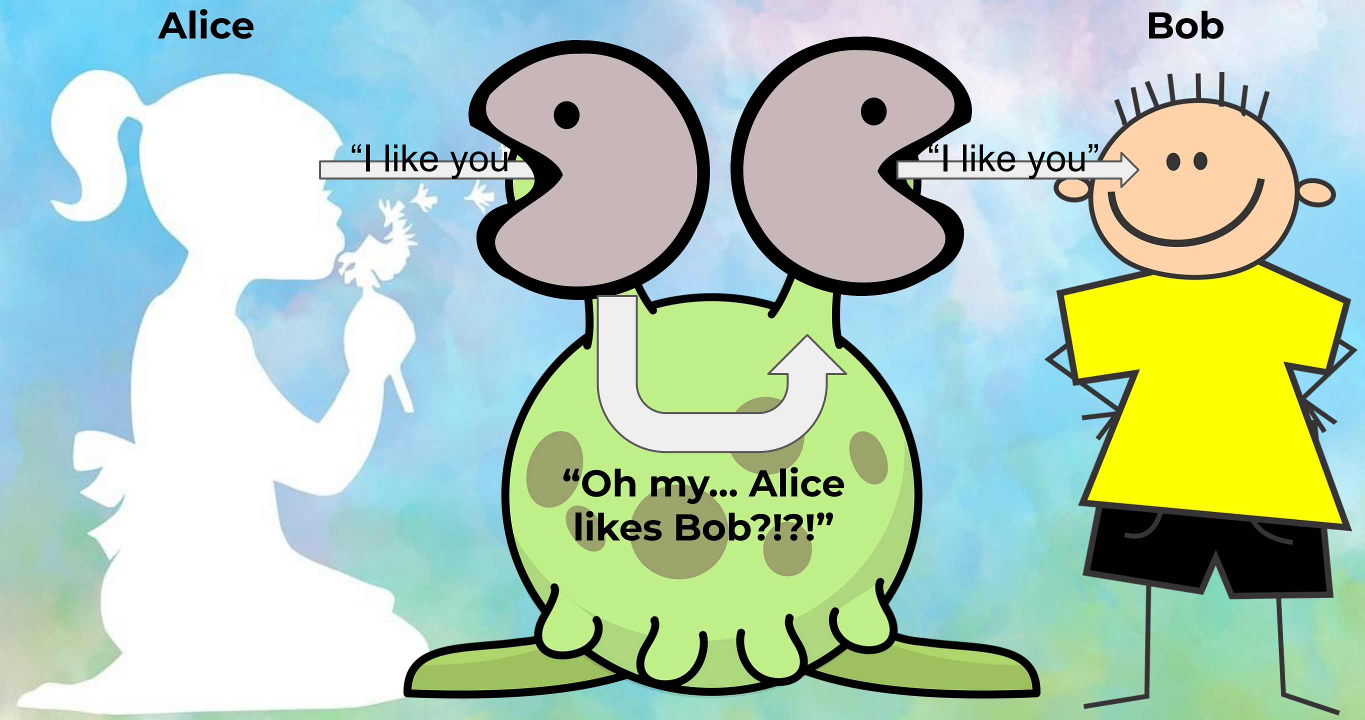


QKD

Why?

Eve eavesdrops on their communications by usually by listening,

By “listening” we really mean “listening” and then “forwarding”.



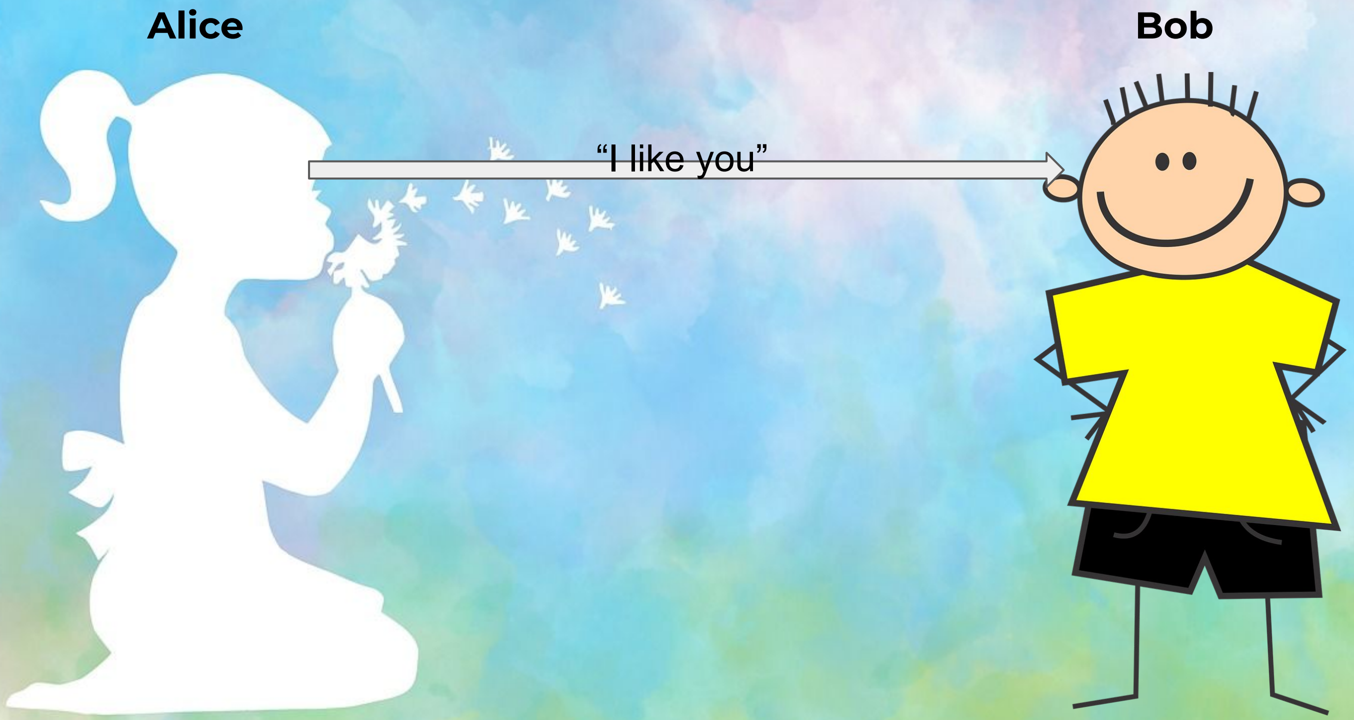
QKD

Why?

QKD

Why?

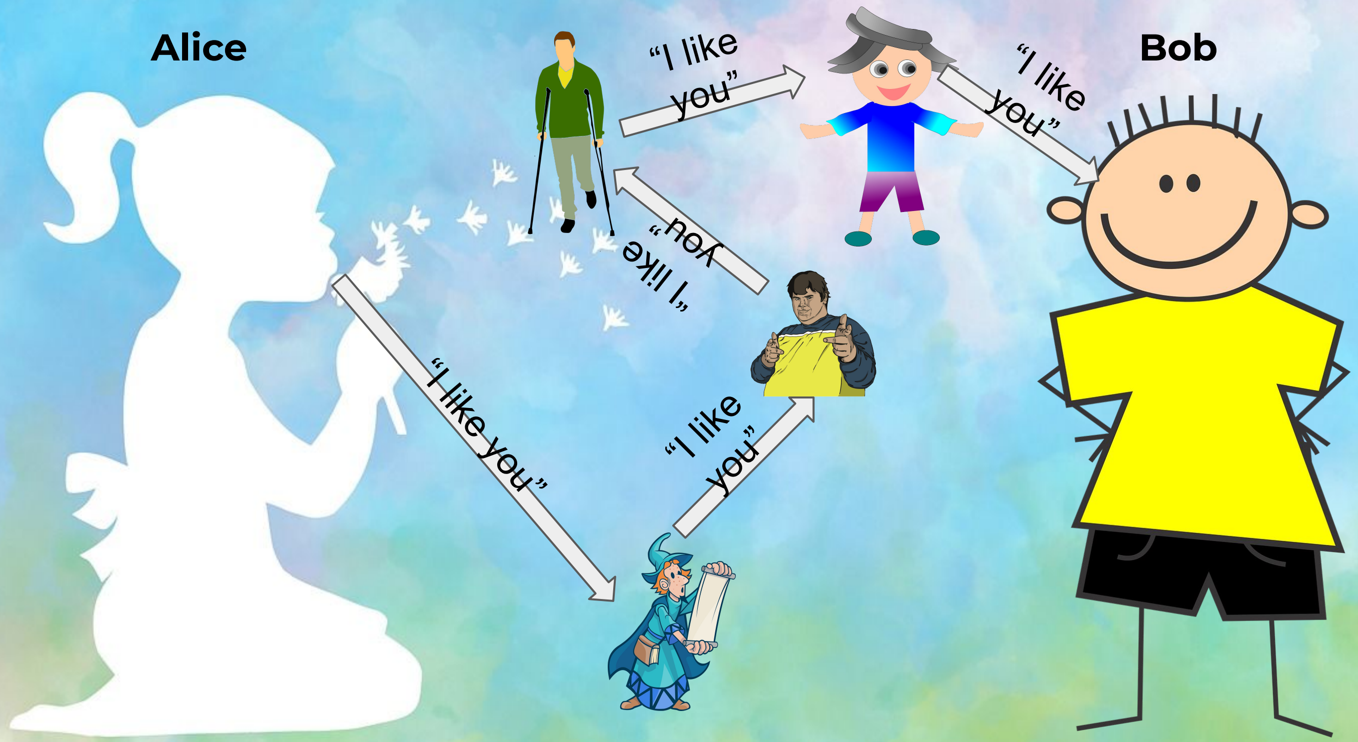
Alice and Bob don’t know that Eve exists…



QKD

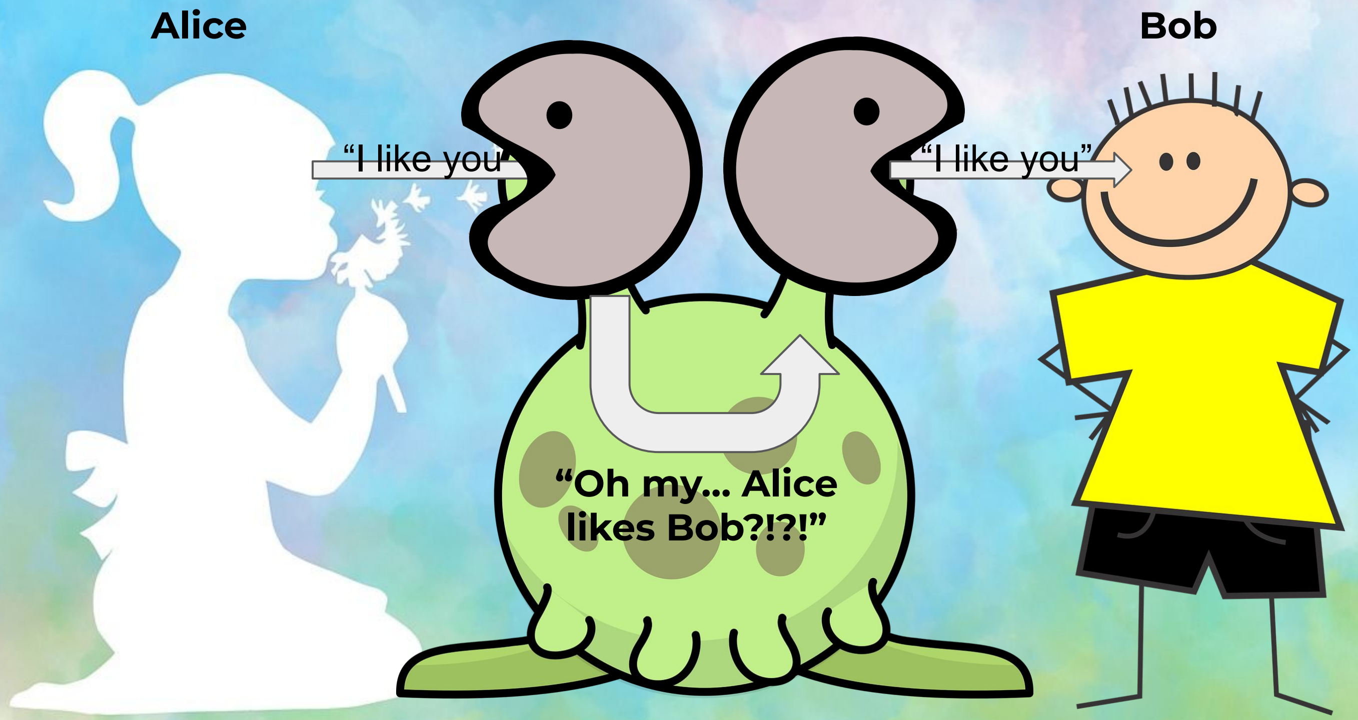
Why?

Because the internet is a huge network.
(Alice doesn't have a direct connection to Bob)



And even if they have a direct connection, Eve can just cut the wire and sit in the middle and they won't know.

(all they see is the input & output)



QKD

Why?

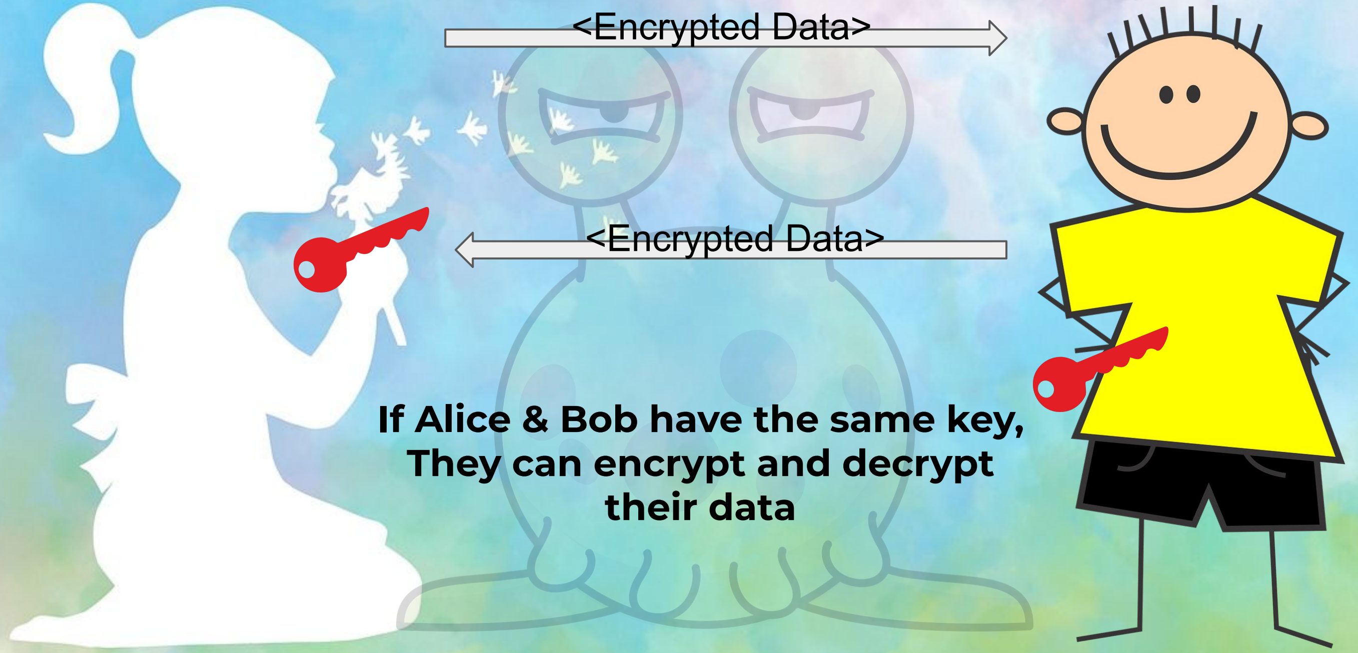
QKD

Why?

So how security is usually done is to assume Eve is always there, and just encrypt the data

Now even if Eve can “eat” and “spit” the data, she can’t understand it

Alice **Bob**



If Alice & Bob have the same key, They can encrypt and decrypt their data

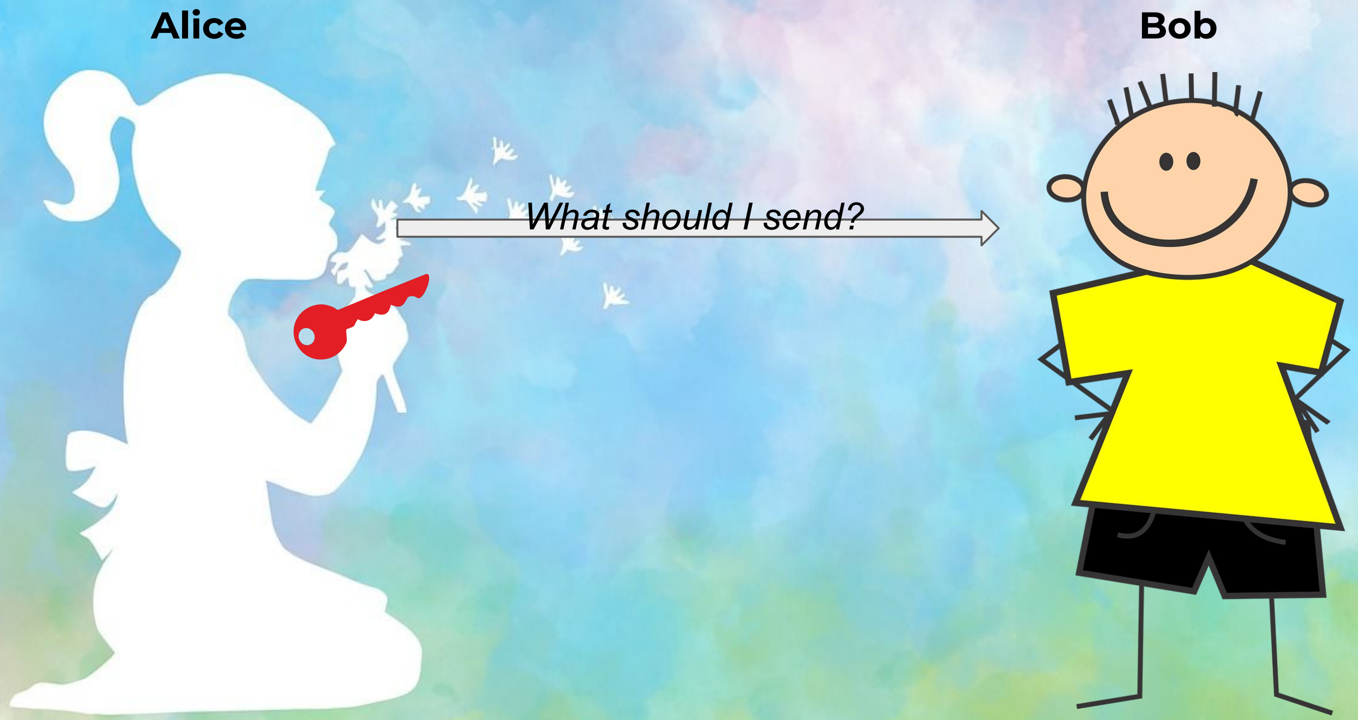
Here's the catch… How to make sure both of them have the same key?

QKD

Why?

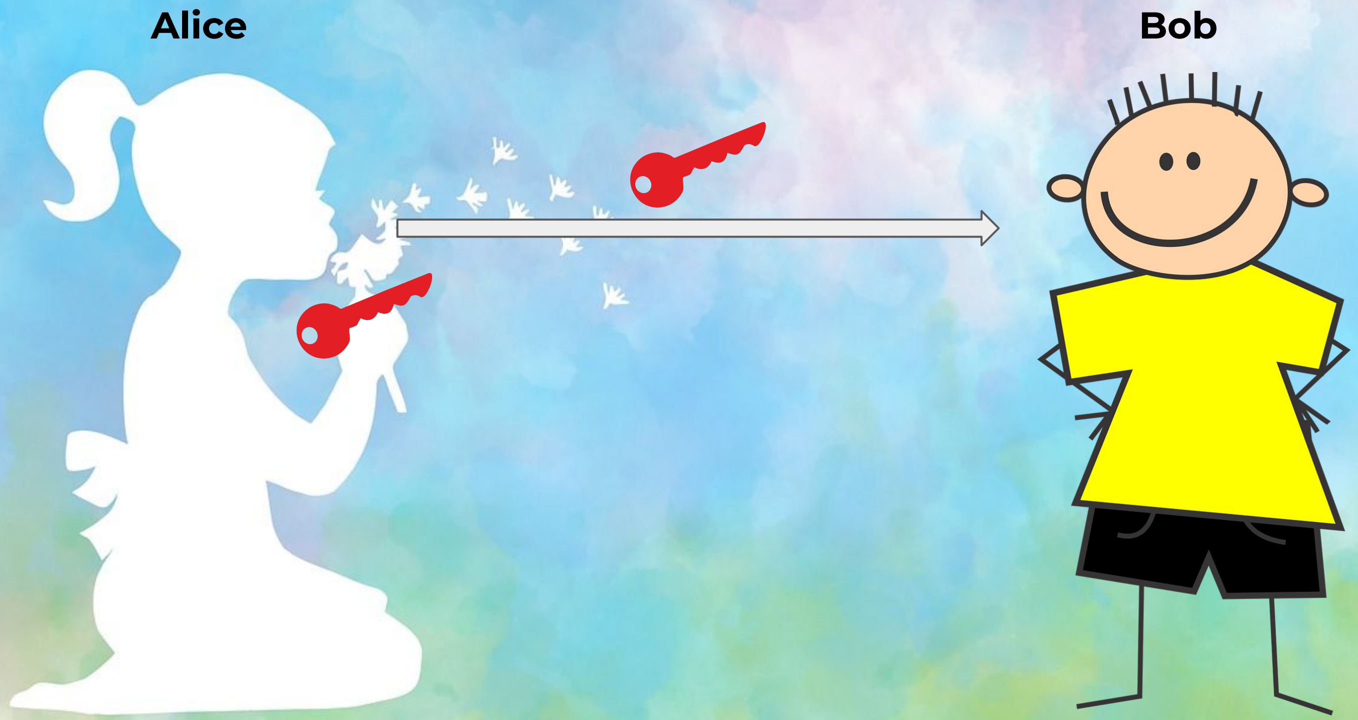
QKD

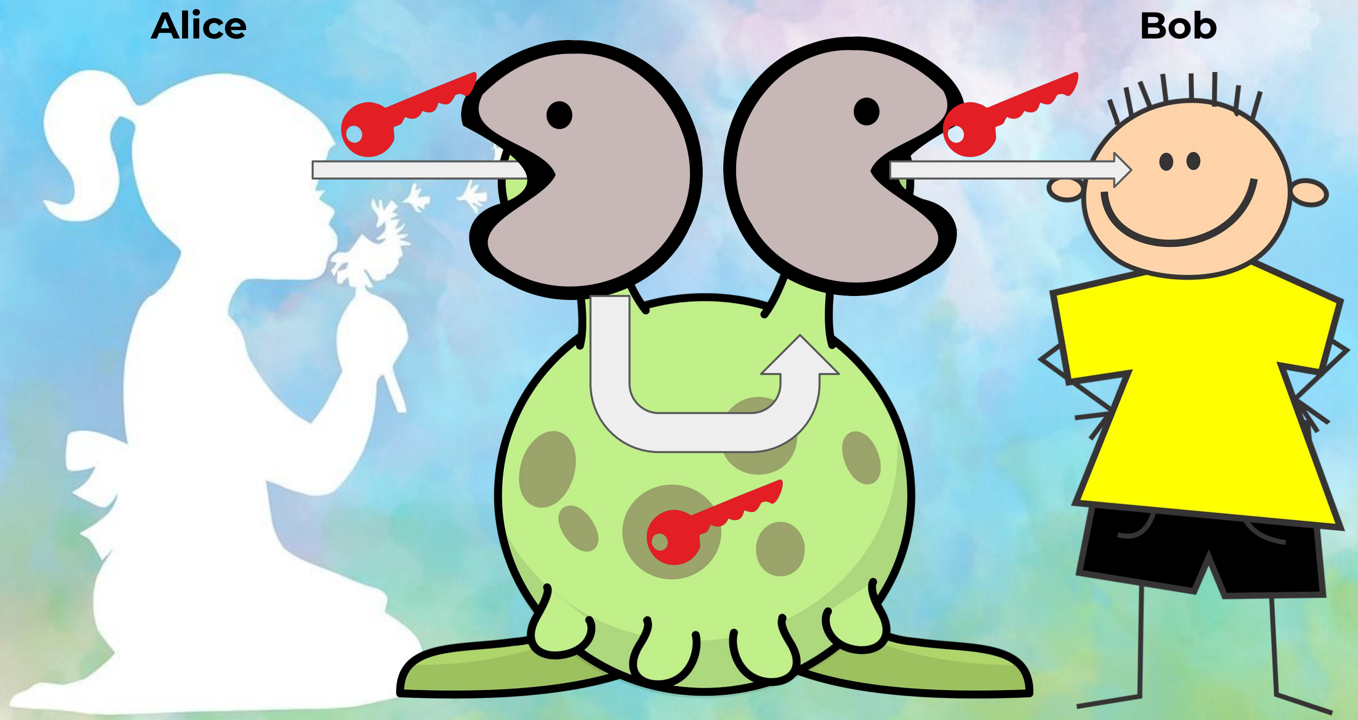
Why?



QKD

Why?





QKD

Why?

QKD

Why?

There are non-quantum ways to do key distribution (e.g. asymmetric key), but that's not what this presentation is about.

Let's talk about quantum key distribution (QKD)

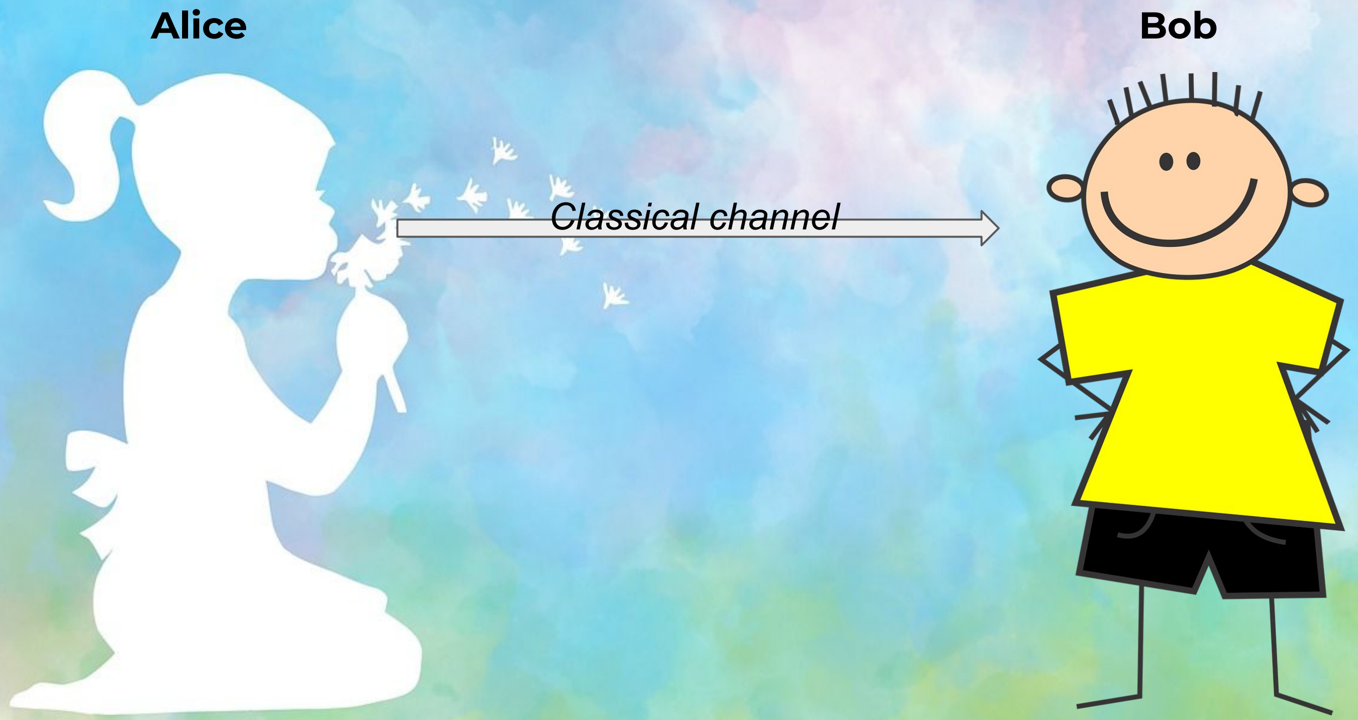
Alice

Bob

QKD

Why?

The classical channel does not provide any info on whether your message was intercepted



But the quantum channel is *fundamentally* different.

"Quantum physics says that measurement in general modifies the state of the measured system" (Scarani et al., 2009, p3)

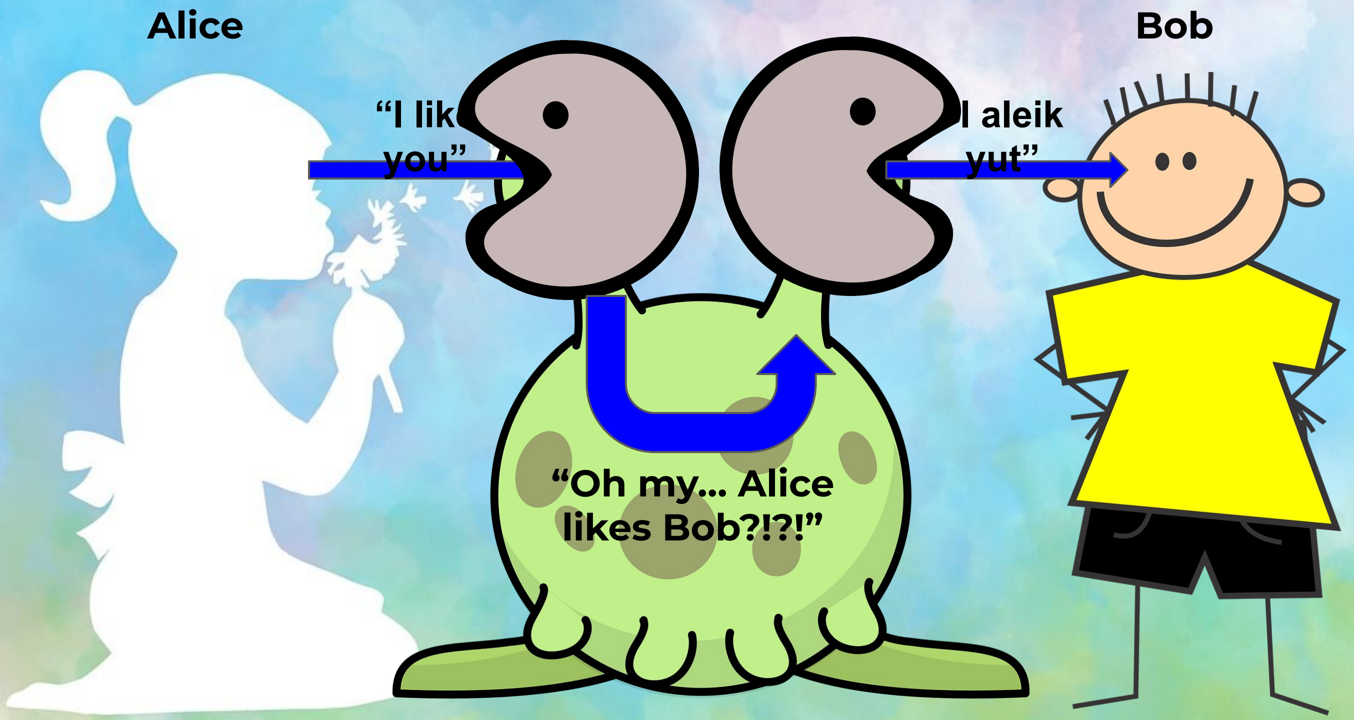
This is due to the no-cloning theorem which states that it is impossible to create an identical copy of an arbitrary unknown quantum state.

QKD

Why?

By reading (measuring) the data from Alice on a quantum channel, Eve *will* introduce errors into the message which can be detected by Bob.

This is known as an **intercept-resend attack**



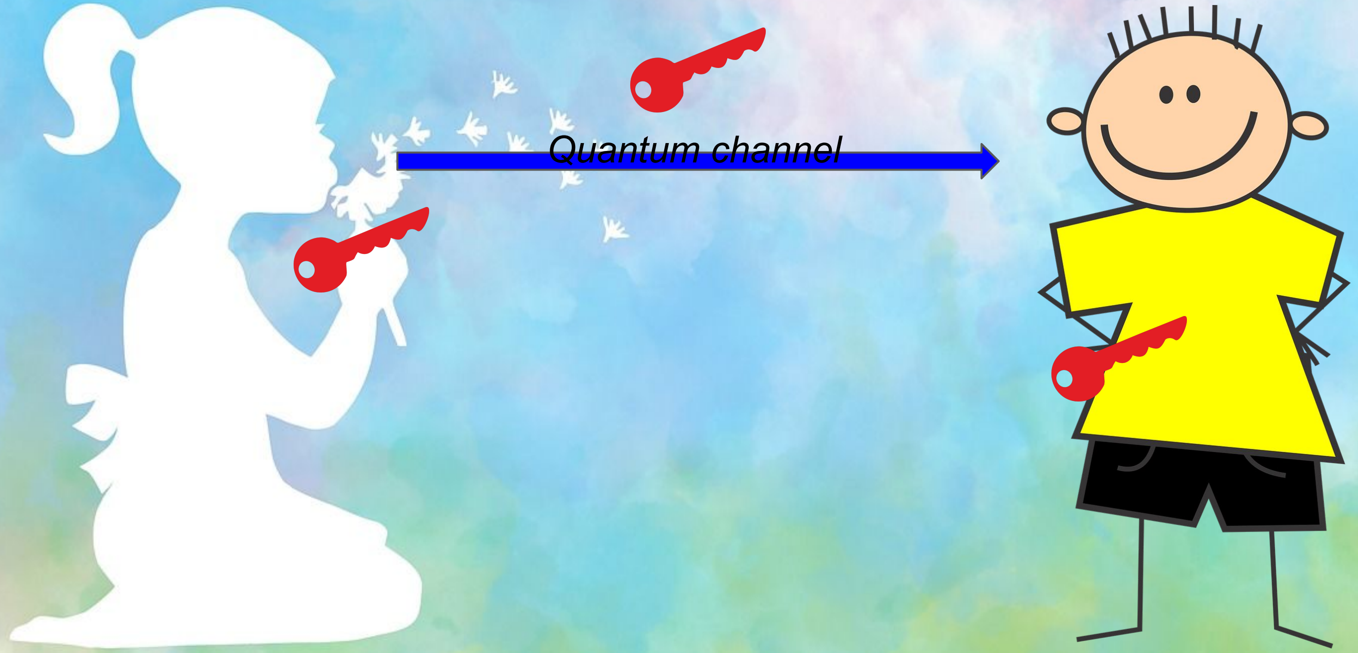
QKD

Why?

Since Eve's knowledge of whatever is sent on the quantum channel can be calculated (by the error rate),

we can send the key on the quantum channel,

Then verify that it has has not been read by checking the number of errors in the key. (Quantum Bit Error Rate i.e. QBER)



QKD

Why?

So how does QKD work?

4 Phases:

1. Detection (Alice sends Bob the key, Bob "detects" it)
2. Sifting (Bob "sifts through" the basis of the key bits)
3. Error Correction (Alice & Bob correct errors in the key)
4. Privacy Amplification (Alice & Bob make their key more "private" by making it harder for Eve to know what it is)

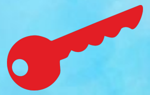
QKD

Detection

QKD

Detection

The key is actually a random sequence of ‘0’ and ‘1’



= “01011010101000101…”

Each ‘0’ or ‘1’ is known as a **bit**

Alice sends the key bit by bit. Simple, right?

If only it were that simple. We're talking about quantum physics!

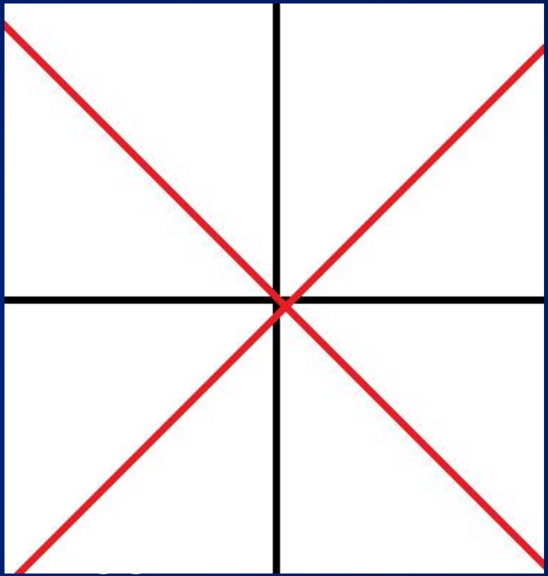
Alice converts each bit into a qubit, then sends the qubit, and Bob measures these qubits.

QKD

Why?

Alice Qubit sender (Photon source)

- Alice chooses between 2 basis randomly
- Based on her i^{th} bit's value, she represents the bit as a straight line
- **Basis 1: Plus**
- 0 = Vertical line
- 1 = Horizontal line
- **Basis 2: Cross**
- 0 = Diagonal \ line
- 1 = Antidiagonal / line



Basis + in black

Basis x in red

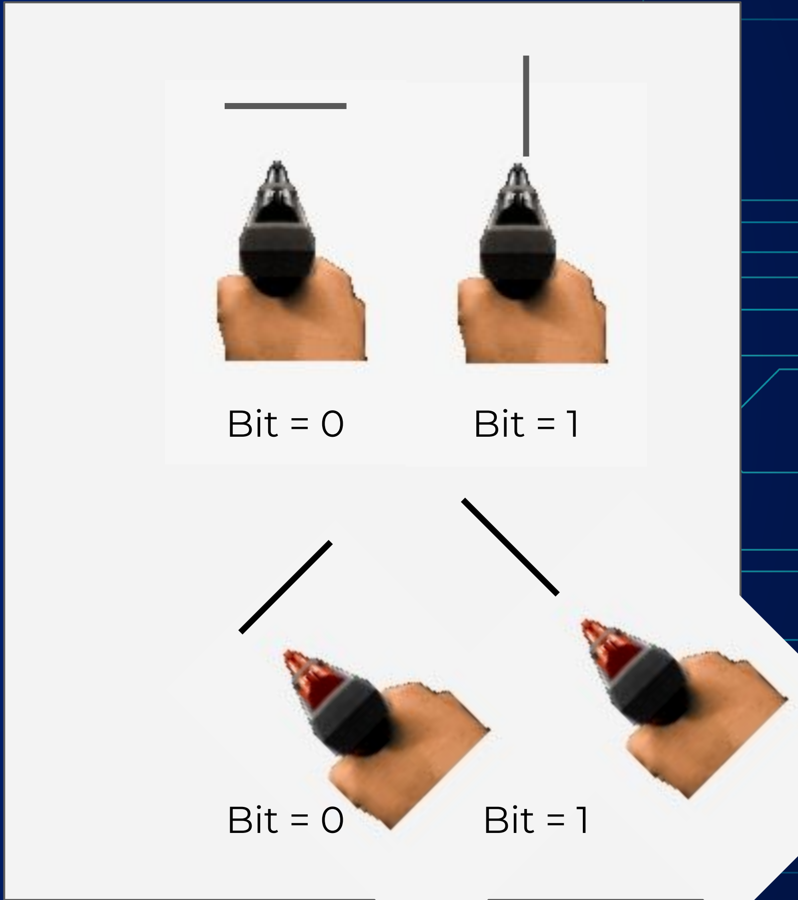
Alice's photon source / qubit "gun"

- Alice has a gun
- Alice randomly chooses whether to shoot it:
  - upright
  - at a 45 deg angle
- She loads her bullet (bit) and shoots it
- It is a flat bullet

2 Angles

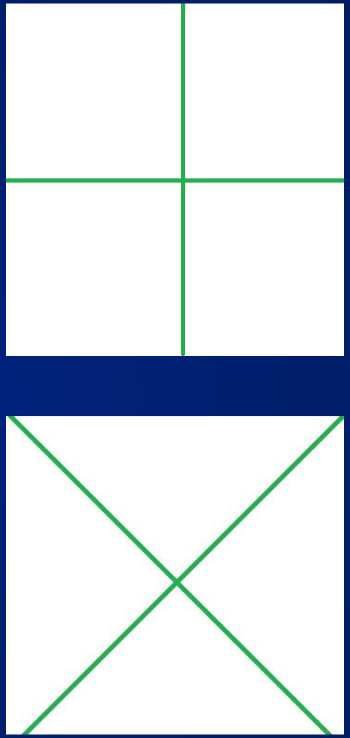
Fired upright

Fired at 45 deg angle



Eve’s / Bob’s Detector

- Eve and Bob have a detector that can only detect in one basis at any given point in time



Detectors

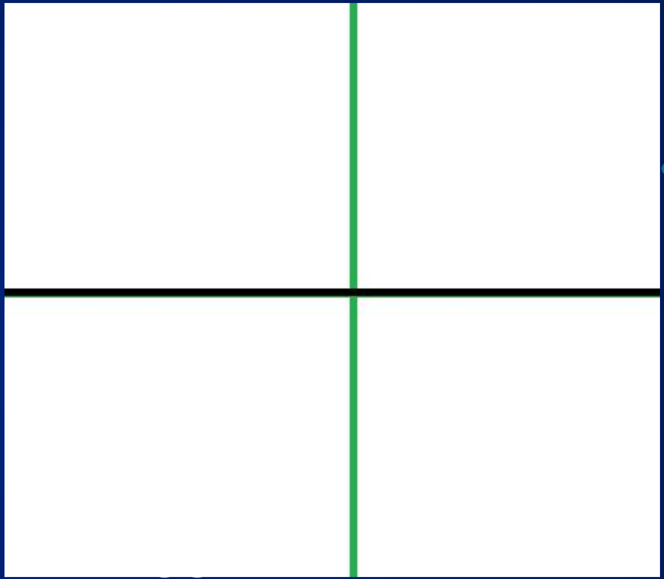
- Eve and Bob have a detector that can only detect in one basis at any given point in time

Detectors

- They can only rotate the detector

Bob’s Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)



If he chooses the same basis that Alice sent the bit in, he will correctly measure the bit value that Alice sent

Bob's Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob’s Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob's Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob's Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob's Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob's Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob’s Detector

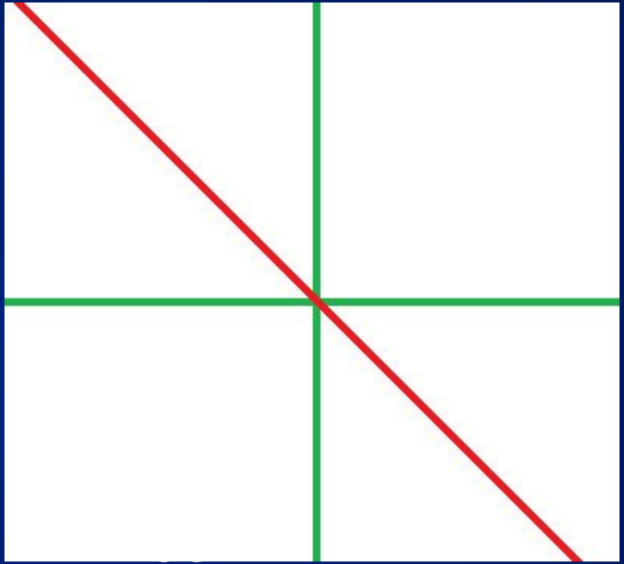
- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob’s Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)

Bob’s Detector

- Bob can only detect in one basis (either like this, or 45 degrees like an x)



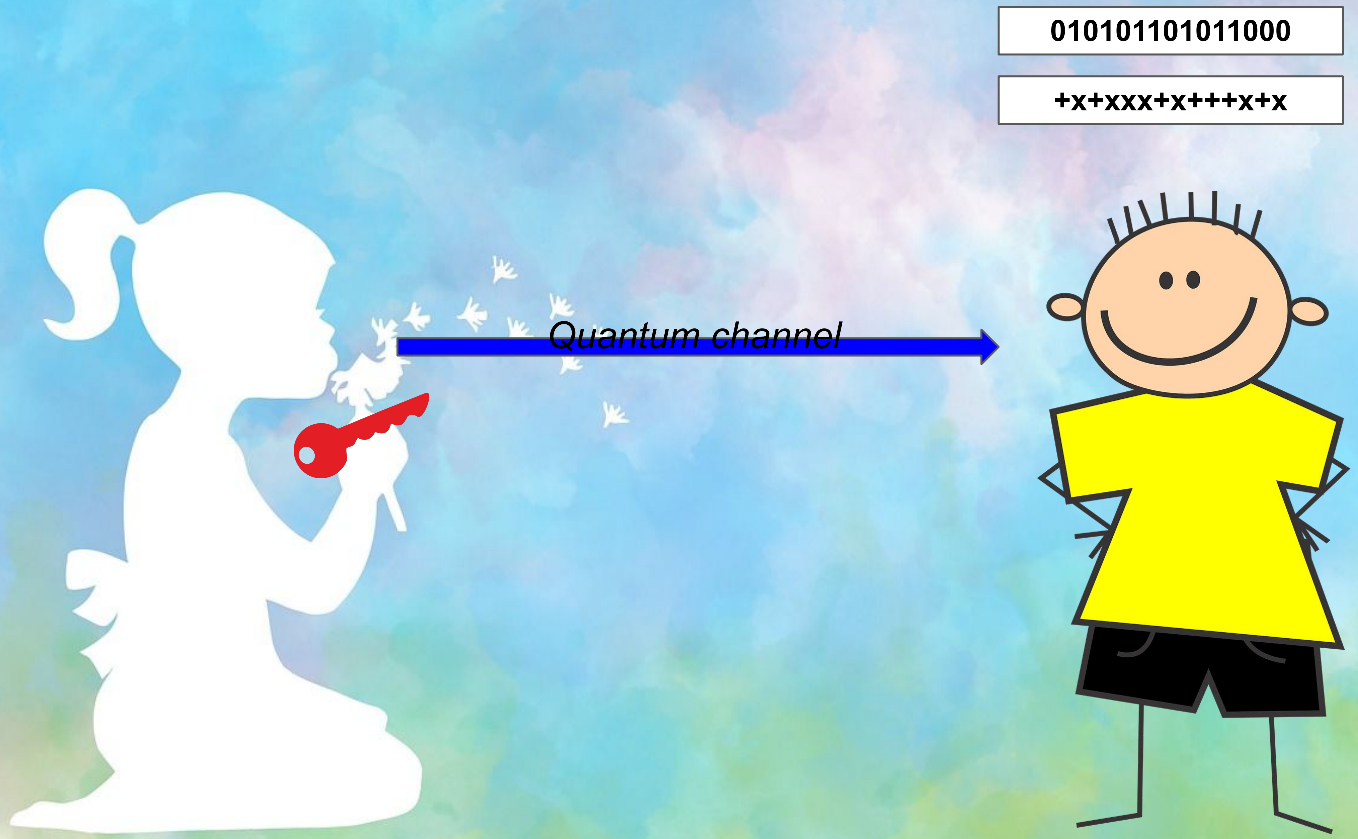
If he chooses the wrong basis, he will get a random value (can either go left or right, thus ‘0’ or ‘1’)

“Is a diagonal line horizontal or vertical?”

QKD

Detection

In the detection phase, Bob doesn't know which bits are sent in what basis. He just measures the bits in random bases and stores the bases he measured them in.



QKD Detection

In other words, 50% of his bits will be the same as Alice's, 50% will be random.

Correct Basis (50%)	Wrong Basis (50%)	
Correct Bits (50%)	Correct Bits (25%)	Wrong Bits (25%)

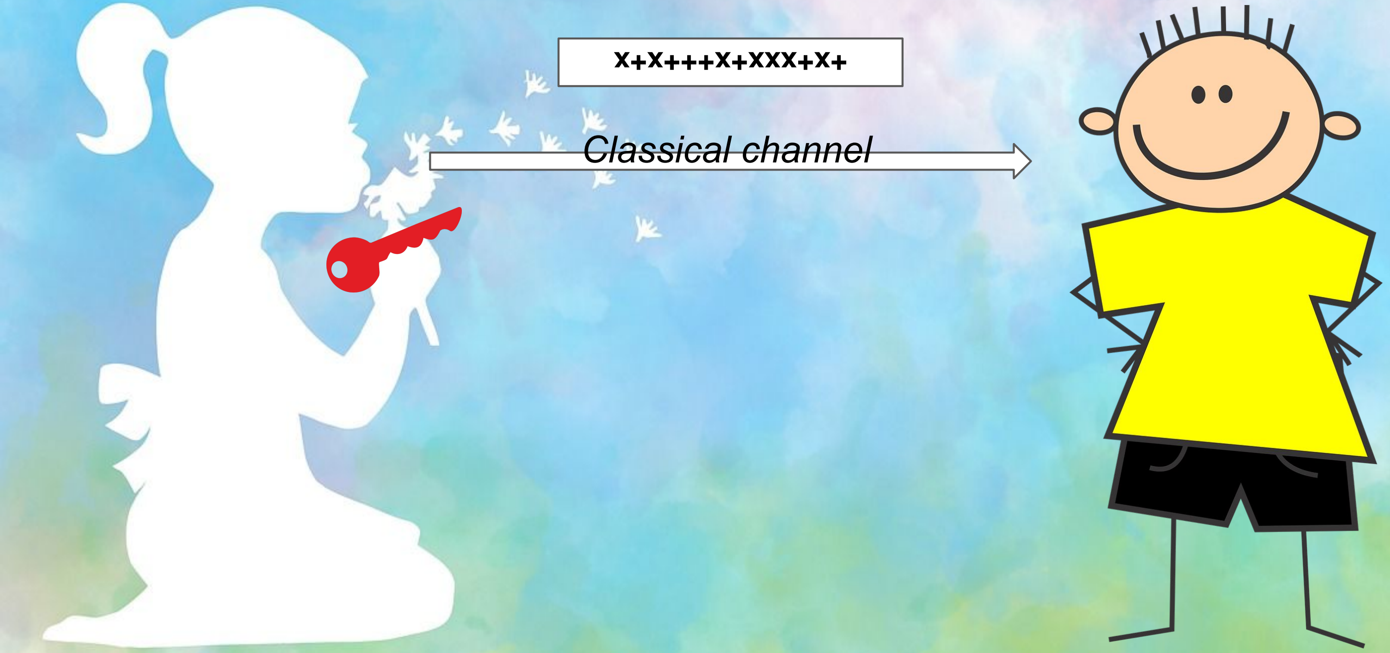
QKD

Sifting

At this point Alice and Bob ditch the quantum channel and use the classical channel.

Alice sends Bob the sequence of bases she sent the bits in. (+,x,+,+...)

Correct Basis (50%)	Wrong Basis (50%)	
Correct Bits (50%)	Correct Bits (25%)	Wrong Bits (25%)



QKD

Sifting

Bob discards the bits that he measured in the wrong basis.
(because they are now random bits)

He tells Alice which bits he discarded and she also discards those bits.

We call the result the
raw sifted key.

Correct Basis (50%)
Correct Bits (50%)

In a perfect scenario, this has no errors.

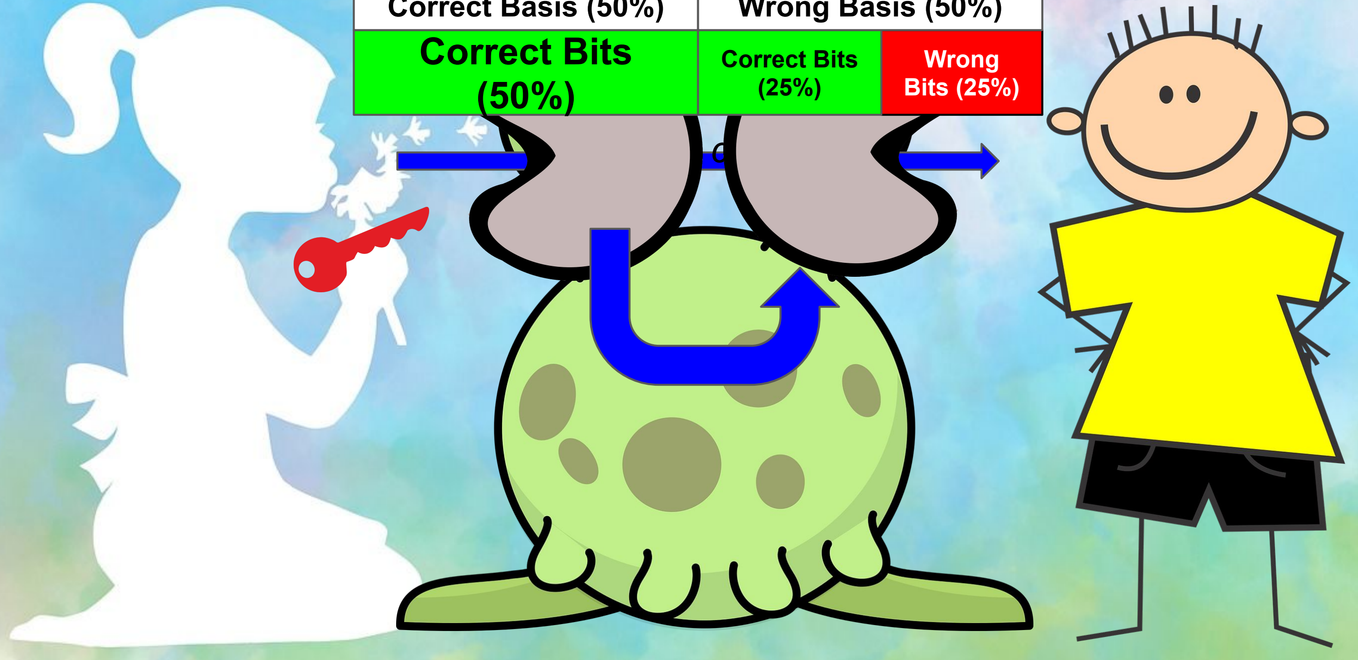
Why does it have to be so complicated?

Because Eve doesn't know the basis either, so if she's intercepting and resending, she also has to measure each bit with a random basis.

50% of Eve's bits will be the same as Alice's, 50% will be random.

Eve resends all the bits she got (random / fixed basis doesn't matter)

Correct Basis (50%)	Wrong Basis (50%)	
Correct Bits (50%)	Correct Bits (25%)	Wrong Bits (25%)



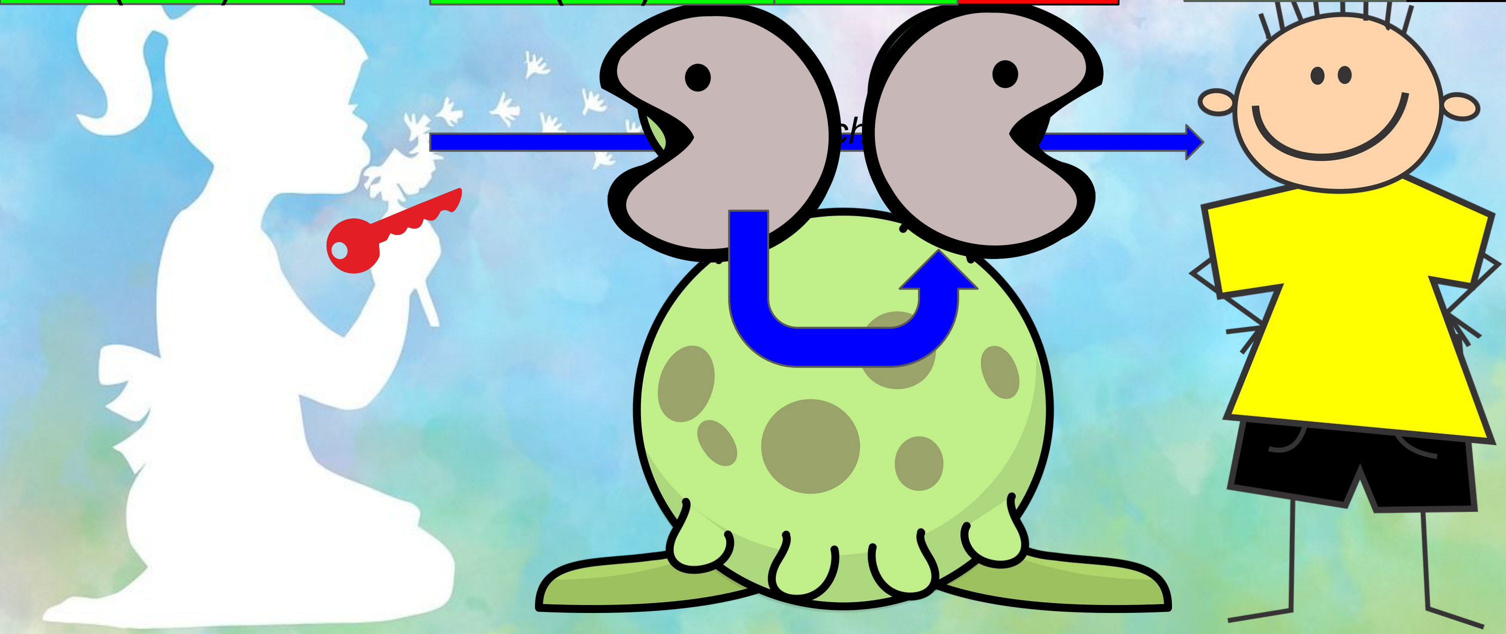
QKD

Detection

Thus with Eve, Bob will have some errors.

Correct Bits (100%)

Correct Basis (50%)	Wrong Basis (50%)	
Correct Bits (50%)	Correct Bits (25%)	Wrong Bits (25%)



Correct Basis (50%)		Wrong Basis (50%)			
Correct Bits (37.5%)	Wrong Bits (12.5%)	18.75 %	18.75 %	6.25 %	6.25 %

QKD

Detection

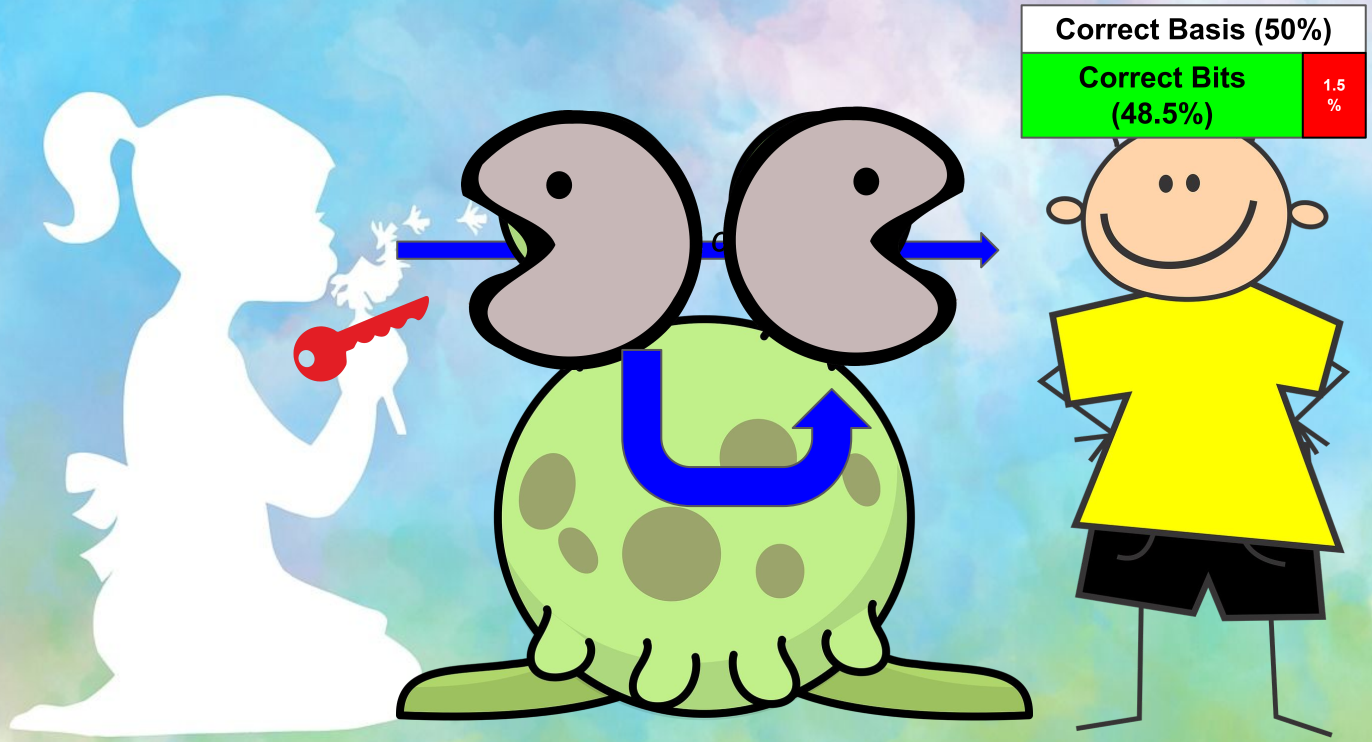
Total Error rate without Eve:
25%

Total Error rate with Eve:
(12.5 + 18.75 + 6.25) =
37.5%

After basis reconciliation:
¼ of raw sifted key is wrong

Eve, or other measurement errors (e.g. accidental detections) introduce errors, so it's never perfect.

For simplicity, let's just assume the errors were introduced by Eve who eavesdropped a small percentage of the bits.



How do you derive a key from this? Let's discuss a simplified version of Cascade, an error correction algorithm.

QKD

Error Correction

011010110000

011110110000

Imagine this scenario: Alice is a teacher and she’s sick, so she’s at home

Bob is the remedial teacher

QKD

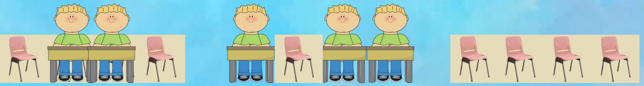
Error Correction

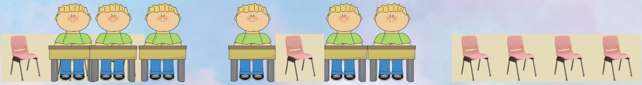
011010110000

011110110000

Imagine the key as a class of students, with each bit representing a chair

We assume the students sometimes play truant or go to the wrong class (“errors”)





011010110000

011110110000

QKD

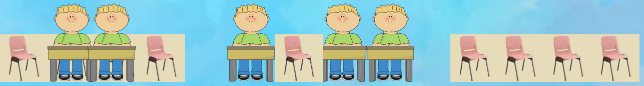
Error Correction

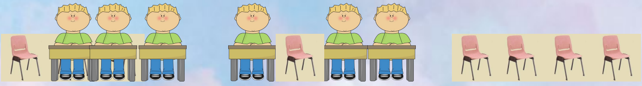
QKD

Error Correction

Alice can tell which chair is supposed to be occupied because she has the attendance sheet

Bob can only see the students and the empty chairs





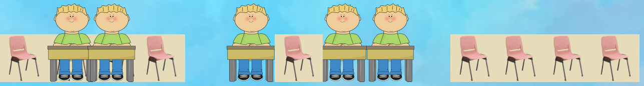
011010110000

011110110000

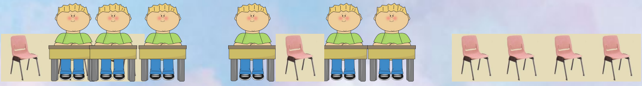
QKD Error Correction

Alice and Bob split the class into groups (e.g. groups of 4)

Alice then tells Bob "okay, the first group should have an even number of students. The second group should have an odd number of students…"



0110	1011	0000



0111	1011	0000

This is also known as "parity".

"Even, Odd, Even"
"0 1 0"

If Bob realizes a parity doesn’t tally, he’ll tell Alice.

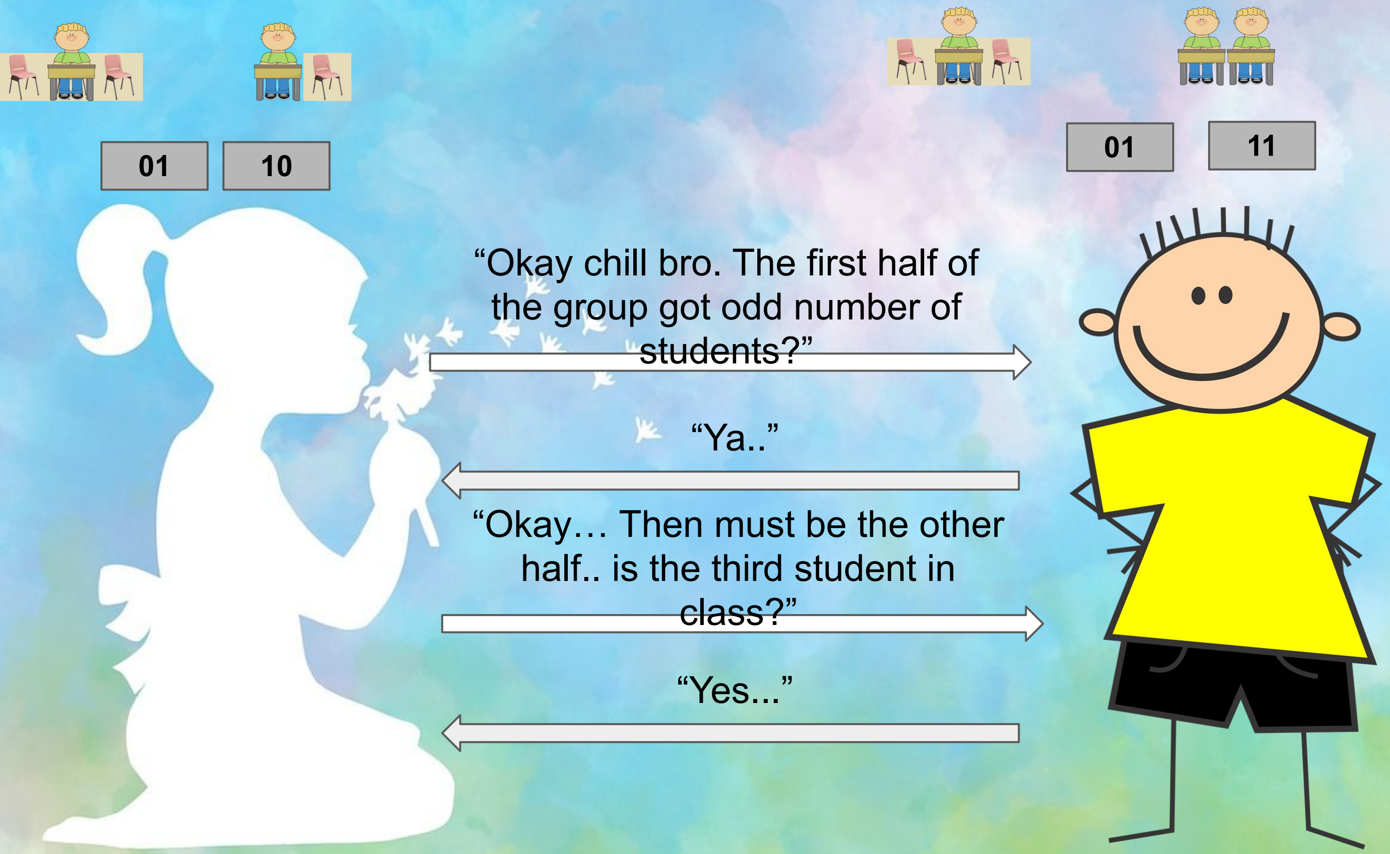
QKD

Error Correction

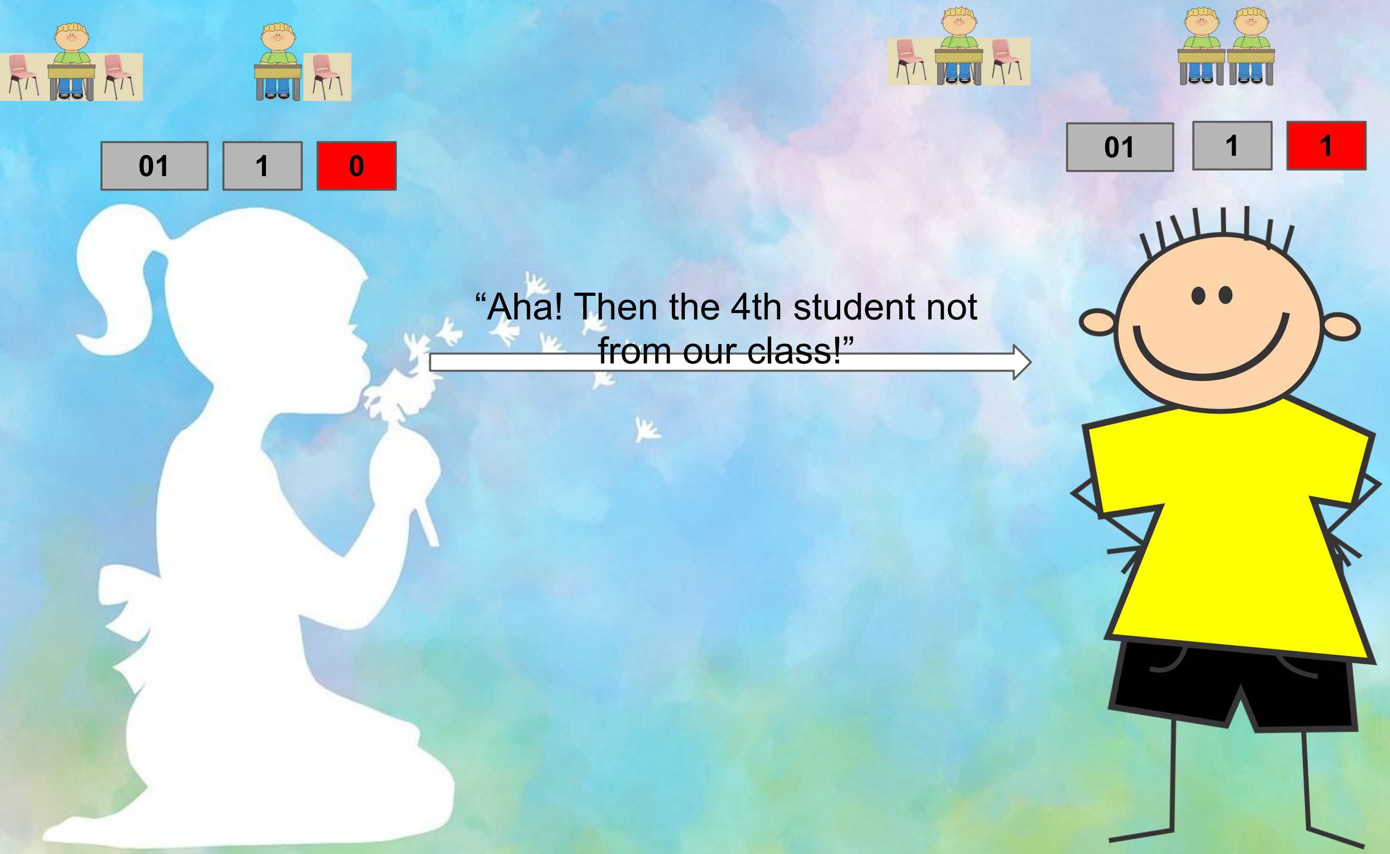
QKD

Error Correction

Once they identify a problem group, they will repeatedly split the group into two, find the chair that is causing the issue, then fix it.



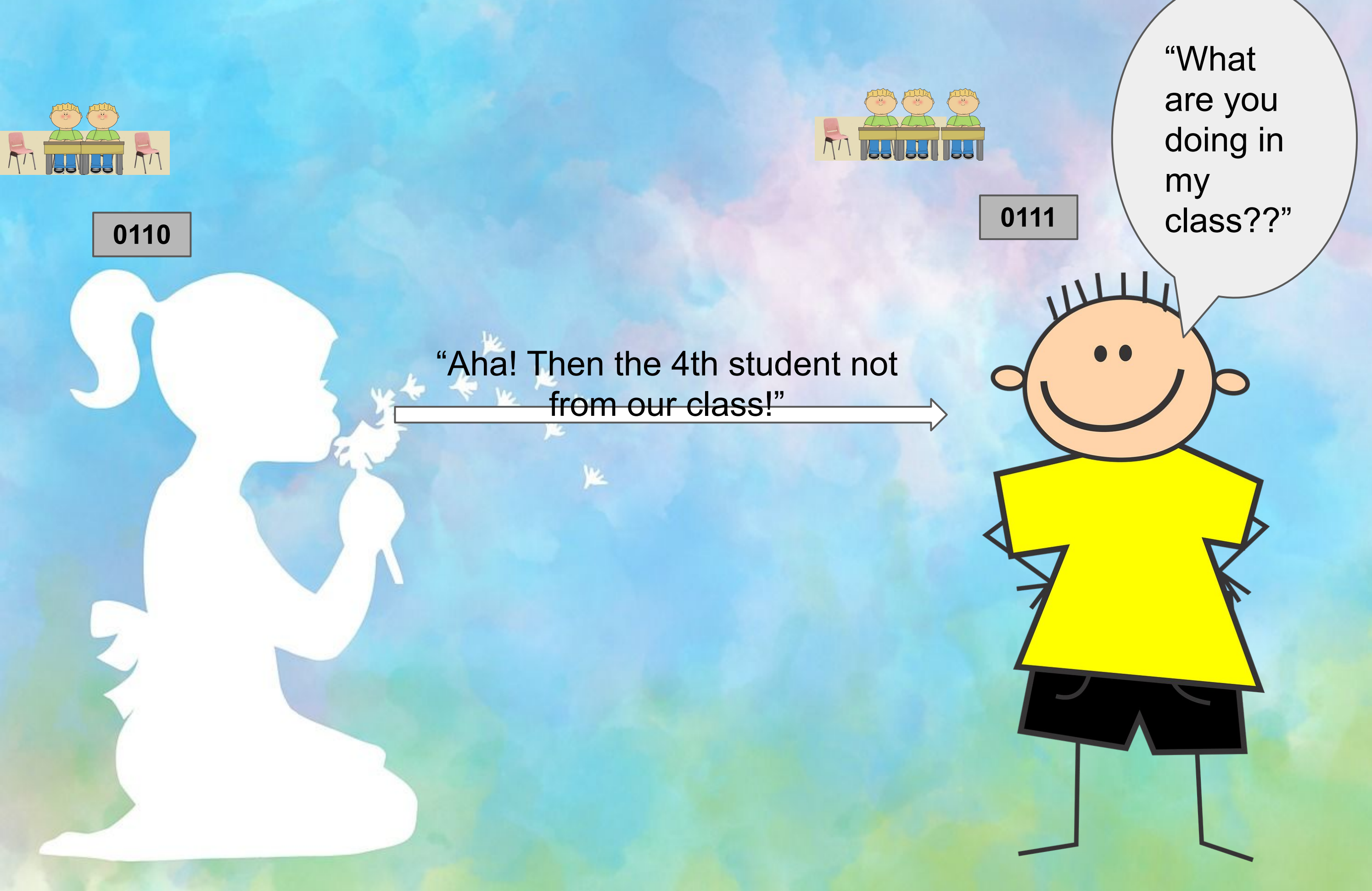
Once they identify a problem group, they will repeatedly split the group into two, find the chair that is causing the issue, then fix it.



QKD

Error Correction

Once they identify a problem group, they will repeatedly split the group into two, find the chair that is causing the issue, then fix it.



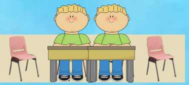
QKD

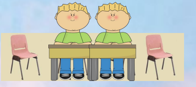
Error Correction

QKD

Error Correction

Once they identify a problem group, they will repeatedly split the group into two, find the chair that is causing the issue, then fix it.





0110

0110

“Aha! Then the 4th student not from our class!”

QKD

Error Correction

They do this multiple times with multiple block sizes until they are confident that their bits are the same.

They also keep a history of what they sent and received so when they correct a student, they can check back and see if there's any more bits they can correct (thus the name "Cascade" from the cascading effect)

011010110000

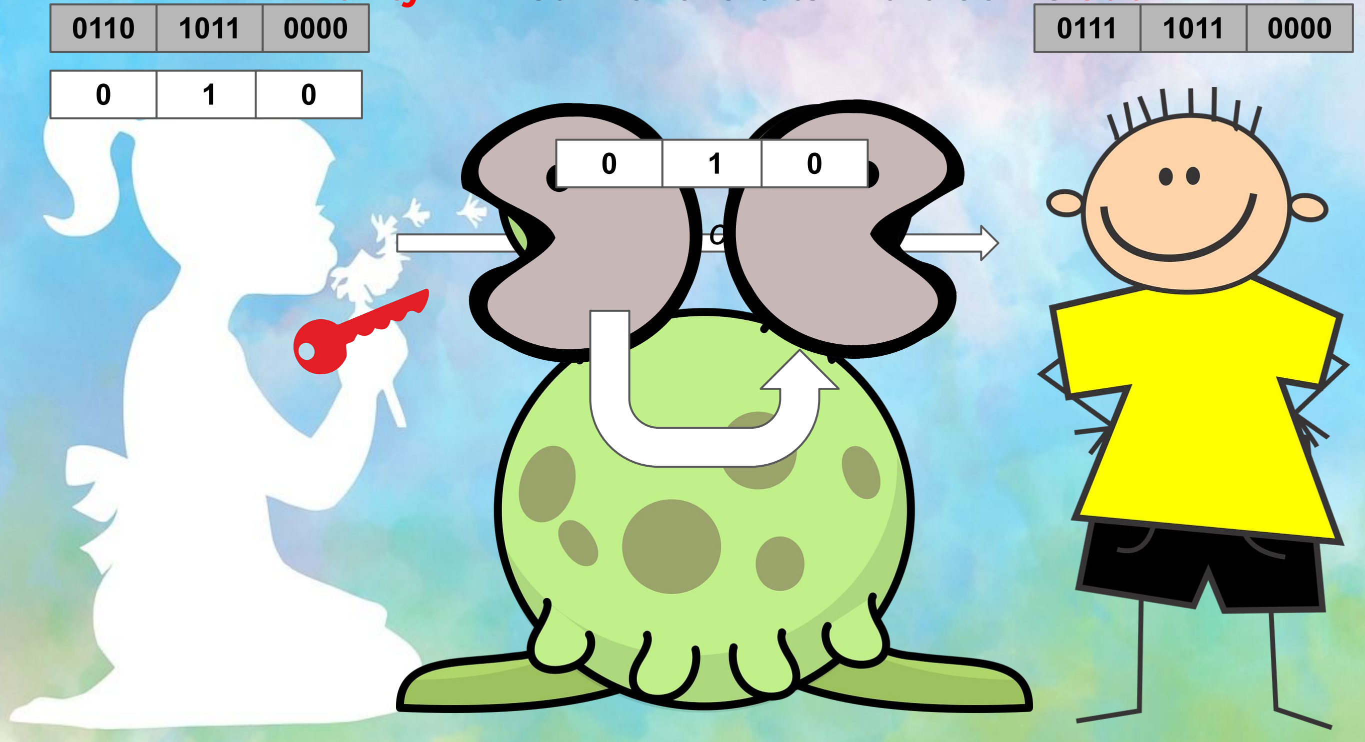
011010110000

QKD

Error Correction

How do you derive a key from this? Let's discuss a simplified version of Cascade, an error correction algorithm.
So Alice will split her key into blocks.
She calculates the parity of each block.

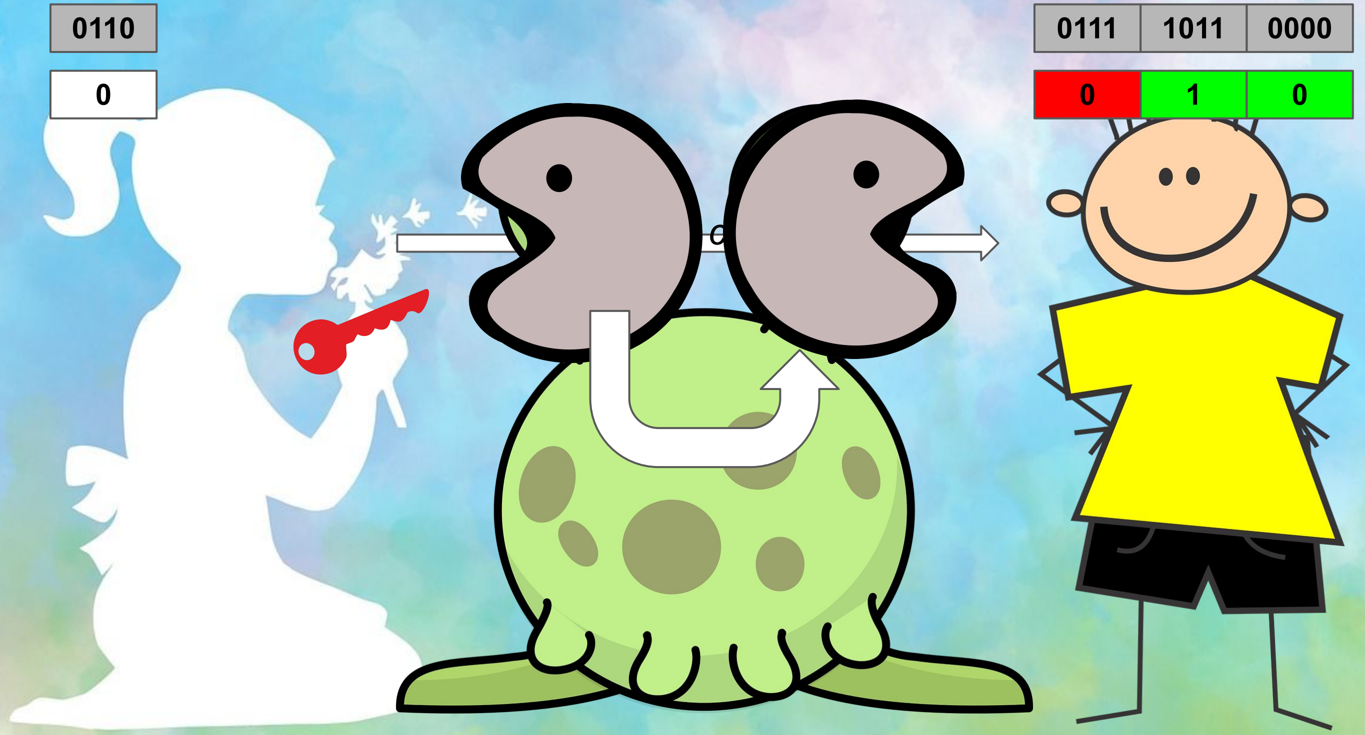
Parity = 0 if sum of the bits in a block is even
Parity = 1 if sum of the bits in a block is odd



Bob checks the parities of each block. If he finds there is a difference, he knows that block has an error and he will inform Alice.

QKD

Error Correction



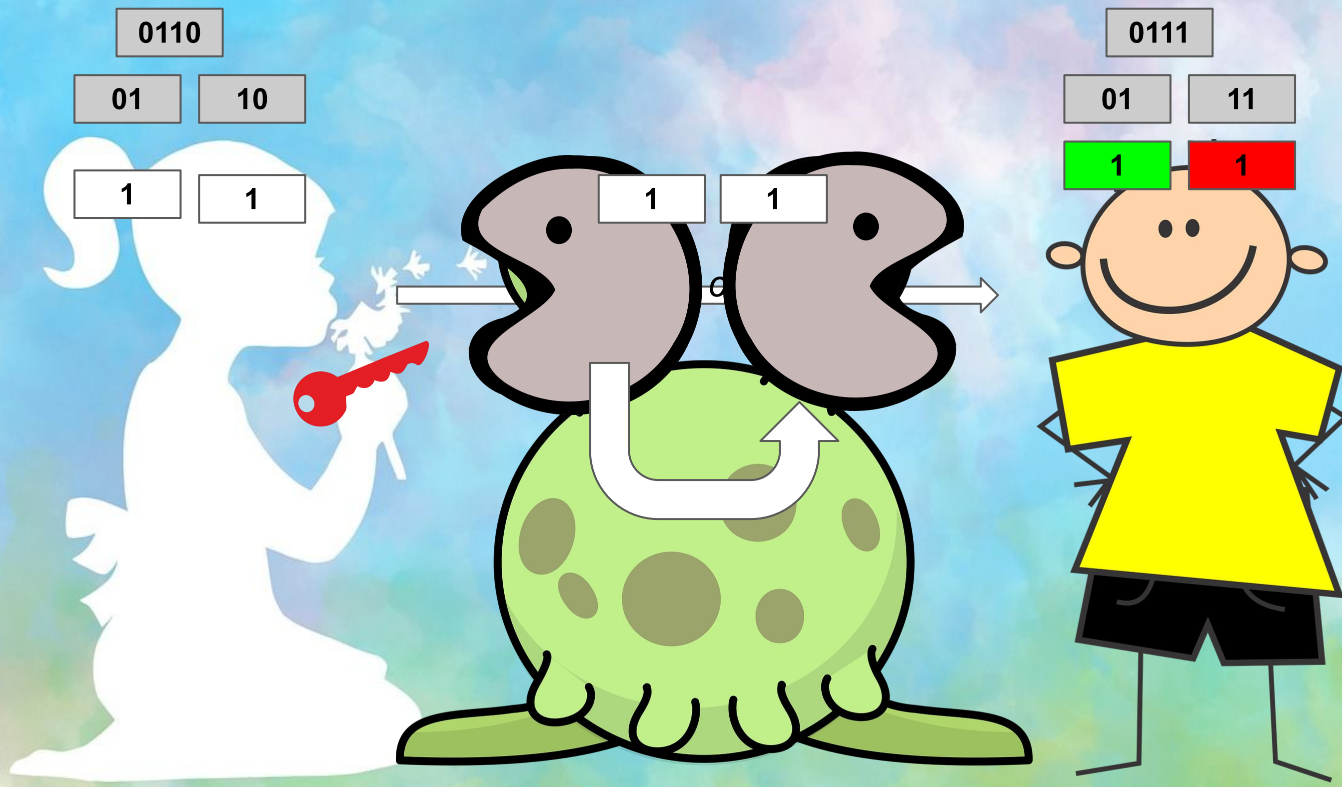
QKD

Error Correction

Bob checks the parities of each block. If he finds there is a difference, he will inform Alice.

Alice will then recursively split the block and calculate parities for each sub-block until the error is in a block of size 1, then they correct the error by flipping the bit.

They do this multiple times with multiple block sizes until they are confident that their bits are correct and their sifted keys have been corrected.



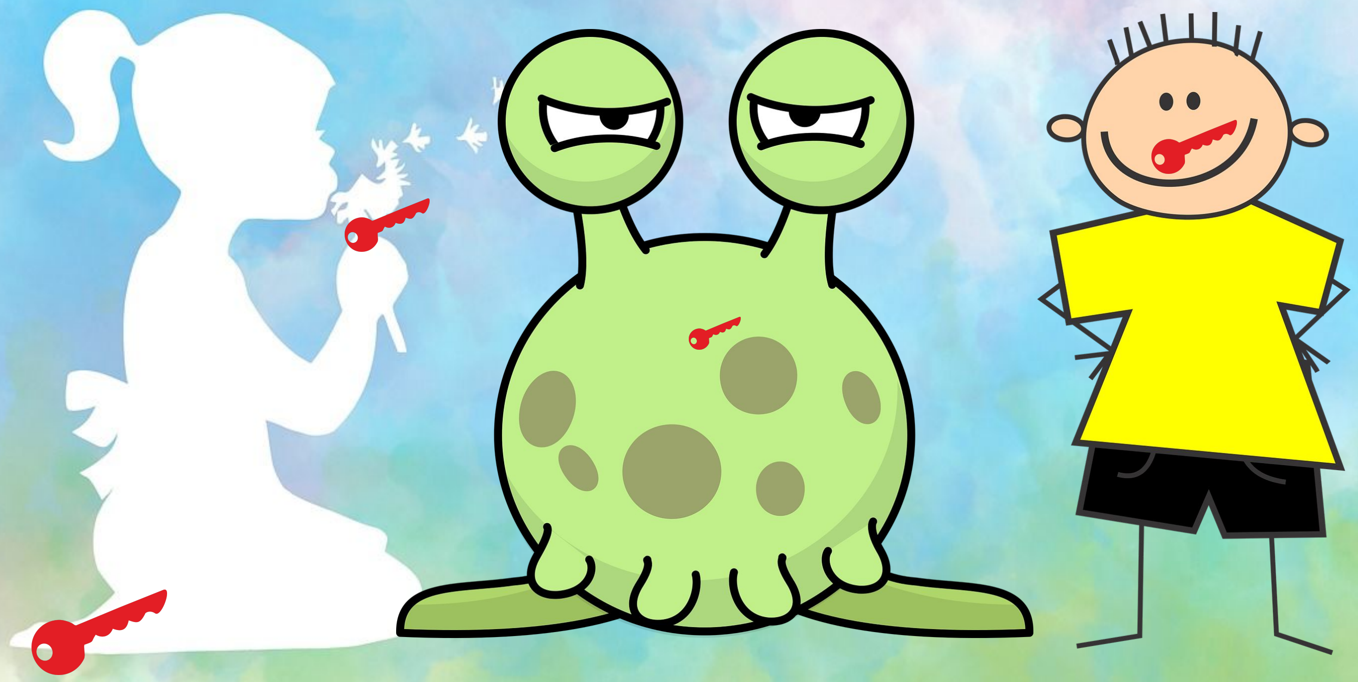
QKD

Error Correction

At this point Alice & Bob have a fraction of their raw sifted key. However, since communications were done through classical channel, Eve also knows:

1. What bases Alice used and the bases Bob accepted
2. Every parity bit that was exchanged between Alice and Bob

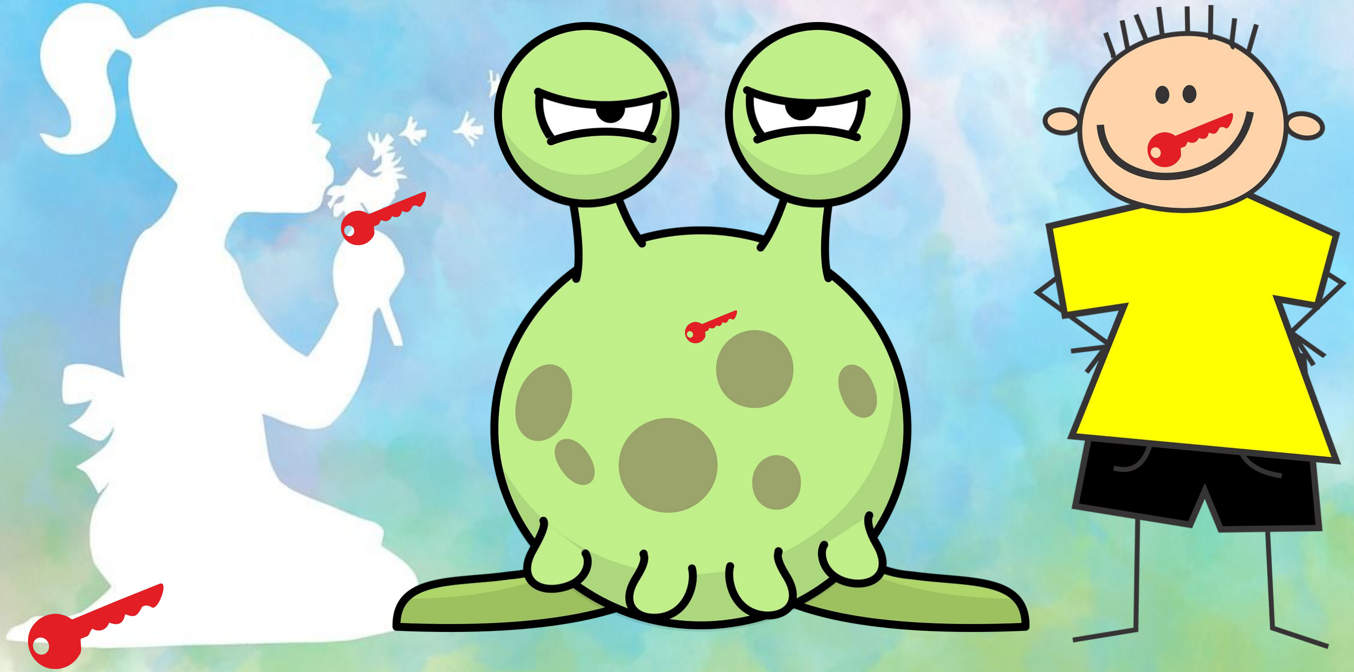
Eve can use these two pieces of information to derive parts of the key that Alice and Bob have now.



QKD

Privacy Amplification

So the question is: Given that Eve knows *t* bits, can they still use the key, or do they have to throw it away and choose a new key?



QKD

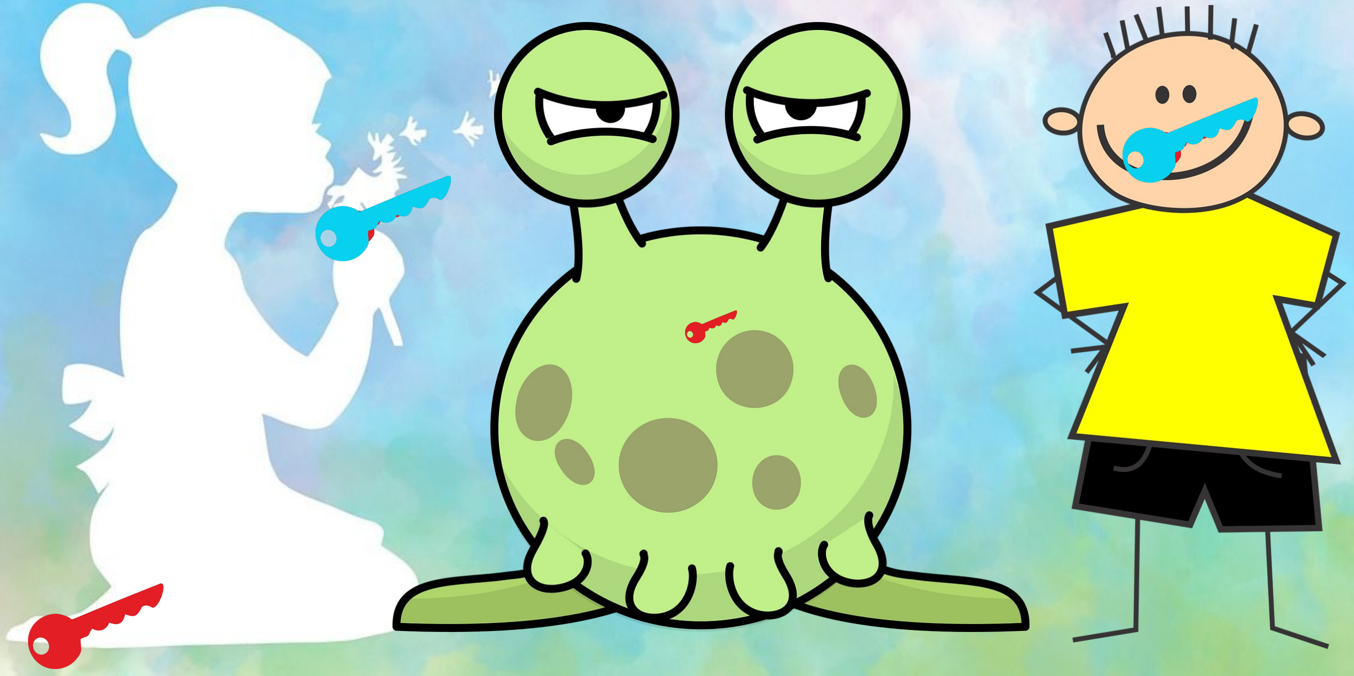
Privacy Amplification

QKD

Privacy Amplification

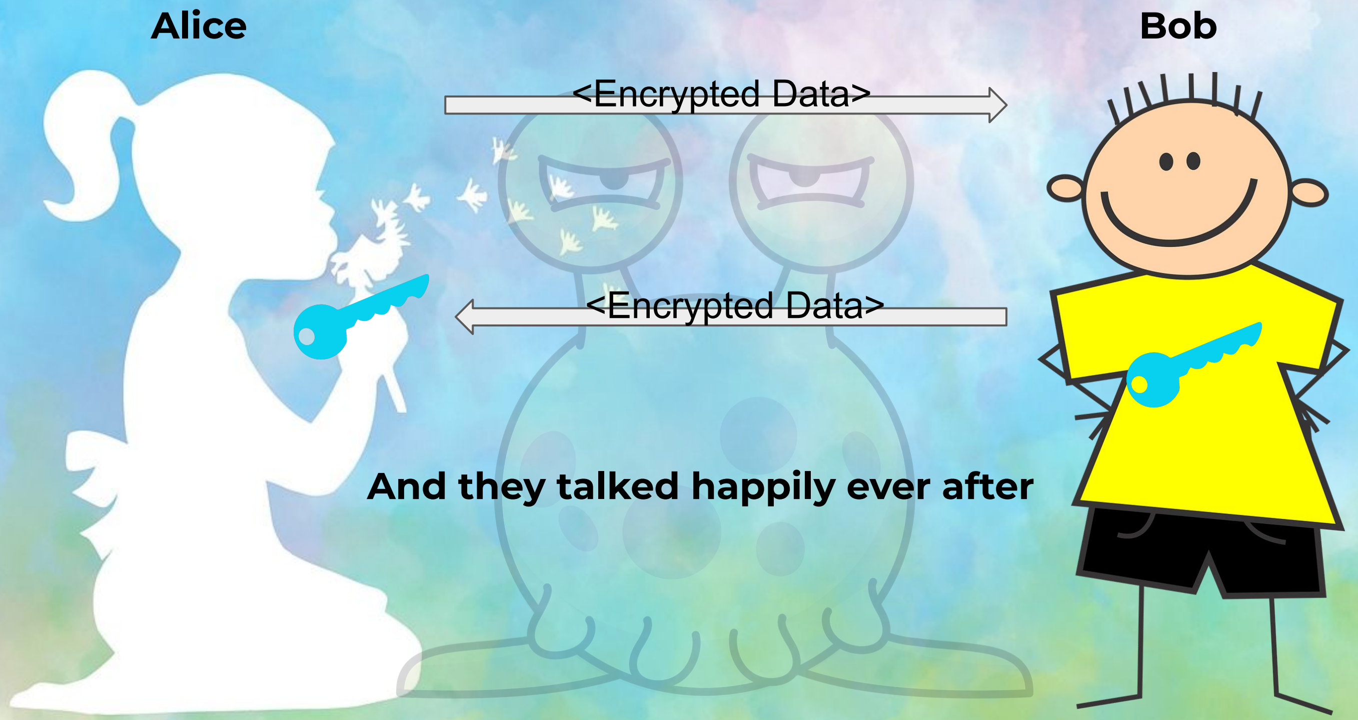
The answer is... they can still use it!

The leftover hash lemma tells us that they can produce a final key of length (n - t) from a raw sifted key of length *n*, over which Eve has negligible knowledge using a universal hash function. This is called privacy amplification, as it "amplifies" the privacy of the key.



QKD

Post-QKD



And they talked happily ever after

There's more…

How do they store their key?

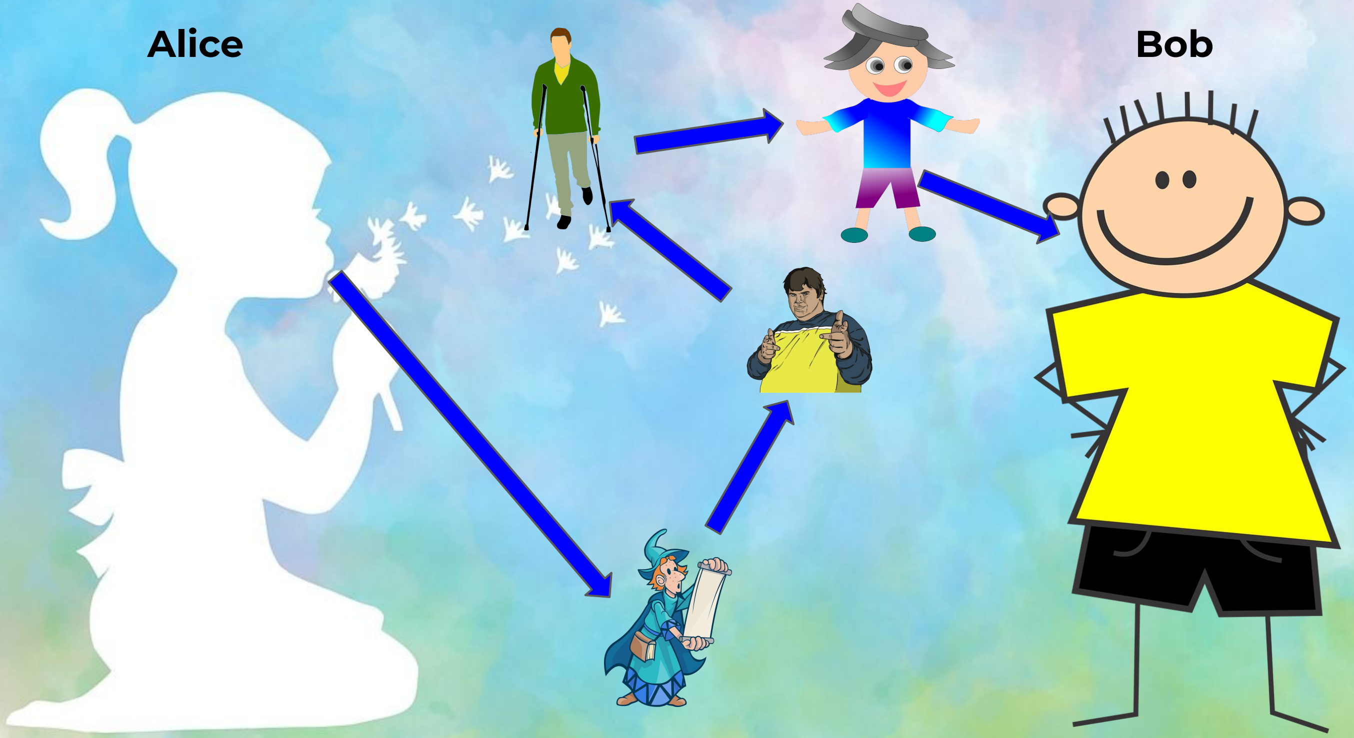
Alice

Bob

QKD

Post-QKD

What if Alice and Bob only have an indirect quantum channel connection?



QKD

Post-QKD

02

Open Source Solutions

Implementing QKD at a larger scale

Open Source Solutions

Questions to ask

- Can it be used with SpeQtral software stack?
- How well developed is it?
- How well maintained is it?

I've looked at 2 stacks:

- OpenQKDNetwork
- CQPToolkit

Open Source Solutions Review

1. OpenQKDNetwork

Reviewed on June 8

- A suite of Java Maven Projects to support QKD at a Wide Area Network (WAN) level
- Pros:
  - Good ideas (I am probably ill-equipped to evaluate their concepts)
- Cons:
  - Unoptimized implementation & naive algorithms used (appears to be proof of concept at best)
  - Incomplete
  - Zero documentation
  - Zero tests

Open Source Solutions Review

2. CQPToolkit

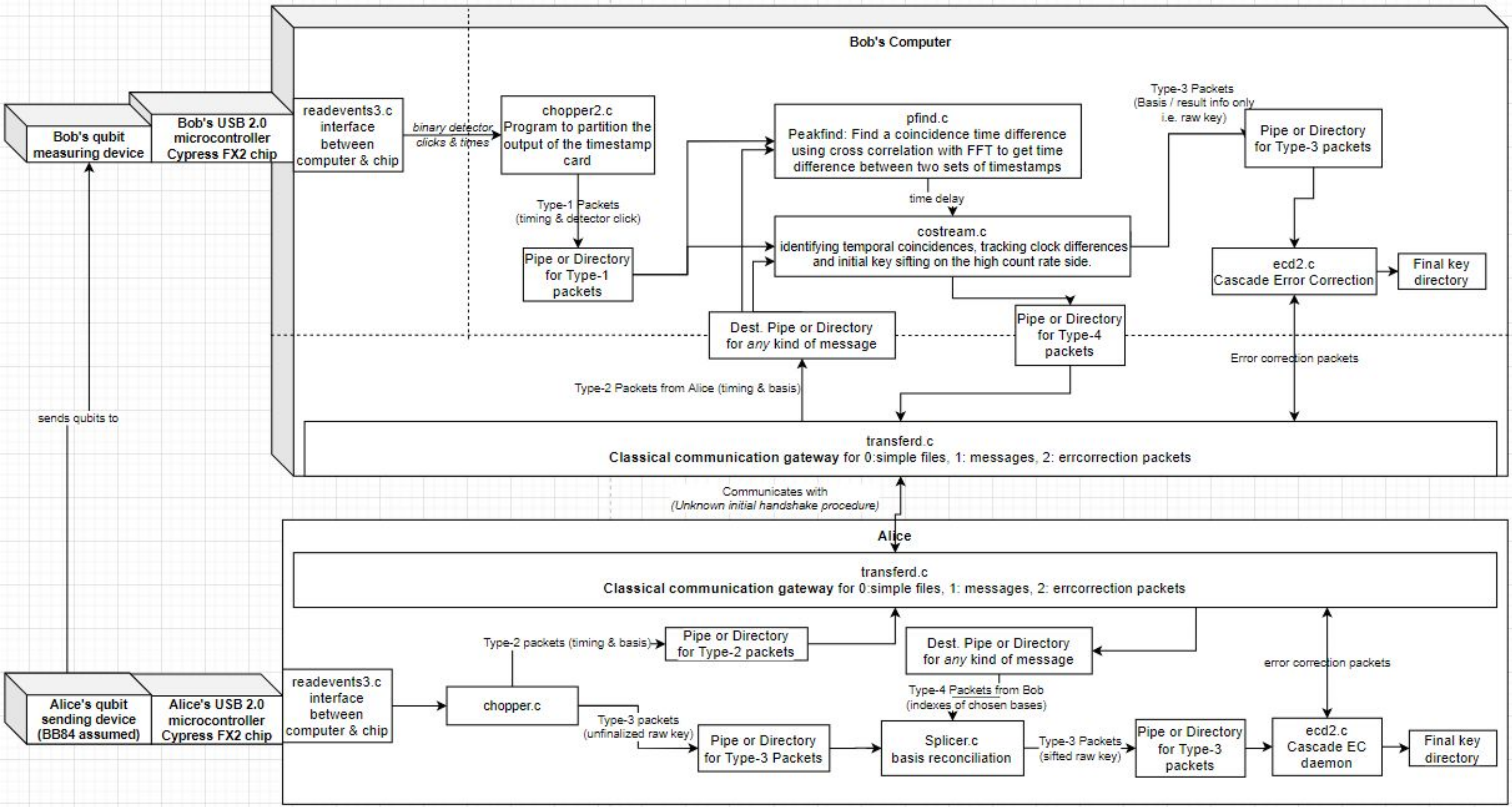
Reviewed on June 8

- A C++ framework to manage QKD devices and keys (can support QKD at a WAN level as well)
- Pros:
    - Comprehensive implementation
        - Decent documentation
        - Feels complete
        - Has test cases
- Cons:
    - Can be useful but will require technical expertise
        - Large and complicated internal code
        - Documentation not in-line with code
        - QKD portion will require our own implementation

03

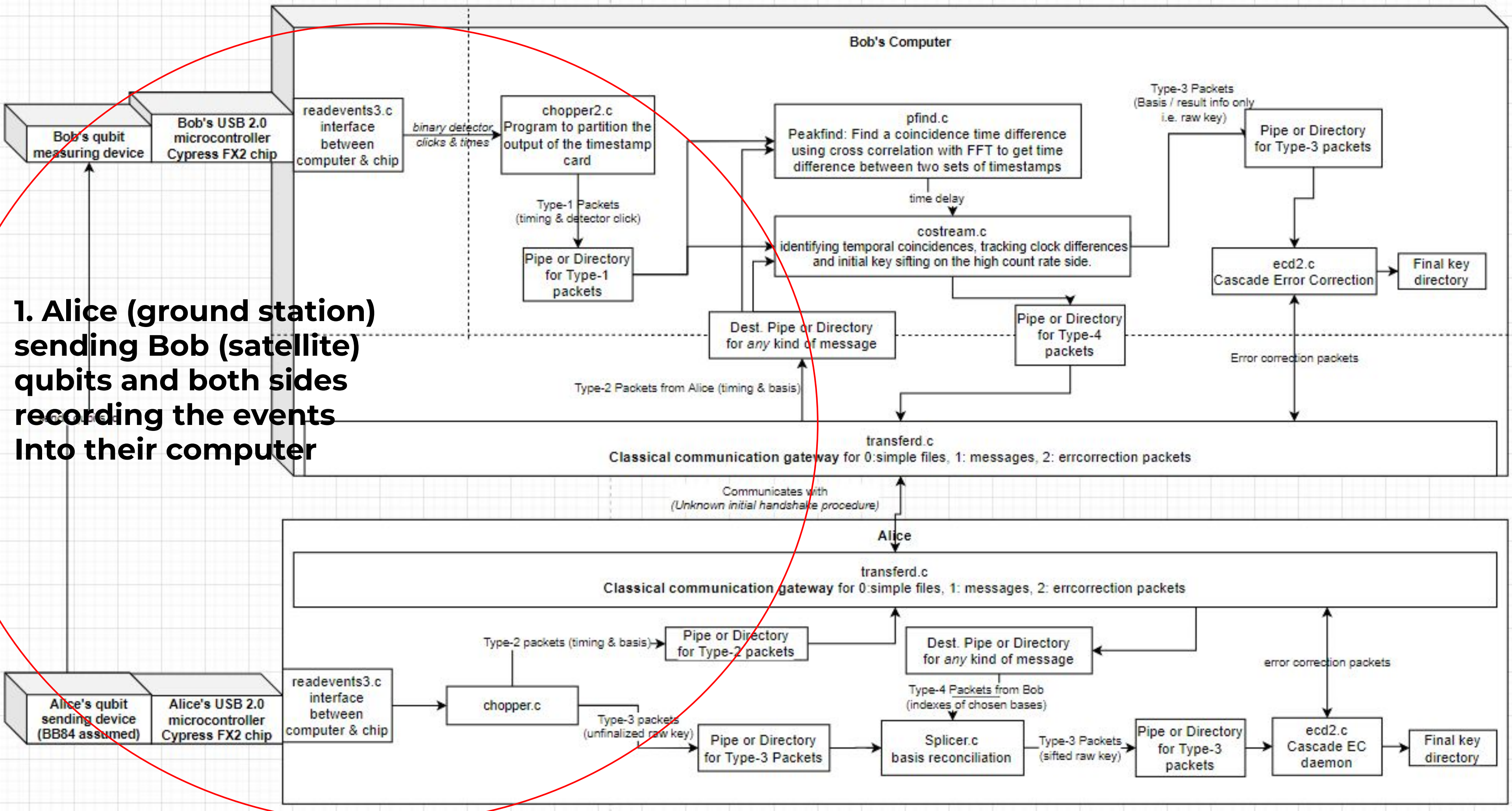
qCrypto

Brief architectural overview of our software stack, which is a suite of programs for QKD

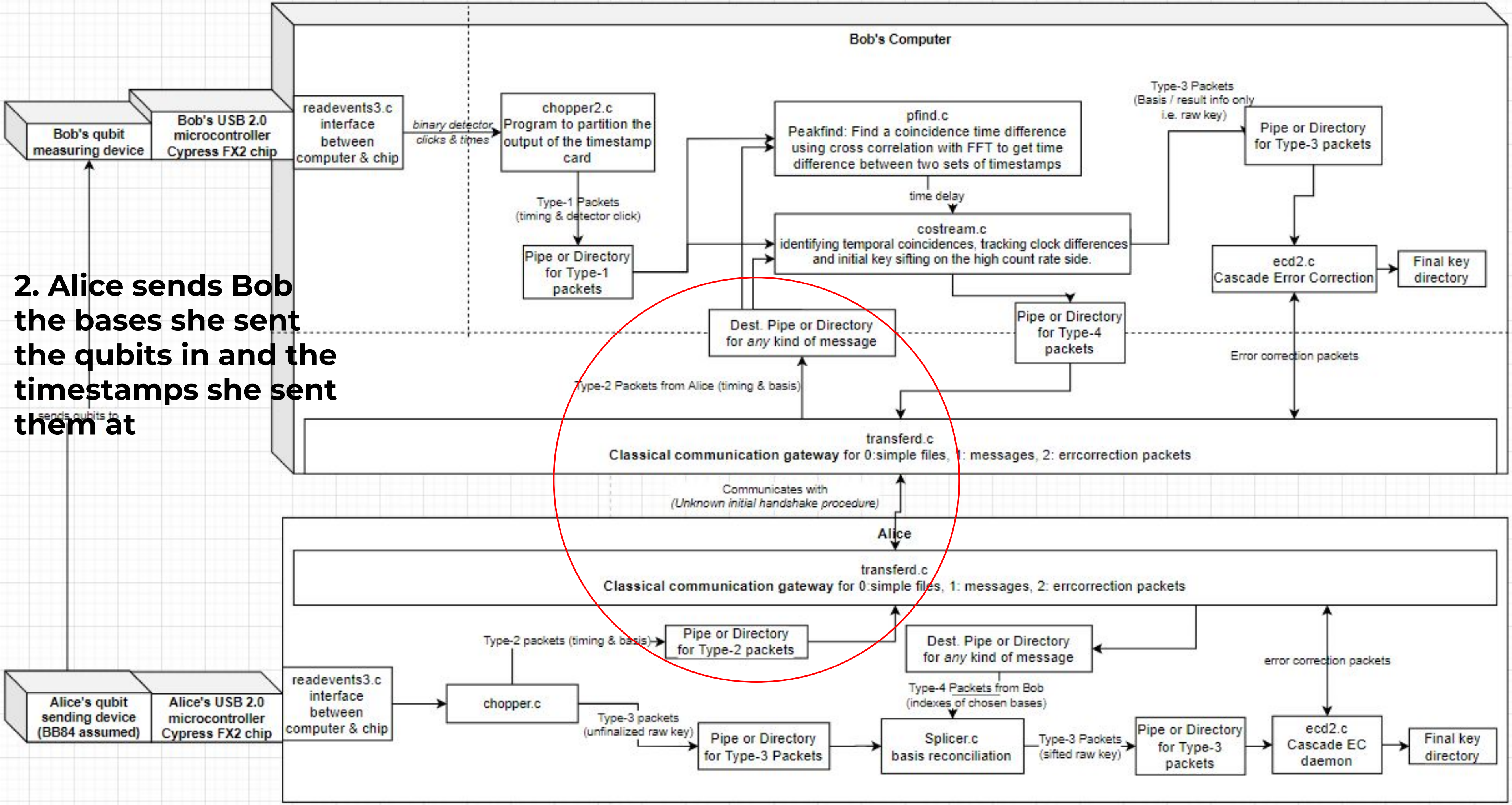


Bob's Computer
Bob's qubit measuring device
Bob's USB 2.0 microcontroller Cypress FX2 chip
readevents3.c interface between computer & chip
binary detector clicks & times
chopper2.c Program to partition the output of the timestamp card
Type-1 Packets (timing & detector click)
Pipe or Directory for Type-1 packets
pfind.c Peakfind: Find a coincidence time difference using cross correlation with FFT to get time difference between two sets of timestamps
time delay
costream.c identifying temporal coincidences, tracking clock differences and initial key sifting on the high count rate side.
Type-3 Packets (Basis / result info only i.e. raw key)
Pipe or Directory for Type-3 packets
ecd2.c Cascade Error Correction
Final key directory
Dest. Pipe or Directory for *any* kind of message
Pipe or Directory for Type-4 packets
Error correction packets
Type-2 Packets from Alice (timing & basis)
sends qubits to
transferd.c
Classical communication gateway for 0:simple files, 1: messages, 2: errcorrection packets
Communicates with
(Unknown initial handshake procedure)
Alice
transferd.c
Classical communication gateway for 0:simple files, 1: messages, 2: errcorrection packets
Type-2 packets (timing & basis)
Pipe or Directory for Type-2 packets
Dest. Pipe or Directory for *any* kind of message
error correction packets
Alice's qubit sending device (BB84 assumed)
Alice's USB 2.0 microcontroller Cypress FX2 chip
readevents3.c interface between computer & chip
chopper.c
Type-3 packets (unfinalized raw key)
Pipe or Directory for Type-3 Packets
Type-4 Packets from Bob (indexes of chosen bases)
Splicer.c basis reconciliation
Type-3 Packets (sifted raw key)
Pipe or Directory for Type-3 packets
ecd2.c Cascade EC daemon
Final key directory

1. Alice (ground station) sending Bob (satellite) qubits and both sides recording the events Into their computer

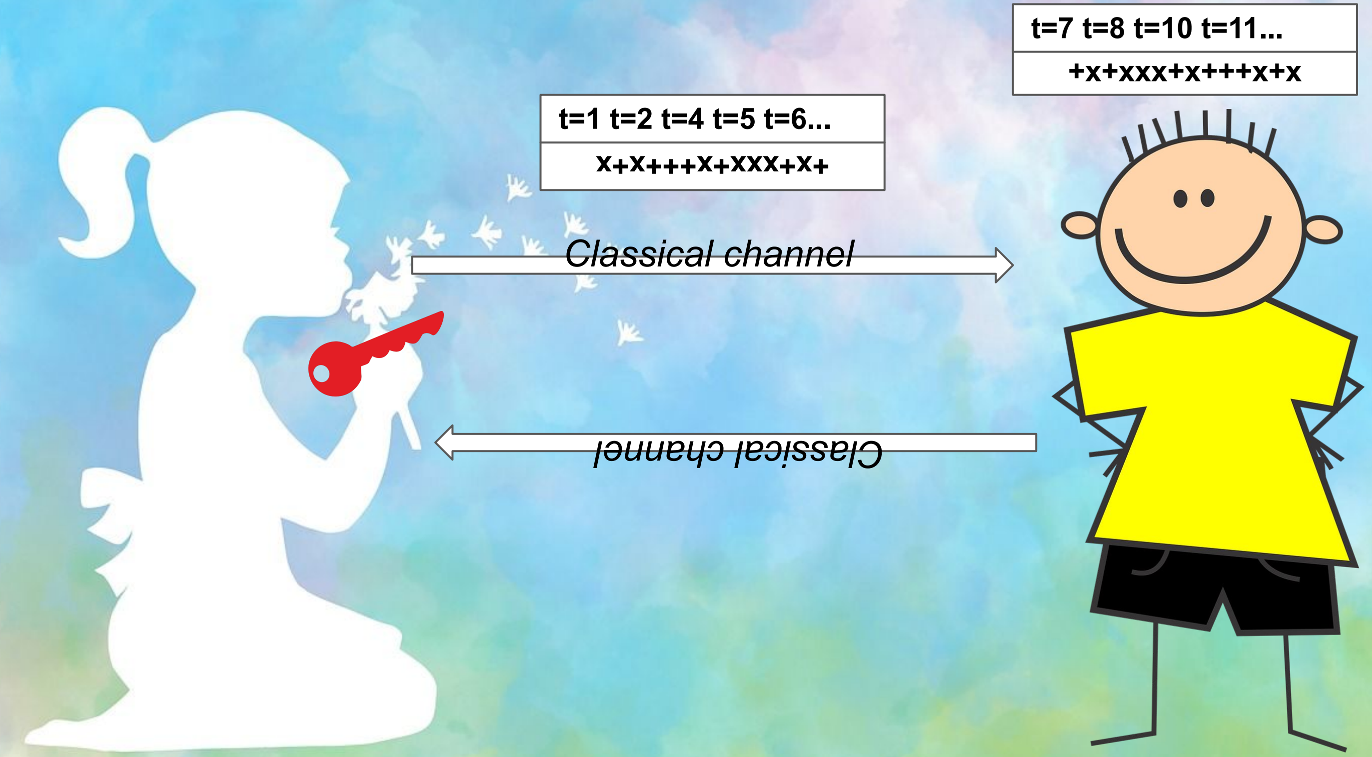


2. Alice sends Bob the bases she sent the qubits in and the timestamps she sent them at

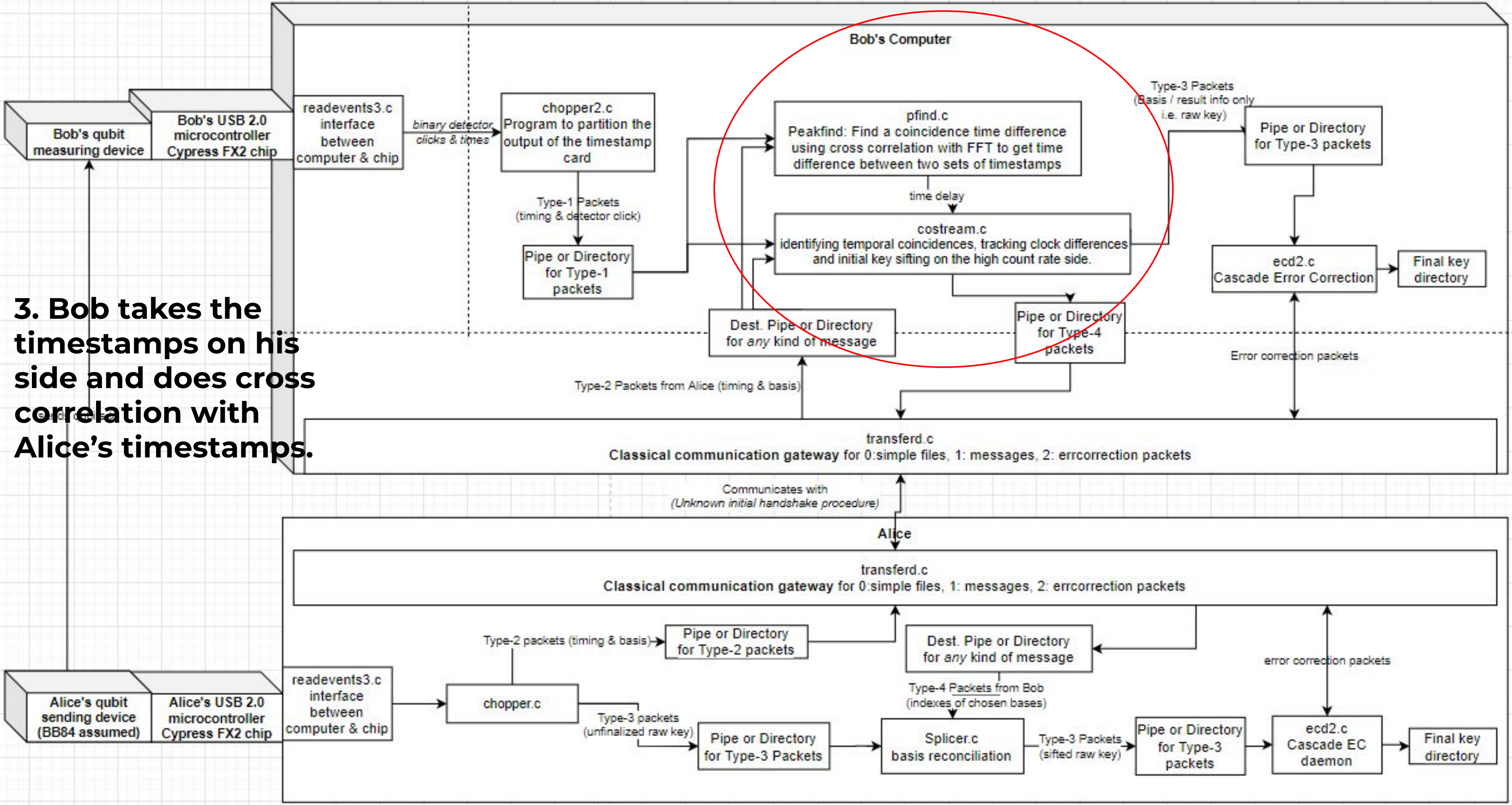


Since the quantum channel is just Alice shooting photons (light) at Bob, some photons may go missing or Bob may measure random light thinking it was from Alice.

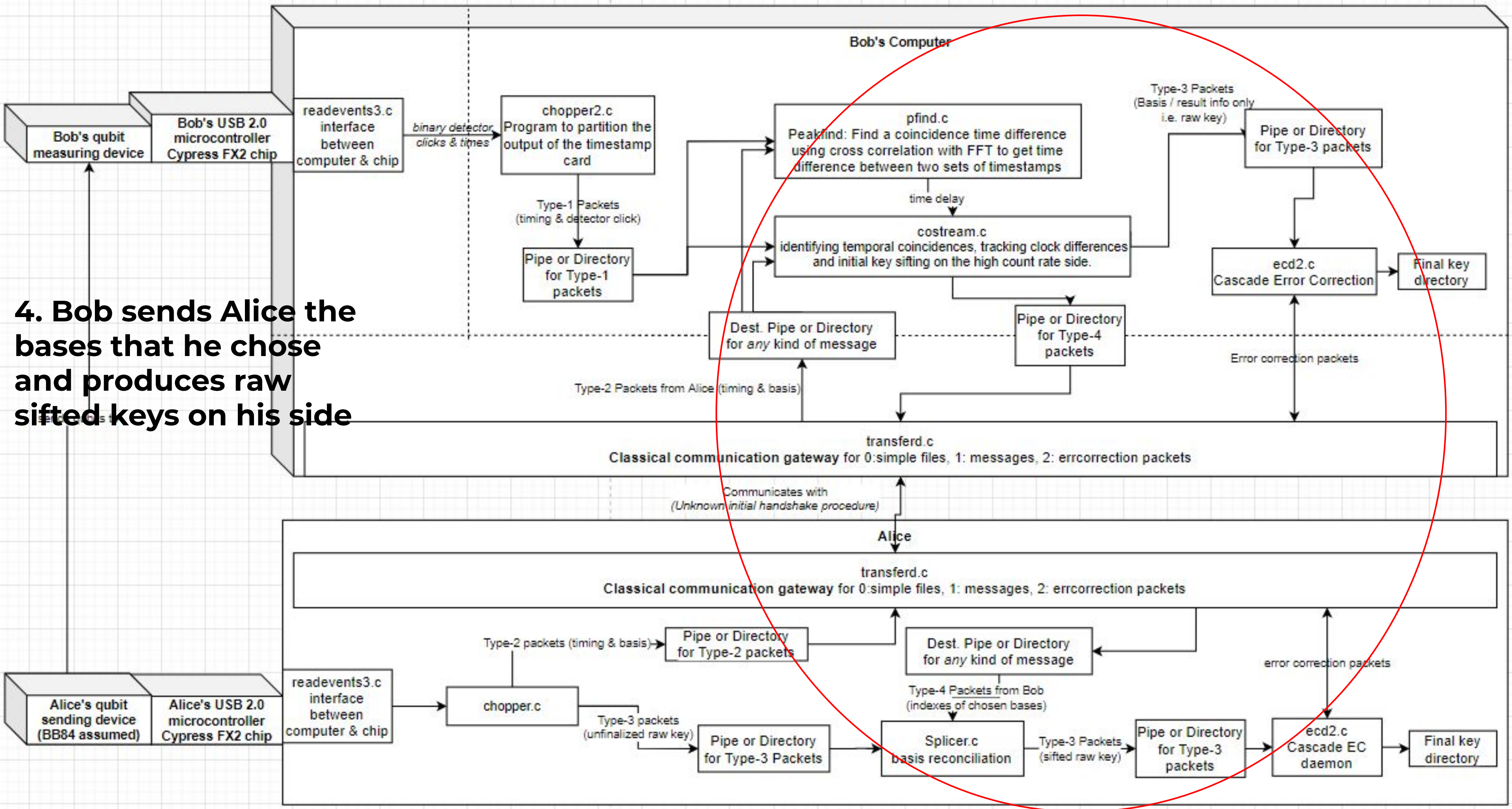
So on top of sending the bases, Alice also sends the timestamps, which Bob can cross correlate with his set of timestamps to determine which bases match up.



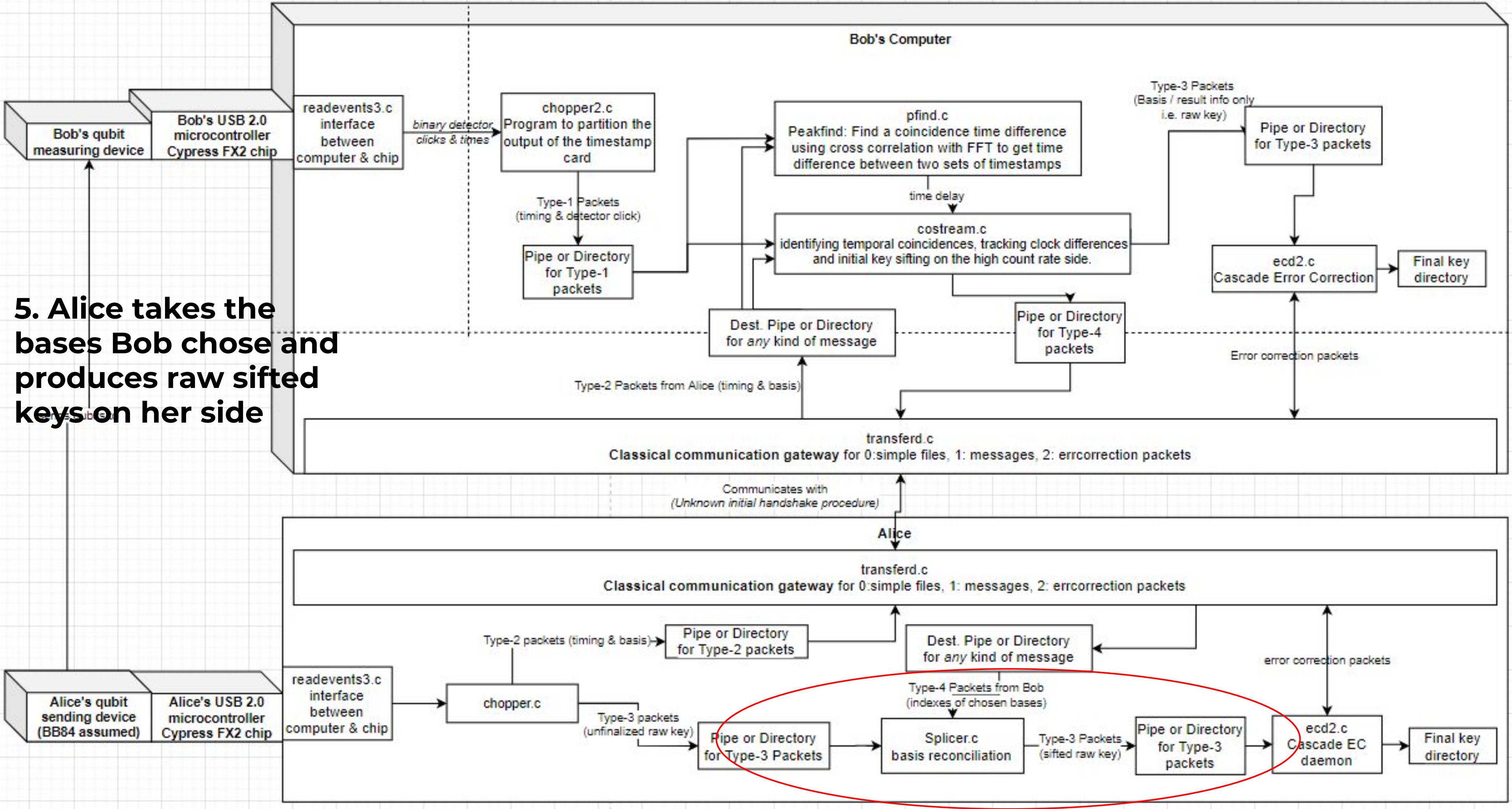
3. Bob takes the timestamps on his side and does cross correlation with Alice's timestamps.



4. Bob sends Alice the bases that he chose and produces raw sifted keys on his side

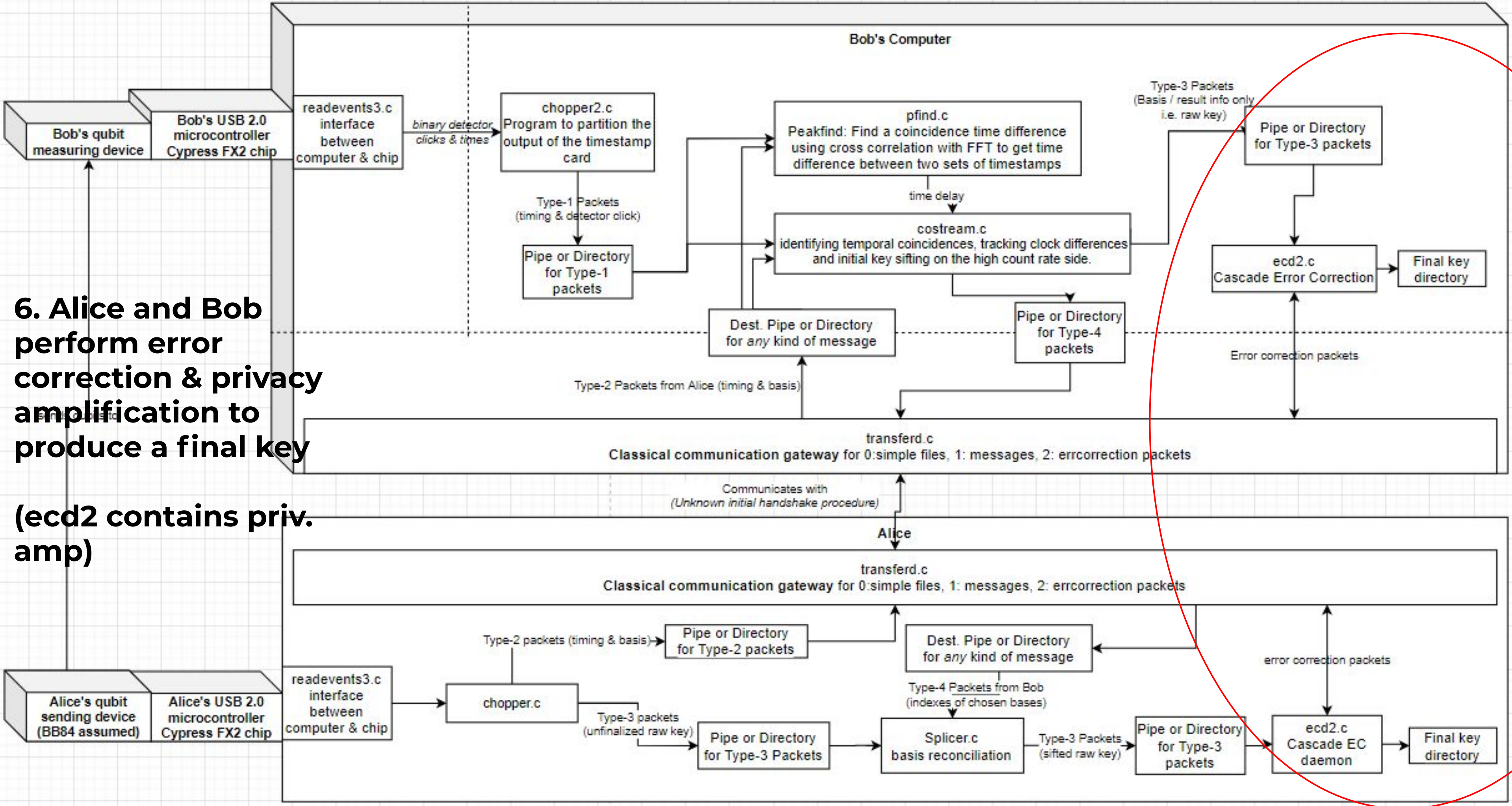


5. Alice takes the bases Bob chose and produces raw sifted keys on her side



6. Alice and Bob perform error correction & privacy amplification to produce a final key

(ecd2 contains priv. amp)



qCrypto Stack on Google Collab

- Jupyter Notebook stored on Google Drive which can be run on Google's virtual machines
  - Used because it is easier for less technical people to follow
  - Repurposed to run Bash, C and C++ code
- Comprehensive documentation on entire qCrypto stack, as well as points of consideration for future developers
- Shows you how to setup and run the stack, and then literally runs the code to set it up and execute on a linux virtual machine
- Other documentation (including differences from Dr Kurtsiefer's code) on the document itself

Main differences from Dr. Kurtsiefer's qCrypto

- Various (minor) changes to various components in the stack
  - Mainly refactoring (improving readability without affecting original functionality)
  - Added features which were useful in setting the stack up (more debug logs, ordering of bases in timestamp events on Bob's side, more input params on transferd.c to configure local port number etc)
  - Added documentation generation
- Error correction module refactored and restructured to support other algorithms
- More detailed version on the Google Collab document for those who are interested

03

Error Correction

Error Correction Procedure

1. Estimate Quantum Bit Error Rate (QBER) by taking a sample from the raw sifted key (that sample is later discarded)
2. Recall that the raw sifted key comprises of a sequence of binary states (now represented as bits)
3. Cascade algorithm: Divide raw sifted key into blocks, exchange parities for each block, and recursively binary search for flipped bits if the parities do not match
    a. Binary search as in if a parity for a block doesn't match, recursively split the block into two and repeat the parity checking procedure until the erroneous bit is found
    b. Iterate this a few times until satisfied that there should be no more errors
    c. Used in qCrypto's ecd2

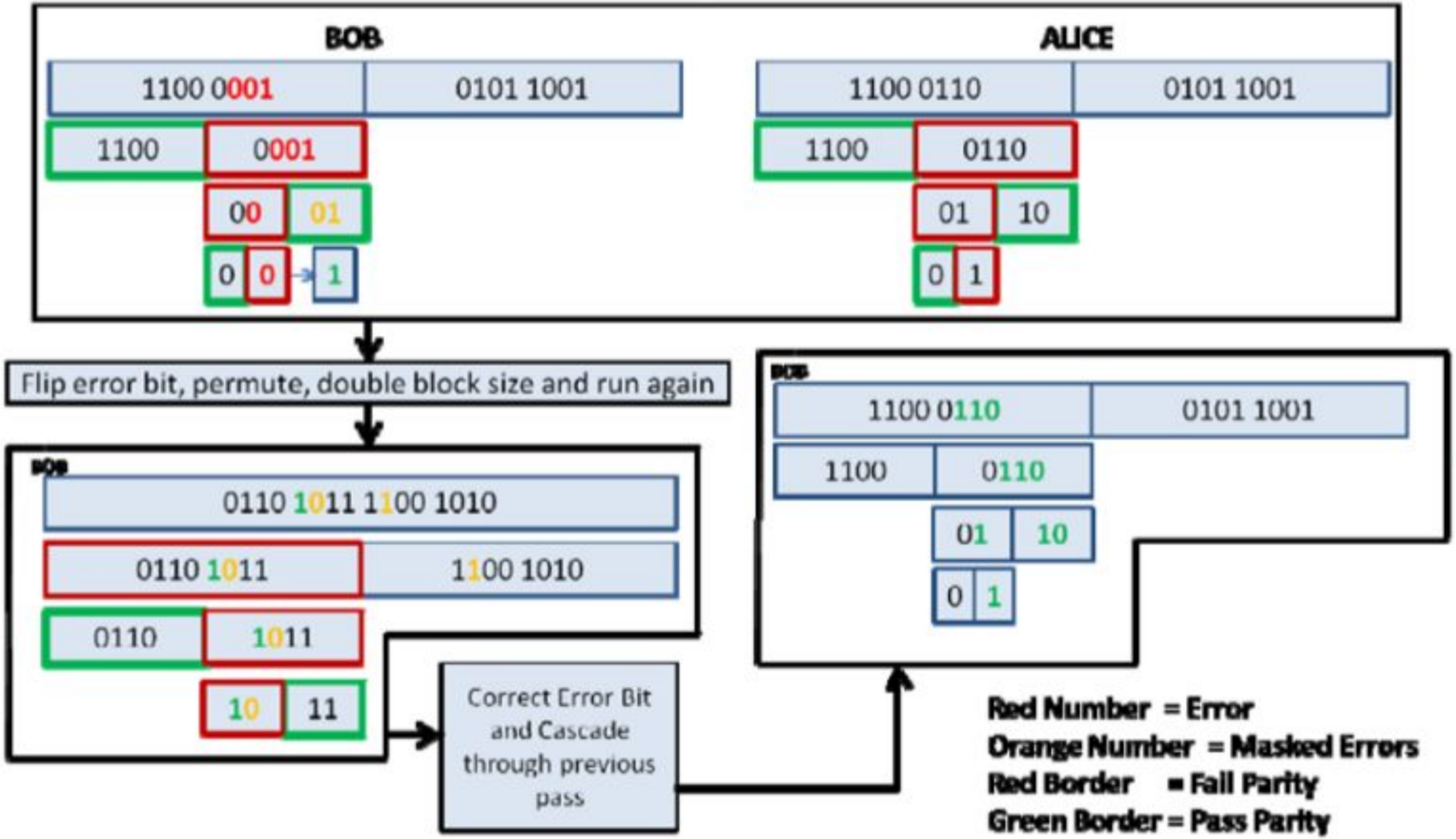


Figure 1. Binary Bisection and Cascade Operation

Privacy Amplification Procedure

1. Perform privacy amplification (matrix multiplication of the corrected key with a universal hash family: a random binary matrix will do)
   a. QBER is used as a parameter for binary entropy function which is then used to calculate length of final key (*L)*.
   b. Revealed bits in Cascade are either discarded during the algorithm or factored into the final key length calculations.
   c. Matrix multiplication: 1 x n corrected key times n x L random binary matrix

04

LDPC

Low Density Parity Check Codes (if the name sounds *complicated* means protocol is also *complicated*)

04

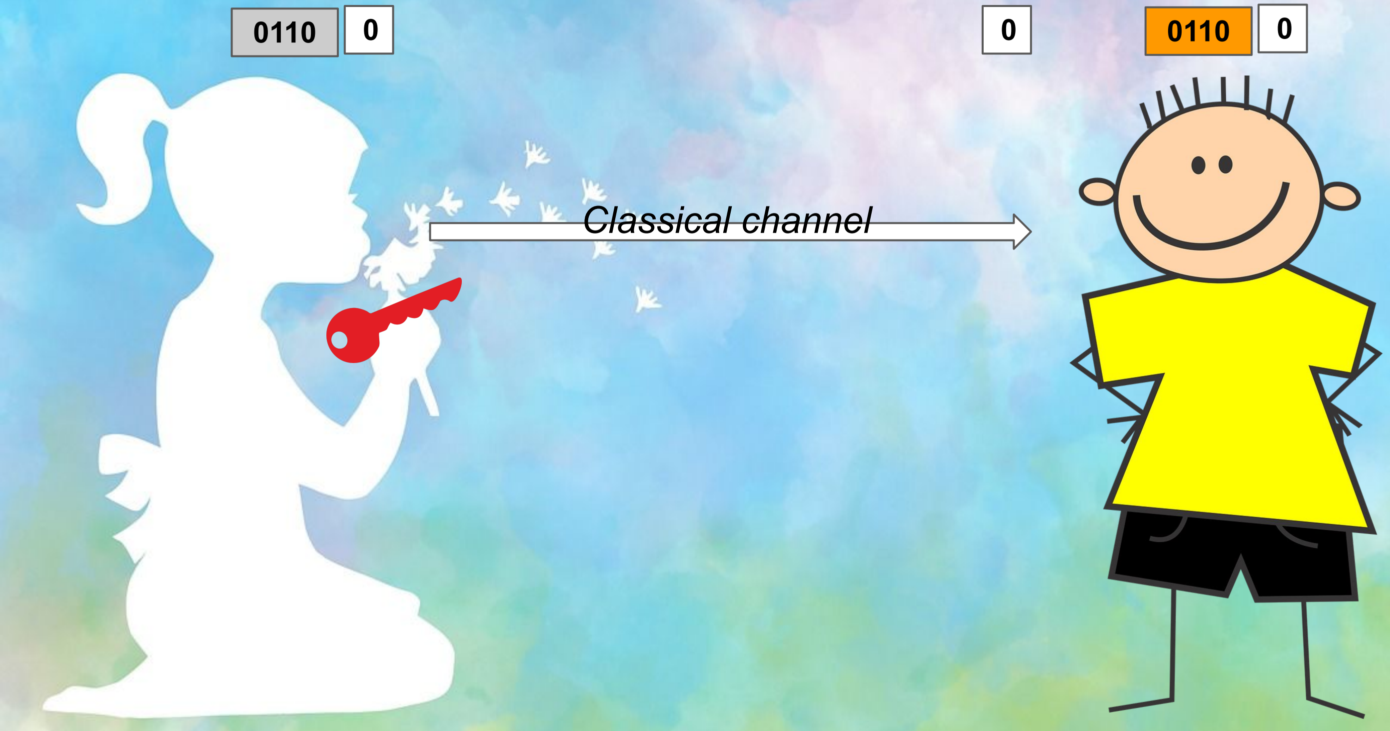
LDPC

Low Density **Parity** Check Codes (if the name sounds *complicated* means protocol is also *complicated*)

Imagine that you compare the parity of a block in the Cascade algorithm. How did you do it?

Alice adds every bit (student) in a block (group), and checks whether it is even or odd.

Alice tells Bob, who then checks if it’s the same parity on his side.



LDPC

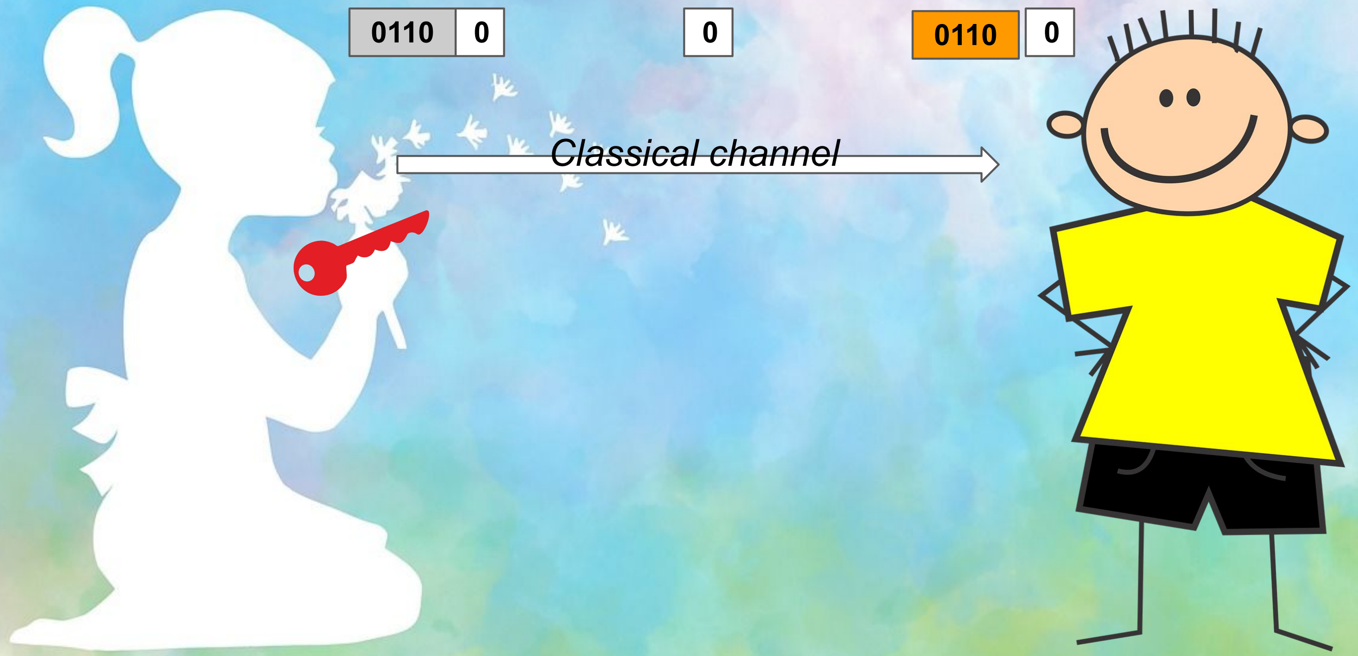
Error Correction

Now let's do the same thing differently.

Alice now tells Bob:

"I'll give you this number, 0 or 1. If you add it to the number of students in the group, that group will always have an even number."

"If it is not even, then there is an error in that group."



LDPC

Error Correction

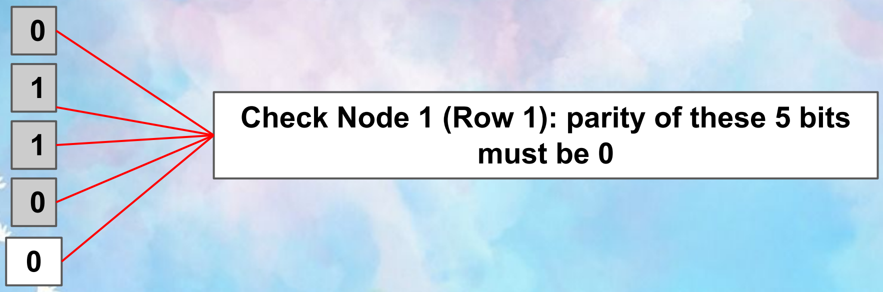
LDPC

Error Correction

If this is represented as a parity-check matrix, this is how it (in yellow) will look like:

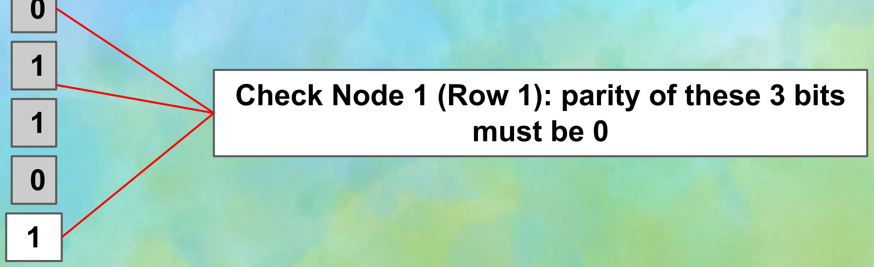
0110	0
1111	1

This matrix can also be represented as a Tanner Graph (bipartite graph): (the values substituted in to make it easier to understand). Each edge is a '1'



If we split the block in half and calculated the parity for the left sub-block:

0110	1
1100	1



If this action is represented as a parity-check matrix, this is how the **parity-check matrix** (in yellow) will look like:

0110	0
1111	1

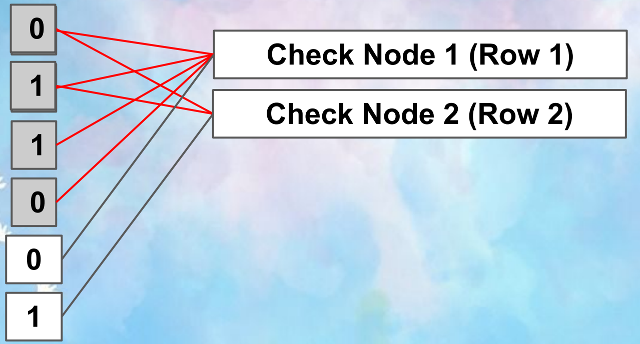
The idea is that your raw sifted key, with the parity bits concatenated behind, should give [0 0 0 0 … 0] when multiplied with the parity check matrix:

$$\begin{bmatrix} 1111 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = [0]$$

It’s just a slightly more complicated way of checking if the parity is correct.

If we combine both rows together:

0110	0	1
1111	1	0
1100	0	1



"**Low-density**": Few number of '1's

"**Parity Check**": Uses parities to "check" the message

"**Code**": Something you send that contains the message (raw sifted key in this scenario) and the parity bits

* for QKD you only send the parity bits

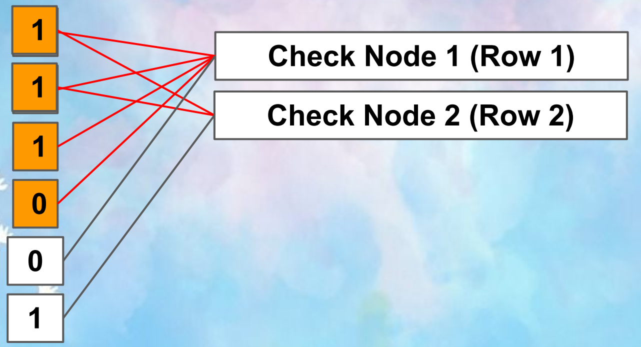
Alice & Bob establish the parity check matrix to use beforehand

LDPC

Error Correction

Alice will send Bob the parity bits only “01” and Bob will substitute the message portion with his own raw sifted key.

1110	0	1
1111	1	0
1100	0	1



Bob iteratively corrects the message bits using the constraint that the parity of the bits that each check node is connected to must be zero

May fail if too few parity bits are used, too few check nodes, check nodes have too high a degree etc

LDPC

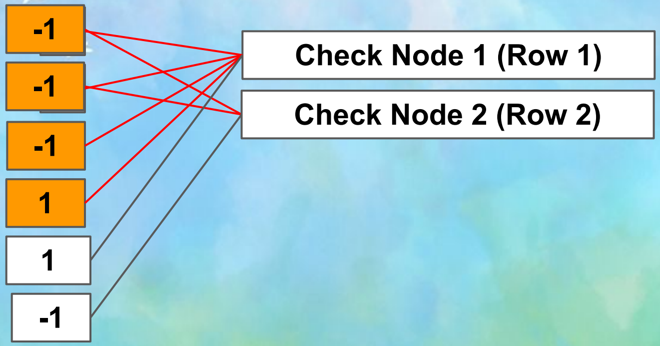
Error Correction

Let's do a simple decoding (Gallager) example with the same matrix. Alice computed parity bits "01" and sends the parity bits to Bob:

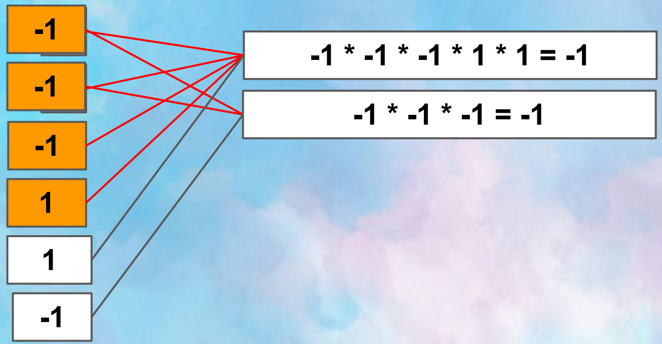
0110	0	1
1111	1	0
1100	0	1

Let's say Bob's raw sifted key bits are 1110

Bob passes the raw sifted key bits and the parity bits into the Tanner graph: For decoding purposes, '1' is converted to -1, '0' is converted to 1



Iteration 0:



LDPC

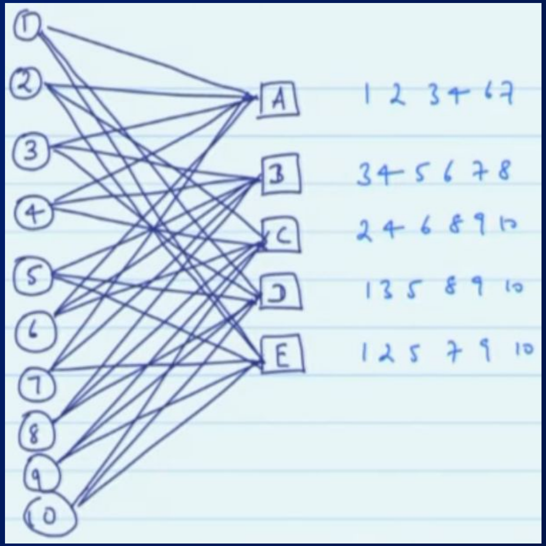
Error Correction

LDPC

1. Error correcting code (basically in qCrypto's ecd2 you swap Cascade for LDPC)
2. Both Alice and Bob establish a parity check matrix *H*
3. Alice calculates a block of parity bits *p* with *H*, such that when *p* is appended to her sifted key m_A, $[m_A\ p] \times H^T = 0$ (**encoding**)
   a. Alice sends *p* to Bob
   b. Bob appends *p* to his own sifted key m_B and using the relationship that $[m_A\ p] \times H^T = 0$, iteratively corrects his key using a decoding algorithm (**decoding**)
      i. Decoding can fail; often referred to as **Frame Error Rate (FER)**. Usually due to insufficient parity bits (this is due to the **code rate** of the LDPC code used, which is the ratio between message bits and parity bits)
      ii. Final decoded value checked with a CRC (checksum) sent by Alice to verify correctness

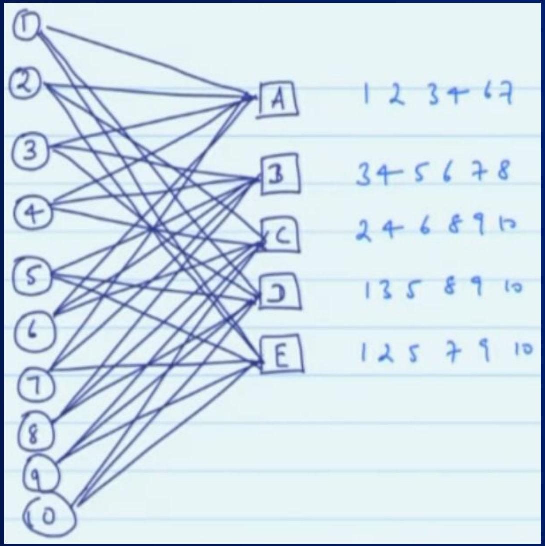
LDPC (Encoding)

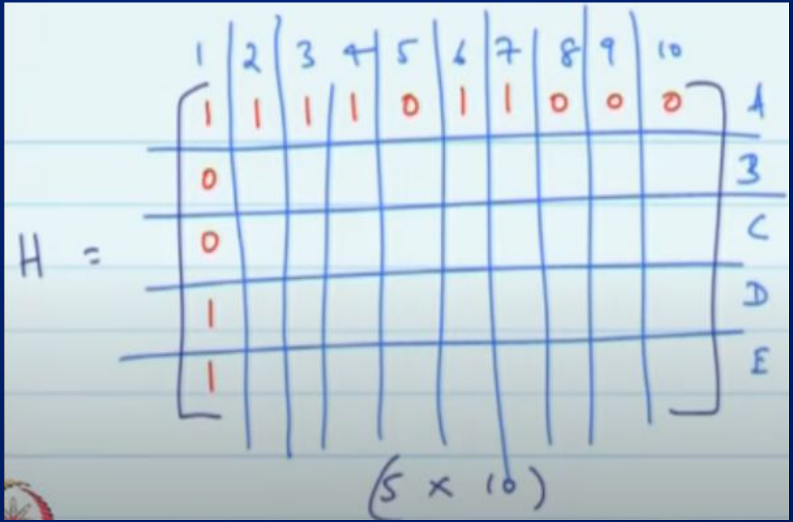
1. Error correcting code (basically in qCrypto's ecd2 you swap Cascade for LDPC)
2. Alice calculates a block of parity bits *p* using a parity check matrix *H*, such that when appended to her sifted key m_A, $[m_A\ p] \times H^T = 0$ (**encoding**)
3. H is a sparse binary matrix and can be represented as a Tanner Graph (bipartite graph):
4. Left nodes: "**Variable Nodes**". Each VN is a column in H
5. Right Nodes: "**Check Nodes**". Each CN is a row in H
6. An edge represents a '1' in $H_{\text{variable\_node\_i, check\_node\_j}}$



LDPC (Parity Check Matrix)

1. Left nodes: “**Variable Nodes**”. Each VN is a column in H
2. Right Nodes: “**Check Nodes**”. Each CN is a row in H
3. An edge represents a ‘1’ in $H_{variable\_node\_i,\ check\_node\_j}$





LDPC (Decoding)

1. Alice sends that block of parity bits to Bob
2. Bob appends *p* to his own sifted key m_B and using the relationship that $[m_A\ p] \times H^T = 0$, iteratively corrects his key using a decoding algorithm (**decoding**)
    a. Variables and check nodes are either directly or indirectly linked to each other, allowing the decoding algorithm to guess what is the most likely value for a particular variable node etc
    b. Decoding can fail; often referred to as **Frame Error Rate (FER)**. Usually due to insufficient parity bits (which are not sent to improve the **code rate**, which is the ratio between message bits and parity bits)
    c. Final decoded value checked with a CRC (checksum) sent by Alice to verify correctness

LDPC Privacy Amplification

1. Almost identical to Cascade (since this is LDPC operating on a binary symmetric channel (BSC))
2. Only difference is all the parities sent in one go
3. Privacy amplification algorithm factors the number of parity bits sent into the final key length
4. For LDPC, there is an additional trade-off: sending less parity bits improves reconciliation efficiency (leading to higher final key length), but can lead to higher frame error rate (e.g. more decoding failures)

Implementation

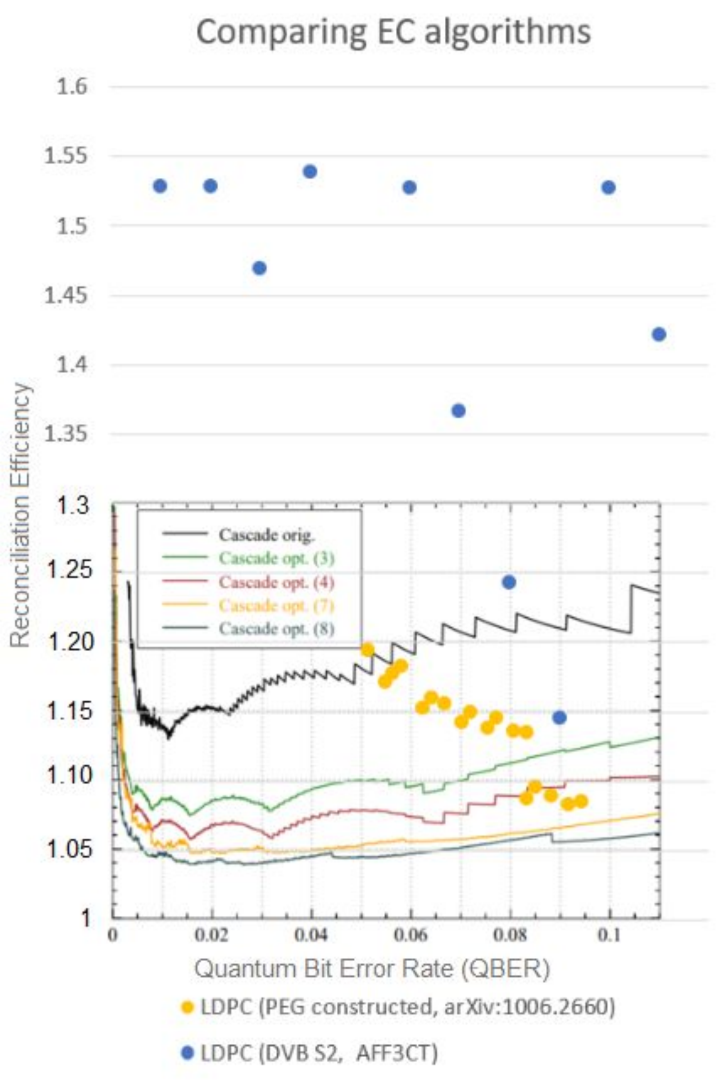
1. AFF3CT C++ library
    a. LDPC is complicated, implementation by myself was prone to error, likely not so efficient and lengthy in development time
    b. Use a library to do it instead
    c. Downside: Initially designed to be a simulator (since LDPC usually done on a very low level, even semiconductor level), so not really optimized (e.g. represents each bit as an 32-bit integer when encoding)
    d. No documentation on how to use it as a library, but I've already developed code on how to use it as a library so that is no longer an issue

Implementation

1. Main considerations:
    a. What matrix to use (and for what QBER)
        i. Rate-variable LDPC codes, where you can modify the parity bits sent, can reduce the number of matrices needed
        ii. Construct your own / use others?
    b. What decoding algorithm to use?
        i. Already tested and sorted out (now using belief propagation flooding sum-product algorithm), but may want to test other decoders if using good matrices
    c. QKD LDPC for the binary symmetric channel is different from QKD LDPC for the continuous channel (already noted in README_LDPC.md & factored into code)

LDPC Parity Check Matrix

1. Parity Check Matrices tested, ordered by secret key rate (ascending order):
   a. From the recently released 5G standard
      i. Code rate can be varied by not sending some parity bits (known as *puncturing*). In the 5G standard, parity bits are punctured starting from the last parity bit
      ii. Frame error rate too high, resulting in bad performance
   b. A few sample matrices from the Sparse Matrix Encyclopedia
      i. Matrices weren't able to support QBERs as well as DVB S2. Probably because they were quite dated
   c. From the DVB S2 standard (decent, but does not measure up to efficiencies quoted by papers on LDPC that constructed their own matrices)



Comparing Cascade with LDPC

1. **X-axis**: QBER, Y-axis: Reconciliation efficiency (the closer to 1, the better).
2. **Blue dots**: LDPC implementation using DVB S2 matrices. Not rate matched, so there is room for improvement. All at 0% FER
3. **Yellow dots**: LDPC with matrices constructed by improved PEG algorithm, arXiv:1006.2660v1 by David et al.. Data from the paper, all at presumably 0% FER
   a. Construction algorithm not released
4. **Jagged lines**: Cascade reconciliation efficiencies based on arXiv:1407.3257v2 by Jesus et al. Diagram from the paper.
5. **Omitted**: LDPC with matrices constructed by PSD-PEG algorithm from doi:10.1109/ACCESS.2017.2688701

05

Future Steps

Future Steps

1. Integrating LDPC into qCrypto
    a. Matrix used needs to be optimized (constructed using whatever algo.)
        i. Investigate matrix construction algorithms
            1. Optimizing degree distribution & girth of matrices
        ii. Look into varying the rate of DVB S2 matrices?
    b. Aff3ct library that I used will need to be optimized for use
        i. Will require some copying pasting, re-implementation of some sections
    c. LDPC protocol needs to be added into ecd2
        i. C++ can be integrated with C quite easily
        ii. Should be easy as ecd2 is better structured to support multiple algorithms

Thanks!

Do you have any questions?

Pictures from Pixabay
NES Zapper by Mike12

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.