Implementing and Testing the BB84 Quantum Cryptography Protocol in Cirq

Eriks Kristians Porietis
e.k.porietis@student.utwente.nl
University of Twente
Enschede, The Netherlands

ABSTRACT

This research investigates the feasibility of using Cirq, a Python framework, to implement the BB84 quantum key distribution protocol, using model-based testing. Additionally, this paper covers how such implementation performs under different noise models. Therefore, this research begins with by briefly recalling research on the BB84 protocol, model-based testing, Cirq and noise models. Next, formal models, adapter and the implementation used by TorXakis, a model-based testing tool, are described in detail. Then, experiment setup and results are covered, supplied by multiple graphs, visualizing test results. This research concludes with the implementation passing all the initial tests, validating the ability of Cirq to simulate BB84. Additionally, the implementation in Cirq can reliably handle depolarizing noise under 10% probability and amplitude dampening noise under 12% probability. However, phase dampening noise has minimal effect on the performance of the implementation. Finally, future work describes how, due to the limited scope of the study, additional research and testing on both model and implementation could be beneficial.

KEYWORDS

Model-based Testing, Cirq, Cryptography, Noise, Quantum Encryption, BB84

1 INTRODUCTION

Online communication is an unavoidable part of our daily lives, making it necessary to develop ways to secure and protect the exchanged information. A way to carry this out is to encrypt all messages between the parties using a key, known only to them. Therefore, even if the message is intercepted by a third party, it cannot be comprehended without directly knowing the secret key. The task becomes more complicated as the secret key must, in turn, be agreed upon and communicated without the third party knowing about it. Multiple algorithms were created to privately exchange secret keys, such as the Diffie-Hellman key e xchange [1] and RSA [2]. However, these methods rely on the computational difficulty of mathematical problems that can become obsolete with the advancement in quantum computing. Hence, quantum key distribution can be a promising solution for the future.

Quantum key distribution is a communication method that uses quantum mechanics combined with cryptographic protocols to ensure security. One such protocol is the BB84 developed by Charles Bennett and Gilles Brassard in 1984 [3]. The BB84 protocol is based on the principle of quantum signals mot being reproducible once measured. This property makes it theoretically possible to detect the presence of an eavesdropper, if one is attempting to intercept the exchange.

However, real-world implementations of BB84 can be affected by other factors, such as noise in the quantum channel [4]. These factors might compromise the correctness and robustness of the protocol in both normal and eavesdropper scenarios. Thus, finding ways to simulate and reaffirm such protocols before they are implemented in real world is essential. This can validate if the protocol is indeed correct, as well as determine the levels of noise under which it can retain its security property.

Therefore, this research focuses on testing and validating whether Cirq [5], a Python-based quantum computing framework developed by Google, can be used to simulate the BB84 protocol. Additionally, this research also finds how different quantum noise affect the security of the protocol.

2 RESEARCH QUESTION

The objective of the research can be summarized into the following research question: To what extent can the Cirq library effectively implement the BB84 quantum key distribution protocol with and without an eavesdropper and how do different noise models affect its Quantum Bit Error Rate.

To address the second part the research question comprehensively, the study focuses on three sub questions related to the noise models.

- How does the BB84 implementation perform under *Depolarizing Noise.*
- How does the BB84 implementation perform under *Phase Damping Noise.*
- How does the BB84 implementation perform under *Amplitude Dampening Noise.*

3 RELATED WORK

This sections analyses different works related to the research. Various studies have assessed the security and correctness of the BB84 protocol implemented in different environments.

Khaleel and Tawfeeq [6] conducted a real-time performance test of a BB84-based quantum cryptographic system. They used polarized laser setup with a receiver to simulate a quantum channel. The work analyzed the influence of hardware parameters, such as avalanche photo-diodes, laser diode power, and temperature, on the Quantum Bit Error Rate of the system.

Another physical implementation of the BB84 protocol was done by Molotkov [7]. To make it practically applicable, a simpler fiber-optic implementation without phase modulators or polarization adjustments was created. Molotkov tries to address practical challenges like dark counts and quantum efficiency.

The paper by Guitouni et al. [8] explored the BB84 protocol in the context of IoT networks. Their research focused on the practical deployment of BB84. By evaluating the protocol's feasibility in resource-constrained IoT environments, they identified critical metrics like entropy variation to assess its adaptability to IoT-specific challenges.

These studies provide valuable empirical data on BB84 performance in hardware systems, while also being vastly different from this research. Their focus is on the hardware implementation, while this research focuses on the quantum channel simulation in Cirq. Additionally, it uses probabilistic model-based testing to test for correctness and robustness of the system.

The paper by Watanabe [9] focuses on improving the BB84 protocol's noise tolerance using random privacy amplification. It demonstrates that tolerable error rates can increase from 7.5% to 11%. By providing a theoretical foundation for noise resilience, it can directly complement noise impact on the BB84 implementation tested in this research. However, it is purely theoretical and does not test implementations under realistic noise models.

Lee et al. [10] propose advanced eavesdropping detection algorithms for the BB84 protocol. Their introduction of the grouped BB84 protocol and combinatorial detection algorithms allows accurate eavesdropping detection despite noise and other variables. Lee uses a statistical and simulation-based analysis, making it a theoretical extension rather than a physical implementation. This makes it different from this research, as it tries it test a Cirq BB84 implementation, rather than just a theoretical model.

In their paper Elboukhari et al. [11] explore PRISM probabilistic model checker to analyze the security of BB84. They created a probabilistic state model of the BB84 specification in PRISM and verified if it retained the security properties of the algorithm, including under noisy conditions. They concluded that the probability of detecting an eavesdropper increases with photon count and channel noise.

While this work, being the closest to this research, contributes to the evaluation of the BB84 algorithms, their approach and focus significantly differ. Elboukhari et al used a verification-driven approach, using model checking to theoretically verify the security properties under controlled conditions. Their work primarily focused on the PRISM model of the BB84 algorithm and provided limited exploration of specific noise models.

In contrast, this research focuses on model-based testing to validate an implementation of the BB84 in Cirq, under different noise models, including depolarizing, phase dampening, and amplitude damping. Therefore, this research attempts to reproduce the results of previous papers, while also exploring a novel aspect of a BB84 being implemented and tested in Cirq.

array multirow graphicx

4 METHODOLOGY

This section describes key concepts used in this research, such as the BB84 protocol, Cirq and model-based testing. Lastly, research methodology is mentioned, summarizing the implementation and experiment process.

4.1 BB84

The BB84 encryption protocol was developed by Charles Bennett and Gilles Brassard in 1984. In it, the sender generates a random key and passes it, by using photons for individual bits, through a fiber optic channel. By using polarizing filters, each individual photon takes an orientation, which the receiver then has to guess. In the end, the sender and receiver compare their used orientations, or bases, and the bits with the matching ones become the actual key. The security of it becomes clear, if a malicious eavesdropper decides to intercept the photons in the middle. If an eavesdropper intercepts the photons, they must guess the correct measurement basis. Incorrect guesses disturb the photon states, introducing detectable errors when the sender and receiver compare their keys.

Table 1 depicts a simplified version of the BB84 protocol with only a sender and a receiver. It uses + and X as the two type of basis. *Sender polarization* being a combination of the sender bit value and its basis type. If the receiver matches the sender basis - the bit is received correctly. As the secret key is formed only from the matched basis between the sender and receiver, only those are shown.

In order to determine, if the exchange was secure and was not compromised, the Quantum Bit Error Rate(QBER) can be used, which is a number of total bits matched with matched basis divided by total number of matched basis.

$$\frac{\text{Matched Bits with matched basis}}{\text{Matched Basis}} = \text{QBER}$$

If the resulting QBER is higher than a theoretical security threshold of 11% [12], the channel can be considered to be compromised.

4.2 Cirq

Cirq [5] is an open source Python software library for writing, manipulating, optimizing quantum circuits, and for executing them on both quantum computers and simulators. Its core functionality includes simulators for testing small circuits with all the operations. Cirq's operations are built around the concept of *Gates*, which represent transformations that can be applied to a *Qubit*, an abstract circuit object representing a unit of quantum information.

These tools can be used to effectively simulate the behavior of the BB84 encryption protocol, by modifying the *Qubit* state as required and running simulations to observe the results.

4.3 Model-Based Testing

In order to test if Cirq can be used to simulate BB84 behavior, Model-Based testing (MBT) was chosen for its ability to test a system against a specification. With MBT a System Under Test (SUT) is tested against an abstract model of its required behavior. The Model describes how the system should behave, and what it should and should not do, that is, the behavior of the SUT shall conform to the behavior prescribed in the model. The model itself is assumed to be

Sender Bit	0	1	1	0	1	0
Sender Basis	+	+	X	X	+	X
Sender Polarization	Horizontal ($\vert\rangle$)	Vertical ($\vert\uparrow\rangle$)	Diagonal ($\vert/\rangle$)	Anti-Diagonal ($\vert\backslash\rangle$)	Vertical ($\vert\uparrow\rangle$)	Anti-Diagonal ($\vert\backslash\rangle$)
Receiver Basis	+	X	+	X	+	X
Secret Key	0	-	-	0	1	0

Table 1: Simplified BB84 Protocol Example

correct and valid; this assumption is fundamental, as the accuracy of the tests relies entirely on the correctness of the model [13].

The main advantage of MBT is its ability to automate test creation and execution, in contrast to the traditional manual way of testing a system. This allows for testing of more varied, longer, and more diversified test cases with less effort. However, due to the inherent limitations of testing, such as the limited number of tests that can be performed in a reasonable time. As Edsger W. Dijkstra observed, "Program testing can be used to show the presence of bugs, but never to show their absence!" [14].

There are different kinds of testing, and thus of model-based testing. It depends on the kind of models being used, the quality characteristics being tested, the level of formality involved, the degree of accessibility and observability of the system being tested, and the kind of system being tested. Because the goal is to validate the ability of a program to conform to an algorithm or a specification, black-box testing was used. Black-box as a testing methodology implies that the inner-workings of the SUT are unknown to the tester and the focus is solely on observing the system's behavior and outputs in response to external inputs or interactions. This goes in contrast with white-box testing, which involves designing and executing tests based on the internal structure, logic, or code of the system.

In this research, to formally define a model, as well as to automatically generate and execute tests against the SUT, the TorXakis [15] MBT tool was used. TorXakis is an academic, research tool that is being developed. It implements the ioco-test generation algorithm for symbolic transition systems, and it uses a process-algebraic modelling language Txs inspired by the language LOTOS [16]. The ioco (Input-Output Conformance) relation is a formal testing theory used to verify whether the behavior of SUT conforms to the expected behavior of a model. At its core, it ensures that the SUT produces outputs that are consistent with the model's specifications when given specified inputs and does not produce any outputs that the model does not allow. Usually ioco is non-deterministic, meaning the specification allows for multiple acceptable outputs for one input, however it is not relevant for this research.

4.4 Research Methodology

In order to research the extent to which the Cirq library can effectively implement the BB84 quantum key distribution protocol under ideal, eavesdropped and noisy conditions, several key steps had been taken.

First, two formal models were created using TorXakis model definition language in accordance with the literature definition of the BB84 protocol. One model defining the specification for BB84 behavior under ideal and eavesdropped conditions, testing the correct sender (Alice) and receiver (Bob) outputs. The other, testing for noise impact, only confirming if the resulting QBER conforms to the security threshold of 11%.

Second, a SUT was created that, in theory, implements the behavior of the BB84 protocol using Cirq circuits, gates, qubits, and simulators. To make the SUT compatible with TorXakis, an adapter was made, which would pass the data from the model to the SUT and back. The adapter works as both the synchronization layer as well as a converter, enabling the SUT to be compared against the model specification.

Finally, multiple tests were executed, using both models, in accordance with the experimental setup. The noise results, including the QBER values, were visualized in a graph to improve clarity.

5 IMPLEMENTATION

We briefly highlight crucial parts of our implementation, such as the TorXakis model, the implementation of BB84 in Cirq itself and the adapter.

5.1 Model

To test whether the SUT correctly implements the BB84 protocol, first a formal model is needed specifying the algorithm. To do that 2 formal models were created [17] in txs, TorXakis' formal model definition language. One model testing for normal behaviour of BB84 and the other was slightly adjusted see the impact of noise on the BB84 algorithm.

The models consist of several logic blocks: type definition, function definition, channel definitions, model definition, process definition and connection definition.

Txs has default types such as Int, String or Bool. However, in order to describe the flow of information from Alice to (possibly) Eve and Bob, bit and basis types were defined.

```
    TYPEDEF Bit ::= Zero | One ENDDEF
    TYPEDEF Basis ::= Rectilinear |
                      Diagonal
    ENDDEF
```

To combine them, for ease of use another type was created.

```
    TYPEDEF Qubit ::= Qubit {bit :: Bit;
            basis :: Basis}
    ENDDEF
```

When creating compound types, TorXakis automatically creates access functions to access the *head* and *tail* of such objects.

For the model to retain information of what qubits were present before, a list type was created. Where *head* contains a qubit object and *tail* links to another list object, which if empty, ends the list.

```
    TYPEDEF List_Qubit ::=
             CNil_Qubit
         |  Cstr_Qubit { head :: Qubit;
```

```
            tail :: List_Qubit }
    ENDDEF
```

Several functions have been defined that are mainly used to manage and transform data. For example, *QList_Append* is used to add a new qubit to an existing list object.

```
    FUNCDEF QList_Append
        ( x :: List_Qubit;
        q :: Qubit ) :: List_Qubit ::=
        ...
    ENDDEF
```

The others are used for different output calculations and transformations.

In the context of TorXakis, channels are the primary means through which components of the system interact and exchange data. Channels provide an abstraction that allows input and output to the model facilitating the flow of information. In txs, a channel has to be defined inside a *CHANDEF* block with a type associated with it to ensure consistency.

```
    CHANDEF CommunicationChannels ::=
        InputAliceQubit :: Qubit;
        QBERResult,
        AliceConf, BobQConf, BobBConf,
        EveQConf, EveBConf,
        OutputAliceTrigger,
        OutputBobTrigger :: Bool;
        InputBobBasis,
        InputEveBasis :: Basis;
        InputEveBit, InputBobBit :: Bit;
        OutputAlice, OutputBob :: String;
    ENDDEF
```

The Model itself is defined inside the *MODELDEF* block. This block encapsulates the entire model and specifies its channels and behaviour. The channels that are used for data input are placed in the *CHAN IN* section, while the *CHAN OUT* section is used for the model data output channels. The *BEHAVIOUR* section specifies the behaviour of the model when it is initiated. In both models, this behavior begins by invoking the Alice[]() process.

The logic of the model is divided into separate processes: Alice, Eve and Bob. A *process* is a logic component of the model that specifies interactions with channels and other logic components. In TorXakis, a *process* has specified channels and local variables. Channels are specified inside square brackers *[]* and are used for outside communication, while variables are specified inside *()* and are used for local calculations and decisions.

Alice specifies the logic of the sender which should prepare a bit and its basis and send along a communication channel to Bob. First, Alice has a local variable n which specifies the number of cycles the model will take, similar to bits in a the secret key. Whenever a bit reaches Bob, one cycle passes and n gets decremented by 1, until it reaches 0. If n is 0 when Alice is called, final information is sent, based on what model it is and process is finished. Otherwise, Alice uses *InputAliceQubit* channel to prepare a qubit (bit and basis) to send.

In TorXakis, a model can request an input from a channel which prompts TorXakis to generate data within specified data type restrictions. Therefore the following randomly creates a qubit *x* that can have *bit* value *One* or *Zero* and basis value *Rectilinear* or *Diagonal*.

```
    InputAliceQubit ? x
```

Additionally, sending an output through a channel works in a similar way, with a difference of having to provide the exact data to send.

```
    AliceConf ! True
```

With a generated qubit, Alice, based on a local variable *eve_flag* decides on whether to send it to Eve or to Bob. Then after after updating the the sent qubit list with *QList_Append*, Alice calls the the next process - Eve if the flag is True and Bob otherwise.

If Eve is present, it generates a basis through *InputEveBasis*. If the generated basis matches with the basis of the qubit sent by Alice, Eve sends the same qubit along to Bob. Hovewer, if the basis did not match, Eve additionally generates a bit through *InputEveBit*, mimicking measuring a qubit under an incorrect basis and getting a random bit, in the result. In this case Bob receives a qubit with the randomly generated but value and Eve's measurement basis.

Bob's process work in a similar way to Eve. It first generates basis with *InputBobBasis*. Then compares the incoming qubit basis to his generated one, if it matches, Bob updates his received qubit list with his generated basis and the received bit value. If the basis did not match, Bob generates a random bit value with *InputBobBit* and updates the received qubit list using it. In the end, Bob calls Alice again, but with a decremented n by 1, ending one cycle.

The key difference between the 2 models is the final output. The first model was made to test the general behaviour of the BB84 algorithm, therefore at the end, when n reaches 0. Alice outputs all qubits that were sent by alice, where *QList_to_Str* function is used to transform a list into String where bits can be 0 or 1 and basis *C* and *H*.

```
    OutputAlice ! QList_to_Str(qa)
```

The same is done with bits received by Bob:

```
    OutputBob ! QList_to_Str(ql)
```

The other model tests if the QBER of the SUT surpasses the security threshold. This is calculated based on the Quantum Bit Error Rate, which is a number of total bits matched with matched basis divided by total number of matched basis. The resulting QBER is then compared to 11%, which would indicate if the channel is compromised or not. The actual output is True if the QBER is bellow 11% and False if above.

```
    QBERResult ! Compare_QBER(bitNum, baseNum)
```

Finally, the communication with an SUT or an adapter is defined in the *CNECTDEF* block. The communication is done through sockets, enabling TorXakis to send or receive data as defined by the model. *CHAN OUT* defines channels which data, generated by TorXakis, needs to be sent to the SUT. *CHAN IN* is the opposite, where it is the information that is received from SUT and is checked against what is expected by the model. Each such channel must specify a local port over which the information will be sent over and the model channel that the information will be taken from. One port can be used for one in and one out channel.

```
    CHAN  OUT  InputAliceQubit
    HOST "localhost"  PORT 7891
    ENCODE     InputAliceQubit ? qA
    -> ! toString(qA)

    CHAN  IN   AliceConf
    HOST "localhost"  PORT 7891
    DECODE     AliceConf !
    fromString(s)  <-   ? s
```

Since TorXakis' SUT output await time is limited, it is essential to manage input-output logic carefully. To synchronize the model with an adapter, sync and trigger channels have been introduced, making the testing less prone to synchronization errors.

5.2 System Under Test

The SUT is a BB84 algorithm implemented in Python using the Cirq library [17]. It was coded specifically for this research, to find if it is possible to use Cirq circuit to simulate full BB84 functionality under normal, eavesdropped and noisy conditions.

In Cirq, a *circuit* is a structured representation of quantum operations applied to qubits over discrete moments in time. It uses gates and measurements to define behavior of the system. Gates are operations that modify the state of qubits, while measurements read out the state of qubits in a specified basis, collapsing their quantum state into classical information. In this SUT, 2 gates are used: X and H gates. The *cirq.X* gate applies a Pauli-X (NOT) gate, flipping the qubit state. The *Cirq.H* gate applies a Hadamard gate, putting the qubit into a superposition. To read a state of qubit, measurements are used, collapsing their quantum state into classical information.

```
    circuit = cirq.Circuit()
    circuit.append(cirq.X(qubits[0]))
    circuit.append(cirq.H(qubits[0]))
    circuit.append(cirq.measure(qubits[0],
    key="eve_measure"))
    result = cirq.Simulator().run(circuit)
```

To simulate BB84 using gates, both gates would be applied according to the bit and basis values chosen. The *cirq.X* gate is applied if the bit value is 1; for a bit value of 0, no gate is applied, leaving the qubit unchanged. Similarly, the *cirq.H* gate is applied only when the diagonal basis is selected. This way, if Alice sends a qubit in one basis and it is being measured in another, only one *cirq.H* gate will be present in the circuit. As a result, the qubit collapses randomly into bit value 0 or 1 during measurement.

Similar to the model, the SUT contains repeated cycles involving Alice, Eve, and Bob. Alice prepares the circuit according to the original qubit. Eve, if present, measures the circuit and prepares a new one based on her measurement results. Like Eve, Bob measures the circuit and saves the results for later. These cycles are repeated for the number of bits set in the key.

Normally, all qubit bit and basis values for Alice, Eve and Bob would be chosen randomly. However, this would prevent comparing the SUT to the model specification. Therefore, a decision was made to treat all these values, as user inputs, which would be taken from the model and converted by the adapter. Furthermore, as one of the models tests for normal behavior of the protocol, it requires the qubits sent by Alice and received by Bob to match the model specification. This makes it necessary to change the way the SUT handles the measurement state collapse when the basis do not match. For this, two SUTs were created: one for testing the impact of noise on BB84 performance (which retains the random measurement collapse), and the other for testing the core BB84 behavior. The second SUT was modified to treat the random measurement outcome also as a user input, which would be passed from the model.

```
    if eve_b != alice_b:
        last_state = eve_state
        circuit.append(cirq.H(qubits[0]))
        if eve_state != alice_s:
            circuit.append(cirq.X(
            qubits[0]))
```

Finally, Alice's and Bob's bit and basis values, along with the calculated QBER, are printed. The adapter reads this information and passes it to the TorXakis model for comparison with the expected result.

5.3 Adapter

The adapter serves as an intermediary layer between the TorXakis model and the SUT. The primary function of the adapter is to convert the signals and data given by the TorXakis model for the SUT. As well as transform the SUT outputs into correct data forms to send back to the model for comparison with the expected outputs. In this case, the adapter is a Python application, that uses sockets to connect to the TorXakis model and starts the SUT as a sub-process [17]. Threading is used to ensure seamless and synchronous communication between TorXakis, adapter and the SUT. Additionally, the adapter also collects execution data from the SUT, including the QBER, which is later analyzed for statistical purposes.

5.4 Noise Models

To test the performance of the BB84 implementation under noisy conditions, Cirq noise models were used. In total three different noise models were applied to simulate realistic quantum system behavior under imperfect conditions.

- Amplitude Damping Noise - this noise describes the dissipation of energy in the quantum system. This causes the qubit to return from the excited state to the ground state [18].
- Phase Damping Noise - this noise describes the loss of quantum information in the system. Which causes the qubit to fall back from the superposition state to the collapsed state [19].
- Depolarizing Noise - representing errors that occur when a qubit's state is replaced with a completely mixed state. It generalizes random disturbances in the qubit's state, affecting both amplitude and phase [20].

In order to apply the noise to the circuit, existing Cirq noise models were used. `cirq.depolarize(`ρ`: float)` is used for Depolarizing Noise, `cirq.phase_damp(`γ`: float)` for Phase Damping Noise and `cirq.amplitude_damp(`γ`: float)` for Amplitude Damping Noise. The ρ and γ values represent the chance with which the noise will be applied. Additionally, when testing for noise, the the simulator was replaced with another, to be able to handle noise simulations.

```
    simulator = cirq.DensityMatrixSimulator()
```

6 EXPERIMENTAL SETUP

The testing was divided into 2 parts: testing the BB84 implementation under ideal and eavesdropped conditions, and testing its performance under different noise models. For all the tests, the total number of bits transferred per test was set to `n = 1000`, in order to minimize the effect of random outliers on the results.

First, it is important to verify whether the BB84 implementation in Cirq conforms to a the standard, non-eavesdropped model. Then, the eavesdropper was enabled both in the model and in the SUT to test whether the model handles an eavesdropper appropriately. Both tests (ideal and eavesdropped conditions) were repeated 10 times for consistency.

Second, after testing behavior of the SUT under ideal and eavesdropped conditions, noise was introduced. For this, the other model was used to compare the QBER to the 11% threshold. Both the model and the SUT were setup without an eavesdropper. Noise was applied by adding `cirq.depolarize(ρ: float)`, `cirq.phase_damp(γ: float)` and `cirq.amplitude_damp(γ: float)` to the SUT and changing the simulator accordingly. Each noise model was tested individually. The error probability parameters (ρ and γ) were varied from 0% to 20%, in increments of 2%. Each ρ and γ value was tested 5 times and the average was taken to reduce the inherent randomness of the system. Additionally, if any of the 5 tests for a given ρ and γ value resulted in a QBER exceeding the 11%, this result would be noted alongside the average.

Finally, all the resulting noise data would be collected and visualized in a graph, representing how much noise can the Cirq BB84 implementation can tolerate.

7 RESULTS

Following the procedure described in section 6, the results gathered during model-based testing were documented and visualized in two graphs. First, tests conducted in the initial phase. These tests validated whether the Cirq implementation of the BB84 protocol adheres to the specified behavior in both ideal (non-eavesdropped) and eavesdropped scenarios. Figures 1 and 2 illustrate the impact of different noise models on the QBER of the SUT without an eavesdropper. Figure 1, however, depicts the average documented result for each noise probability. On the other hand, Figure 2 highlights the highest QBER result for each noise probability.

For the initial phase, 10 tests were conducted with and without an eavesdropper. In both cases, all the tests passed.

Figure 1 contains five plots, each representing the average QBER value of the five results taken for each noise probability. The color scheme is as follows: orange for phase dampening noise, green for amplitude dampening noise, and dark blue for depolarizing noise. Additionally, the 11% constant security threshold and the identity plots were added for visualization, colored light blue and purple, respectively.

Both depolarizing and amplitude dampening noises closely follow each other, with depolarizing noise generally making slightly larger impact on the QBER. Both plots exceed the 11% threshold at probability of 0.14. However, depolarizing noise reaching QBER of 10.976, without surpassing the threshold. Both depolarizing and amplitude dampening noise plots are linear, with a slight downward deviation from the identity line.

On the other hand phase dampening noise is distinctly different from the other noise models. It still retains a linear nature, but never reaches the 11% security threshold, with the highest average value being 2.899% at 0.16 noise probability.

Figure 2 closely resembles Figure 1, with the key difference being that the QBER data in this figure represents the highest value of the five results taken for each noise probability, rather than the average. All the plots maintain their linear nature, with slightly higher values. Depolarizing noise surpasses the security threshold at noise probability of 0.1, reaching QBER of 11.594%. Amplitude dampening noise approaches close to the 11% threshold at 0.12 mark, with a QBER value of 10.806%. Phase dampening noise, as in Figure 1, never reaches 11%, with the highest value of 4.374 at 0.16 noise probability.

8 CONCLUSIONS

The primary objective of this research was to validate whether the Cirq library can be used to implement BB84 protocol using its quantum channel. Additionally, this research aimed to evaluated the performance of the BB84 Cirq implementation under three different noise models - depolarizing noise, phase damping noise and amplitude dampening noise.

To achieve these objectives, Model-Based Testing was used to test if the implementation correctly follows the BB84 protocol specification. Two TorXakis specification models were created, along with the actual Cirq BB84 implementation that underwent testing.

The findings from the executed tests, as shown in Figure **??**, demonstrate that the Cirq implementation of BB84 performs well in ideal and non-eavesdropped conditions. All 20 tests passing with no issue, which, assuming the model is correct, confirms that the Cirq library can successfully implement the BB84 quantum key distribution protocol.

Results gathered after executing test under different noise models, depicted in Figure 1 and Figure 2, show that the performance is significantly affected by noise beyond certain thresholds.

- Depolarizing Noise: Under the depolarizing noise model, the Cirq BB84 implementation was secure until the noise level reached 10%, indicating a potential vulnerability.
- Amplitude Dampening Noise: Under the amplitude dampening model, the Cirq BB84 implementation was secure until the noise level reached 12%, indicating a potential vulnerability.
- Phase Dampening Noise: Phase dampening noise showed little impact on the QBER, with highest value only reaching 4.374% at noise probability of 16%.

The results indicate that Cirq's BB84 implementation correctly behaves in ideal and eavesdropped conditions, making it complete, but its performance is significantly influenced by the type and level of noise introduced. Specifically, the implementation in Cirq can reliably handle depolarizing noise under 10% probability and amplitude dampening noise under 12% probability. However, phase dampening noise has minimal effect on the performance of the implementation.

Additionally, this research can be considered a case study of model-based testing, as it was the primary tool used to verify an implementation of an algorithm. By successfully comparing the Cirq implementation against the specification model, which was assumed to be correct, we concluded that model-based testing was the proper tool in this context and can be used for the future research.

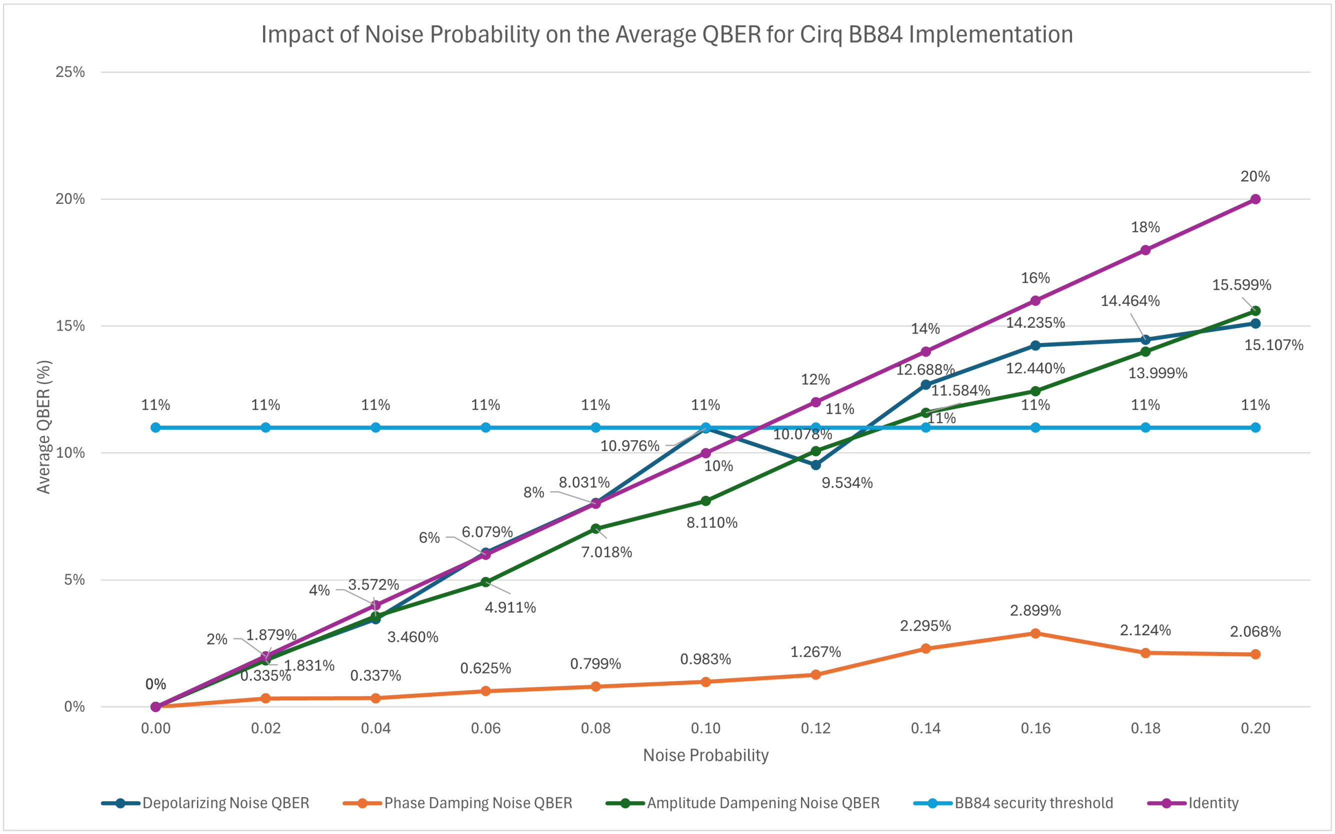

Figure 1: Noise test results with average QBER

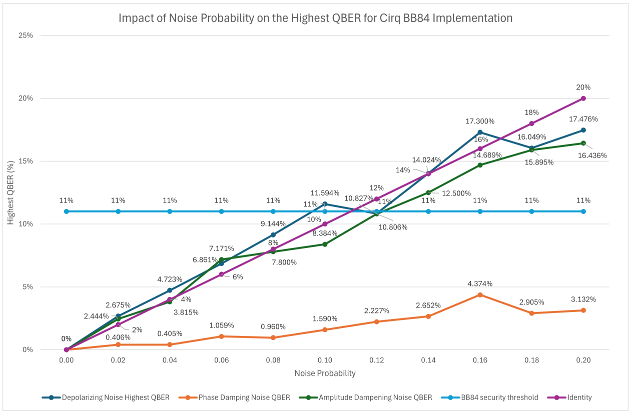

Figure 2: Noise test results with highest QBER

9 FUTURE WORK

The scope of this research was rather limited. Thus, future work can be done to build on this foundation in several key areas:

First, while the implementations created in this research were intended to be correct, future research could explore the results of testing faulty implementations that deviate from the intended BB84 specification. Such tests can be used to assess the soundness, proving that upon an incorrect implementation, the testing would fail, as intended.

Second, this research is based on the assumption that the specification models created in TorXakis correctly describes the BB84 protocol. This should be further tested and validated, in order to enhance confidence in the current research and ensure the reliability of the findings.

Third, the conclusions cover the quantum noise level at which the implementation fails and surpasses the security threshold. This can be expanded upon by doing further research to determine what quantum noise levels are realistic in practical quantum communication systems. Additionally, an aspect worth exploring, is testing for multiple noise models applied simultaneously, which can better simulate real-world conditions.

Finally, as the scope of this research excluded testing the probabilistic properties of BB84, future research can be done to include probabilistic model-based testing on an implementation of BB84 in Cirq.

REFERENCES

[1] N. Li, "Research on diffie-hellman key exchange protocol," in *2010 2nd International Conference on Computer Engineering and Technology*, vol. 4, pp. V4–634–V4–637. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5485276?casa_token=bSVjF88qWaAAAAAA:jwFmdzNZx461sXphVxL-eGtbWjqd1QX5570U39pcnPYYoPLacUwxjwnaq5n-b0FAqvLMdg8
[2] M. Shand and J. Vuillemin, "Fast implementations of RSA cryptography," in *Proceedings of IEEE 11th Symposium on Computer Arithmetic*, pp. 252–259.

[Online]. Available: https://ieeexplore.ieee.org/abstract/document/378085

[3] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," vol. 560, pp. 7–11. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397514004241

[4] G. Smith, "Quantum channel capacities," in *2010 IEEE Information Theory Workshop*, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5592851?casa_token=aXDcMz6drr0AAAAA:aNSB6mA860ALh2594U7Bw6z_AJVqM_X-u6IZyNGIUm_QFWkxWfZZPwCEyBn3B2g69d21wB0

[5] S. V. Isakov, D. Kafri, O. Martin, C. V. Heidweiller, W. Mruczkiewicz, M. P. Harrigan, N. C. Rubin, R. Thomson, M. Broughton, K. Kissell, E. Peters, E. Gustafson, A. C. Y. Li, H. Lamm, G. Perdue, A. K. Ho, D. Strain, and S. Boixo, "Simulations of quantum circuits with approximate noise using qsim and cirq." [Online]. Available: http://arxiv.org/abs/2111.02396

[6] A. I. Khaleel and S. K. Tawfeeq, "Real time quantum bit error rate performance test for a quantum cryptography system based on BB84 protocol."

[7] S. N. Molotkov, "On the secrecy of a simple and effective implementation of BB84 quantum cryptography protocol," vol. 16, no. 7, p. 075203, publisher: IOP Publishing. [Online]. Available: https://dx.doi.org/10.1088/1612-202X/ab189b

[8] G. Zied, M. Sirine, Z. Mounir, and M. Mohsen, "Security analysis of the BB84 protocol in IoT networks," vol. 13, no. 4, pp. 169–174. [Online]. Available: http://www.warse.org/IJATCSE/static/pdf/file/ijatcse021342024.pdf

[9] S. Watanabe, R. Matsumoto, and T. Uyematsu, "Noise tolerance of the bb84 protocol with random privacy amplification," vol. 04, no. 6, pp. 935–946, publisher: World Scientific Publishing Co. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0219749906002316

[10] C. Lee, I. Sohn, and W. Lee, "Eavesdropping detection in BB84 quantum key distribution protocols," vol. 19, no. 3, pp. 2689–2701, conference Name: IEEE Transactions on Network and Service Management. [Online]. Available: https://ieeexplore.ieee.org/document/9750206/?arnumber=9750206&tag=1

[11] M. Elboukhari, M. Azizi, and A. Azizi, "Analysis of the security of BB84 by model checking," vol. 2, no. 2, pp. 87–98. [Online]. Available: http://www.airccse.org/journal/nsa/0410ijnsa7.pdf

[12] P. W. Shor, "Simple proof of security of the BB84 quantum key distribution protocol," vol. 85, no. 2, pp. 441–444.

[13] P. Van Den Bos and M. Huisman, "The integration of testing and program verification: A position paper," in *A Journey from Process Algebra via Timed Automata to Model Learning*, N. Jansen, M. Stoelinga, and P. Van Den Bos, Eds. Springer Nature Switzerland, vol. 13560, pp. 524–538, series Title: Lecture Notes in Computer Science. [Online]. Available: https://link.springer.com/10.1007/978-3-031-15629-8_28

[14] E. Dijkstra, *Notes on structured programming*, ser. EUT report. WSK, Dept. of Mathematics and Computing Science. Technische Hogeschool Eindhoven.

[15] TorXakis user documentation — TorXakis v0.9.0, document version v0.1.3 documentation. [Online]. Available: https://torxakis.org/userdocs/stable/

[16] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," vol. 14, no. 1, pp. 25–59. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0169755287900857

[17] JustANerd2, "JustANerd2/research-project," original-date: 2025-01-19T03:31:19Z. [Online]. Available: https://github.com/JustANerd2/Research-Project

[18] R. Srikanth and S. Banerjee, "Squeezed generalized amplitude damping channel," vol. 77, no. 1, p. 012318, publisher: American Physical Society. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.77.012318

[19] S. Harraz, S. Cong, and S. Kuang, "Optimal noise suppression of phase damping quantum systems via weak measurement," vol. 32, no. 5, pp. 1264–1279. [Online]. Available: https://doi.org/10.1007/s11424-018-7392-5

[20] D. Crow and R. Joynt, "Classical simulation of quantum dephasing and depolarizing noise," vol. 89, no. 4, p. 042123, publisher: American Physical Society. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.89.042123