Public

Network Security in a Quantum Future

Proof-of-concept development of a Quantum Threat Tracker (QTT) for the energy sector

NESO
National Energy System Operator

Network Security in a Quantum Future Contributing Partners

NESO	National Energy System Operator (Lead)
cambridge consultants Part of Capgemini Invent	Cambridge Consultants
THE UNIVERSITY of EDINBURGH	University of Edinburgh

Executive Summary

The field of quantum computing opens up many new opportunities. The topic is technically challenging, and its potential benefits and threats have attracted a great deal of academic research. The quantum threat to cybersecurity is gaining more prominence as a challenge that critical national infrastructure (CNI) providers must begin addressing in the near-term. Notably, the National Cyber Security Centre (NCSC) has just published guidelines for a timeline for CNI providers to migrate their cryptography to be quantum-safe, for example.

Network Security in a Quantum Future (NSiaQF), a Strategic Innovation Fund (SIF) Alpha project, is the next step in providing much-needed visibility for energy network operators. The insight it aims to provide into the scale and timing of the quantum threat for energy systems will enable energy networks to pursue appropriately tailored mitigations.

During the NSiaQF SIF Discovery phase, the project team scoped and designed two interconnected automated tools – the Quantum-Aware Risk Manager (Q-ARM) tool and the Quantum Threat Tracker (QTT). These will assist energy networks to identify and prioritise the systems requiring migration to post-quantum cryptography (PQC).

The objective of the Alpha phase was to develop proof-of-concept demonstrators of the proposed Q-ARM and QTT tools. This work has been led by the National Energy System Operator (NESO) with partners Cambridge Consultants and the University of Edinburgh.

The Q-ARM is a decision support software tool designed to help operators rapidly model energy sector systems and systematically identify the risks to those systems posed by quantum computers. It is designed to make post-quantum readiness assessments of energy network assets relatively simple, scalable and reproducible, helping to ensure good value for consumers and support network resilience. It will also help energy networks to meet the previously mentioned PQC-readiness timelines. See "Proof-of-concept development of Quantum-Aware Risk Manager (Q-ARM) for the energy sector"[1] for the Alpha phase report on the Q-ARM.

The QTT, the subject of this report, is designed to inform and directly feed into the risk assessment delivered by the Q-ARM tool with the latest quantum research. It does this by standardising and formalising the process of estimating when a cryptographically relevant quantum computer will emerge for specific algorithm/key length pairs.

In the Alpha phase, we have developed a modular architecture for the QTT that will enable quantum computing developments to be quickly integrated to provide the most accurate and up to date information for cybersecurity professionals. This architecture

[1] Both reports can be found on the ENA portal at https://smarter.energynetworks.org/projects/10129418/

has been implemented in a proof-of-concept demonstrator. It implements Shor's algorithm for generating estimates for RSA (Rivest–Shamir–Adleman), Diffie-Hellman and Elliptic Curve cryptography, enabling estimates to be produced for initial testing of the Q-ARM.

The estimates produced by the proof-of-concept demonstrator have been carefully benchmarked against the estimates that have been previously produced in the quantum computing academic literature. The estimates produced by the QTT are comparable to those in literature, providing confidence in the obtained results. Insights gained from this analysis include:

- Trade-offs in algorithmic performance to provide an estimate of the earliest vulnerability can lead to a reduction in the estimated qubit requirements in comparison to the literature.
- The produced estimates highlight that Elliptic curve cryptography with a key length of 256 is expected to be the first major public key system to fall to a quantum attack.

The value of the modular approach has been demonstrated by including a recent development in cryptographic algorithms (Chevignard et al., 2024) to reduce the number of required qubits at the cost of increased run time. From this we have found that the RSA-2048 protocol may be able to be broken with approximately 3.5x fewer qubits than the current leading estimate in the literature in exchange for a roughly 40x longer runtime.

There are three key components which make up the proof-of-concept demonstrator implementation of QTT:

- **Quantum Resource Estimation**: Evaluates the number of qubits and expected time to run a cryptographic algorithm for a given protocol and key size pair, and quantum computer specification (including details like error rates, gate times, and quantum error correction protocol).
- **Threat Report Generation**: Produces a detailed expected timeline for attacks on a given protocol and key size pair for a given quantum hardware development roadmap. Included in this timeline is the required number of qubits and time of attack by year.
- **Specification Requirement Estimation**: Determines the quantum computer specifications needed to break cryptographic protocols with a given number of qubits.

There are four key areas that require further developments to enable the success of the QTT and wide scale adoption within the energy sector:

1. **Providing more insight to Q-ARM**: Extending the functionality of the QTT to enable further insight that can be used by future versions of Q-ARM to provide refined risk estimates. This includes tools to capture and model the uncertainty in estimates for both time-to-attack and time-of-attack, along with modules to explore hypothetical algorithm improvements.
2. **Improvements to Data Accuracy**: Integrating recent advancements in quantum arithmetic and error correction codes. This includes expanding the library of cryptographically relevant quantum algorithms.
3. **Usability**: Developing a Graphical User Interface (GUI) and improving data presentation. Enhancing the QTT's usability by adding a GUI will make it more accessible to users who do not have extensive technical expertise.
4. **Open-Source Community Support**: The development of tools for the quantum computing community to promote engagement with the QTT and support the development of a community to provide long term support.

The QTT proof-of-concept demonstrator has demonstrated the feasibility of building a flexible and modular quantum resource estimation tool tailored to practical quantum cryptanalysis. Future work will focus on enhancing functionalities, improving data accuracy, and supporting usability and community engagement. The QTT's ability to provide detailed insights into the quantum threat landscape will be crucial for stakeholders in the energy sector to make informed decisions regarding cryptographic security.

Contents

Network Security in a Quantum Future Contributing Partners 2
Executive Summary 3
1. Introduction 7
The Network Security in a Quantum Future (NSiaQF) project 7
Understanding the Threat of Quantum Computers to Cybersecurity 7
Structure of this Report 8
2. Quantum Resource Estimation Preliminary 10
Quantum Resource Estimation as a Tool for Cryptanalysis 10
Quantum Resource Estimates in the Literature 10
Existing Software Tools for Quantum Resource Estimation 11
3. Quantum Threat Tracker (QTT) 14
Alpha phase QTT objectives 14
Contributions of the Quantum Threat Tracker 17
Threat Report Generation 17
Specification Requirement Estimation 19
Core Modules 21
4. Comparison With Resource Estimates in the Literature 25
RSA 25
ECC 25
Discrete Log 26
Simplifying Assumptions 26
5. Review of Alpha Phase Objectives 27
6. Future Directions 28
Improvements to Provide More Insight to Q-ARM 28
Improvements to Data Accuracy 29
Improvements to Usability 31
Features to Support the Development of an Open-Source Community 32
7. Glossary of Terms 34
References 36

1. Introduction

This report[2] forms one of the key deliverables from the Network Security in a Quantum Future (NSiaQF) project. The purpose of this report is to document the research performed in the proof-of-concept demonstrator development in the Strategic Innovation Fund (SIF) Alpha stage. This project follows a successful SIF Discovery phase, completed in 2024.

The Network Security in a Quantum Future (NSiaQF) project

During the Discovery phase of NSiaQF we scoped and developed an initial design for two interconnected tools. These tools could help energy network operators to prepare for the security threat posed by quantum computing by identifying and prioritising the systems requiring migration to post-quantum cryptography (PQC) in a scalable way.

The first of these tools – the Quantum-Aware Risk Manager (Q-ARM) – is designed to help operators rapidly model energy sector systems and systematically identify the risks to those systems posed by quantum computers (see NSiaQF Alpha phase Q-ARM report[3] for more details). The other – the Quantum Threat Tracker (QTT) - is designed to inform the risk assessment delivered by the Q-ARM tool with the latest quantum research, by standardising and formalising the process of estimating when a cryptographically relevant quantum computer will emerge for specific algorithm/key length pairs.

This report outlines the work completed to create a proof-of-concept demonstrator for the QTT during the Alpha phase of the NSiaQF project and is one of the key deliverables for the project.

Understanding the Threat of Quantum Computers to Cybersecurity

Public key, asymmetric, cryptography underpins modern digital security systems, including RSA, Diffie-Hellman, and elliptic-curve cryptography. These systems rely on the computational difficulty of a range of mathematical problems like integer factorisation and discrete logarithms, which are currently infeasible for classical computers to solve within a reasonable timeframe. However, Shor's algorithm demonstrates that a sufficiently large quantum computer could solve these problems efficiently, rendering public key cryptographic systems vulnerable. The implications are profound: encrypted data could be decrypted, digital signatures forged, and secure communications

[2] This report is written for a technical audience assuming a reasonable knowledge of cryptography and quantum computing. For an accessible introduction to cryptography the authors recommend "Codes and Cryptography" by Dominic Walsh and for quantum computing the primer by IBM to be found at https://www.ibm.com/think/topics/quantum-computing.

[3] Both reports can be found on the ENA portal at https://smarter.energynetworks.org/projects/10129418/.

compromised. Understanding quantum resource requirements for breaking these protocols is therefore an urgent priority.

Shor's algorithm, developed by Peter Shor (Shor, 1997), was one of the first theoretical demonstrations of quantum advantage. Shor discovered a way to use quantum computers to factor large numbers efficiently, solving a problem that is difficult and forms the foundation of many cryptographic systems. Since then, researchers have worked to refine Shor's algorithm and optimise it to make it more practical for real-world quantum hardware. In addition to these optimisations, newer variants (Chevignard et al., 2024; May & Schlieper, 2019; Regev, 2023) of quantum factoring and discrete logarithm algorithms have been developed, potentially offering alternative approaches to breaking cryptographic protocols with quantum computers. While significant progress has been made in improving the efficiency of key operations within the algorithm, such as modular arithmetic, implementing these factoring algorithms at scale remains a major challenge due to the limitations of current quantum hardware. Quantum computers today are error-prone and have limited physical resources, making it difficult to execute complex algorithms like Shor's algorithm without significant overhead.

Despite these challenges, advancements in quantum hardware and quantum error correction give reasons to believe that cryptographic systems relying on public key encryption will become vulnerable in the medium to long term. This raises critical questions for stakeholders: when should encryption systems migrate to post quantum protocols? How close are we to building quantum computers capable of breaking widely used cryptographic schemes? Answering these questions depends on multiple factors, including the progress in quantum hardware development, quantum error-correction methods, and quantum factoring algorithms. It is of utmost importance to be able to get regularly updated answers for these questions.

To respond to this need we have proposed and developed a proof-of-concept demonstrator, the QTT, a tool that addresses these pressing challenges at the intersection of quantum computing and cybersecurity. The QTT will leverage technical insight from quantum computing and quantum information to help security experts respond appropriately.

Structure of this Report

- In section 2 we introduce quantum resource estimation – in other words, the modelling of quantum algorithms to understand resource requirements and potential performance. This is the core on which the QTT is built. Included in this, we describe the work by Google and Microsoft that we build upon.

- In section 3 we provide a detailed description of the QTT architecture and the proof-of-concept demonstrator that we have developed in this project.
- Section 4 benchmarks the results we have obtained from the developed proof-of-concept demonstrator with the wider literature.
- Section 5 will look back at the objectives for the Alpha phase development of the QTT and assess the progress.
- Finally, section 6 looks forward to the next steps for developing the QTT to maximise its value to the community and to support its long-term applicability. These are split into 4 categories:
    1. Improvements to Provide More Insight to Q-ARM
    2. Improvements to Data Accuracy
    3. Improvements to Usability
    4. Features to Support the Development of an Open-Source Community

2. Quantum Resource Estimation Preliminary

Quantum Resource Estimation as a Tool for Cryptanalysis

Shor's algorithm provides the basis for theoretically efficient quantum algorithms for breaking public-key cryptographic protocols. However, answering the question of how powerful a quantum computer must be to run such algorithms is a complex task. This is primarily due to discrepancies between the physical assumptions used in the initial complexity analysis of Shor's algorithm, and the quantum computers we expect to be able to build in the future. Real quantum computers are expected to be error-prone, sparsely connected, and have non-negligible gate operation times. Quantum error correction is expected to be essential in the execution of Shor's algorithm, which will add significant overhead in both the required number of physical qubits and the algorithm runtime.

Quantum resource estimation (QRE) is essentially the study of how large a quantum computer needs to be built to run algorithms of interest in a fault-tolerant manner. QRE is therefore a natural tool for achieving the QTT's goal of understanding the timescales associated with Cryptographically Relevant Quantum Computing (CRQC). In the following subsections, we briefly review the existing tools for QRE and outline both the need for the QTT and how it fits within the larger ecosystem.

Quantum Resource Estimates in the Literature

There exist many resource estimates in the quantum computing literature, including those focussed on Shor's algorithm. One highly cited estimate is the result of Gidney and Ekerå (Gidney & Ekerå, 2021), who estimate that 2048-bit RSA numbers could be factored with about 20 million physical qubits in 8 hours based on realistic assumptions for future superconducting quantum computers. The QTT also uses this implementation as a benchmark.

In the reports produced in the Discovery phase of the project (NESO, 2024), the current state-of-the-art in implementations of Shor's algorithm were reviewed. For factoring and discrete logarithm problems, Gidney and Ekerå's variant is still considered to be the leading implementation with regards to resource estimation. For solving elliptic curve-based problems, new work such as (Garn & Kan, 2025) has been able to further reduce resource estimates.

A fundamental limitation of quantum resource estimates presented in academic papers is that they are often static, meaning they reflect resource requirements based on fixed assumptions at the time of publication. These results are often unable to capture how resource requirements will evolve with changes to the algorithmic subroutines,

cryptographic protocols, cryptographic key sizes, hardware assumptions, quantum error correcting codes, and qubit modalities. For example, if a cryptographic protocol adopts larger key sizes or if hardware improves with lower error rates, or a more efficient quantum error correction code is developed, the original estimates may no longer be accurate or relevant. The goal of the QTT is to automate the resource estimation process so that changes in these parameters can quickly yield new resource estimates with minimal effort, time, or technical expertise from the user.

Existing Software Tools for Quantum Resource Estimation

The rising interest in executing fault-tolerant quantum algorithms, and understanding the potential value of quantum computers, has led to the development of several software tools for performing QRE in recent years. Among these tools, the QTT utilises Qualtran and Azure Quantum. In the following section, we provide a brief overview of each tool and explain how their unique features contribute to the QTT's approach to resource estimation.

Qualtran

Qualtran, developed by Google Quantum AI, is an open-source Python library designed for expressing and analysing quantum algorithms targeting error-corrected quantum computers. It offers a modular approach to quantum algorithm development, employing "bloqs" as fundamental building blocks of quantum gates and subroutines that can be hierarchically composed. This compositional feature is a central strength, allowing for the construction of complex quantum algorithms from simpler, well-defined components. Qualtran enables users to simulate, test, and automatically generate diagrams for quantum algorithms.

Key features:

- **Hierarchical Composition**: Complex quantum routines can be built from simpler "bloqs."
- **Bloq Library**: Qualtran provides a library of common quantum subroutines and operations, streamlining the development process.
- **Resource Estimation**: The tool can estimate physical costs like wall-clock time and the number of physical qubits required for a surface-code architecture.
- **Correctness Guarantees**: Qualtran enforces correctness properties, such as the no-cloning and no-deletion theorems, to ensure that quantum computations don't violate the fundamental laws of quantum mechanics.

The QTT leverages Qualtran's bloq library to simplify the composition of complex algorithms such as Shor's. By utilising pre-built bloqs for modular arithmetic operations

and other subroutines, the QTT can express and analyse large quantum circuits with ease.

Azure Quantum

Azure Quantum is Microsoft's quantum computing platform and is home to the Azure Quantum Resource Estimator.

The standout feature of the Azure Quantum Resource Estimator utilised by the QTT is its ability to produce highly detailed physical resource estimates from logical resource estimates based on hardware assumptions and a sophisticated modelling of the overheads of quantum error correction (see Figure 1).

Figure 1: High-level workflow for using the Azure Quantum Resource Estimator for producing physical resource estimates. 'Logical counts' refers to abstract quantum circuit costs such as the number of logical qubits required and the number of gates. 'Hardware specification' refers to the parameters in the model of the physical quantum computer, such as gate times and error rates. 'Physical counts' refers to the physical resources required to execute the quantum circuit described by the logical counts, such as the number of physical qubits and wall-clock time.

Key features:

- **Logical to Physical Translation**: It converts logical resource estimates, such as the number of logical qubits and T gates, into physical resource requirements, such as the number of physical qubits and wall-clock time.
- **Quantum Circuit Analysis**: It can directly accept an explicit quantum circuit (e.g., constructed in quantum programming languages, including Qiskit or Q#), and analyse the physical resources required to execute it on hardware.
- **Hardware Customisation**: Users can model specific quantum architectures by customising parameters like qubit error rates, gate times, and quantum error correction schemes.
- **Extensibility**: The tool supports a wide range of quantum systems and allows for the modification of assumptions to suit unique hardware characteristics.

- **Trade-off Analysis**: It provides insights into the freedom of choosing space-time trade-offs when executing quantum algorithms via Pareto frontier estimation.

3. Quantum Threat Tracker (QTT)

The QTT module is one of the two key innovations being developed under the NSiaQF project (the other being the previously referenced Q-ARM, which ingests technical analysis about quantum from the QTT module).

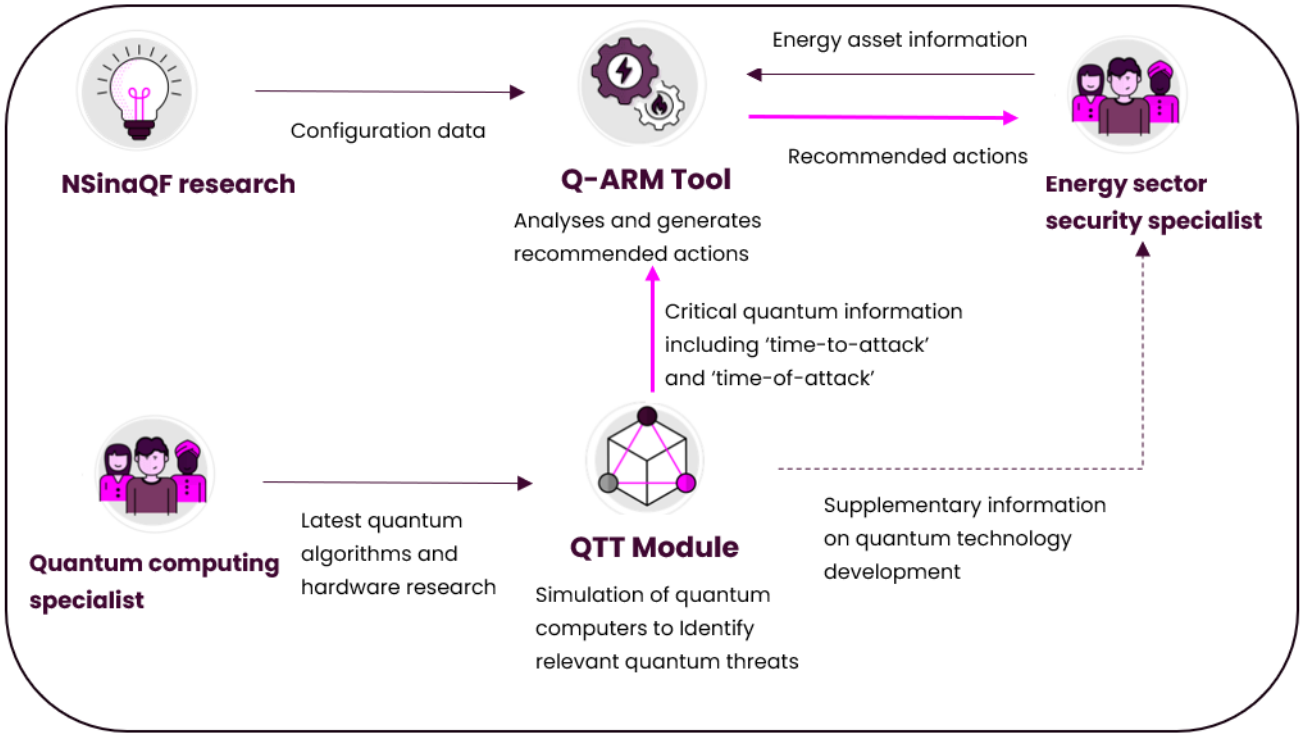

Figure 2: Tooling strategy for the NSiaQF project

The QTT is a modular framework designed to dynamically estimate the quantum resources required for cryptanalysis and provide actionable insights into the vulnerabilities of cryptographic systems. Achieving this involves three key steps. Firstly, it is essential to determine the quantum resources required to run a cryptographically relevant algorithm. Secondly, estimate when quantum computing resources will become available using hardware roadmaps. Finally, to support well-informed decision-making, it is important to assess the maximum potential of quantum hardware (of a given size and quality) to compromise cryptographic systems.

Alpha phase QTT objectives

Within Alpha, we aimed to demonstrate the feasibility of the QTT and that it can provide the required information for the Q-ARM tool. Through the development of a proof-of-concept, we looked to de-risk the proposed innovation prior to developing a fully featured tool and building an open-source community. Such an open-source

community will be required for the long-term success and maintenance of the QTT (work envisioned for a future Beta phase).

Though there has been previous work in quantum resource estimation for cryptographically relevant algorithms, there were still a few areas which required investigation. This led to the following objectives for the Alpha phase:

- Demonstrate the ability to generate useful estimates for the Q-ARM tool in the form of time-to-attack and time-of-attack for the most used asymmetric cryptographic protocols and relevant key sizes.
- Deliver these estimates using a modular software architecture which will allow an open-source community to implement alternative algorithms and error correction schemes to provide updated estimates.
- Compare generated estimates of system resource requirements against those available in the quantum computing literature to provide confidence that the approach taken produces reliable data.
- Demonstrate the versatility of the architecture by developing modules for both standard implementations of Shor's algorithm, and for Chevignard et al.'s recently published alternative factoring algorithm.
- Enable users to vary the hardware timelines used to generate the estimates rather than working only with a fixed timeline to enable quick integration of quantum hardware developments.

Overview of QTT Modules

The QTT is organised into three core modules designed to address these needs: Quantum Resource Estimation (QRE), Threat Report Generation, and Specification Requirement Estimation. These modules collectively enable users to analyse quantum computing threats, generate detailed reports, and explore hypothetical hardware requirements for breaking cryptographic protocols.

- **Quantum Resource Estimation:** This module quantifies the quantum resources needed to execute cryptographic-breaking algorithms, such as Shor's algorithm, for specific cryptographic protocols and key sizes. It provides estimates for key metrics like qubit count and runtime based on user-defined parameters.
- **Threat Report Generation:** The flagship functionality of the QTT, this module generates comprehensive threat reports that predict when cryptographic systems will become vulnerable. These reports are based on hardware roadmaps and algorithmic advancements and include detailed resource estimates alongside timestamps indicating when specific protocols are expected to be broken.

- **Specification Requirement Estimation:** This module reverses the typical resource estimation process by allowing users to input a maximum number of available qubits and receive the hardware specifications required to break a given cryptographic protocol.

By combining these modules, the QTT provides a versatile tool for assessing quantum threats.

Quantum Resource Estimation

As discussed in the Quantum Resource Estimation section, understanding how large of a quantum computer needs to be built to execute Shor's algorithm is vital in quantifying the quantum threat. At a high level, the resource estimation module of the QTT accepts two compulsory inputs and two optional inputs, as listed below:

- **Quantum algorithm:**This is the quantum algorithm to analyse in the creation of the resource estimate. For example, this could be Gidney and Ekerå's highly optimised implementation of Shor's algorithm for factoring, or Litinski's implementation for computing elliptic curve keys.
- **Cryptographic parameters:** The parameters specifying the cryptographic protocol. These are the protocol (e.g., "RSA") and key length (e.g., 2048). The protocol specified must be compatible with the algorithm used; for example, if the algorithm is Litinski's elliptic curve implementation then the protocol must be "ECDH".
- **Algorithm parameters (optional):** Many implementations of Shor's algorithm have user-specifiable parameters offering trade-offs between performance metrics such as algorithm runtime and qubit count. All of the QTT's algorithm implementations have a default set of algorithm parameters, but the user may also specify their own.
- **Estimator parameters (optional):** For creating physical resource estimates, the QTT requires information about the quantum computer specification. These are specified in Azure Quantum's format and contain information such as the gate error rates, gate operation times, maximum physical qubits, and error budget. The user may even model custom QEC codes in these parameters. If unspecified, the QTT defaults to Azure Quantum's "qubit_gate_ns_e3" preset, which models realistic superconducting qubit parameters.

The QTT then uses these inputs to generate a quantum resource estimate, output using Azure Quantum's format. This specifies details like the qubit count and algorithm runtime.

Contributions of the Quantum Threat Tracker

The Azure Quantum Resource Estimator provides a straightforward way to produce resource estimates by explicitly constructing and analysing quantum circuits. However, during the development of the QTT, it was discovered that this approach is impractical for algorithms like Shor's, which require circuits far too large for existing quantum programming languages such as Qiskit or Q# to handle. To address this limitation, the QTT employs a modular approach that breaks down larger quantum circuits into smaller components. Instead of analysing entire circuits, it focuses on estimating resources for a single repeating subcircuit and scales these results to represent the full algorithm. This method enables efficient analysis of complex algorithms involving billions of operations.

For example, Shor's algorithm performs an expensive quantum arithmetic operation termed modular exponentiation, which is composed of repetitive controlled modular multiplication operations. Rather than constructing and analysing the entire modular exponentiation circuit, the QTT estimates resources for just one controlled modular multiplication—a much smaller component—and extrapolates these results to represent the full algorithm. This avoids direct analysis of the complete circuit, significantly reducing computational effort while maintaining accuracy.

The QTT achieves this efficiency through a hybrid workflow that combines Qualtran's decomposition capabilities with Azure Quantum's physical resource modelling. Qualtran represents circuits as abstract bloqs, enabling logical resource estimation at an abstract level. Azure Quantum translates these logical estimates into physical qubit counts and runtime, accounting for error correction and hardware constraints. Together, these tools make resource estimation feasible for cryptographic-breaking algorithms at scale.

Threat Report Generation

The QTT's flagship functionality is its ability to create 'threat reports', which are automatically generated reports in the JSON file format detailing the estimated threats against cryptographic protocols. A threat report contains a list of cryptographic protocols, each of which is paired with a list of threats. The report can be generated in varying levels of detail; at the simplest level, a threat is simply a future timestamp at which the QTT expects the protocol to be broken. More detailed levels include information like the quantum algorithm runtime, number of qubits, code distance, etc. Since different quantum algorithms and algorithm parameters offer trade-offs between qubit count and runtime, in general, the QTT outputs multiple threats associated with each cryptographic protocol.

To generate a threat report, the QTT requires two inputs from the user:

- **Hardware roadmap:** A projection into the future of where we expect quantum computing hardware to be at any given time.
- **Cryptographic protocols:** The list of cryptographic protocols and key sizes of interest.

Hardware roadmaps can be constructed by the user using the propriety class included in the QTT. The quantum computers present in the hardware roadmaps are specified using Azure Quantum's estimator parameters class (the same as that which is described in the Alpha phase QTT objectives section). As well as gate error rates and operation times, all quantum computers present in the hardware roadmap must specify a number of maximum available physical qubits, as the QTT will use this information to decide whether or not it is possible to execute Shor's algorithm on the specified hardware.

When a threat report is generated, the QTT lists all quantum algorithms eligible for breaking each cryptographic protocol then automatically optimises the quantum algorithm parameters. For each algorithm, the QTT makes two optimisations: one which targets the smallest possible number of physical qubits, and one which targets the shortest possible runtime. Both of the threats which these two optimisations create are included in the threat report.

Once the full threat report has been generated, the QTT is able to plot some of the most important information about the threats. An example output of this graphing functionality is shown in Figure 3. For readability, in this plot the QTT only outputs the 'soonest' threat associated with each protocol; that is, that which has the smallest timestamp. Generally, not all threats are shown on the plot; that is, there may be other threats that have timestamps that are further away but have shorter algorithm runtimes.

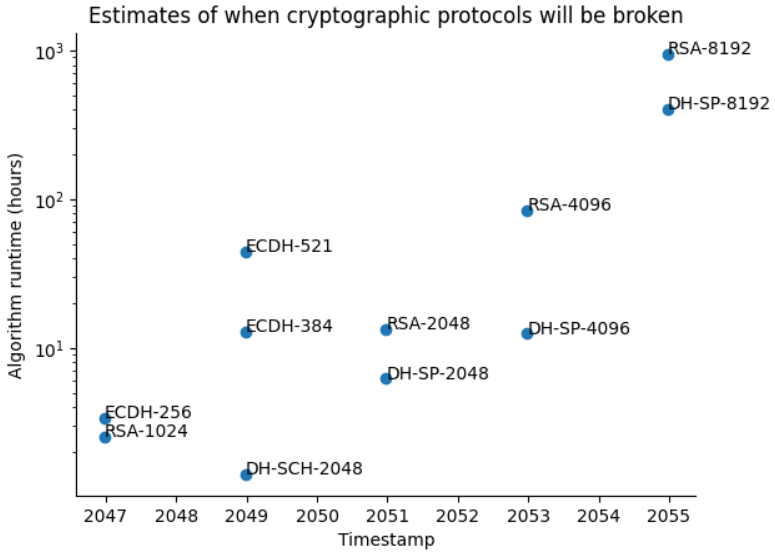

Figure 3: Example output of the QTT in graphical form.

Specification Requirement Estimation

The final core feature of the QTT we discuss here is 'specification requirement estimation'. In general, QRE asks the user for quantum computer specifications such as gate error rate and outputs the number of qubits required to perform the computation. The specification requirement estimation feature of the QTT essentially tackles the converse of this task; in other words, the user inputs a maximum number of available qubits, and the QTT returns the quantum computer specifications required to perform the computation. Specifically, for this task, the QTT requires two inputs from the user:

1. **Cryptographic protocol**: Cryptographic protocol and key size.
2. **Maximum number of available qubits**: Number of qubits available.

The QTT then returns the hardware specification required to break the cryptographic protocol with the given number of maximum qubits. Specifically, it returns an estimator result in Azure Quantum's format, which includes details like the gate error rate and code distance.

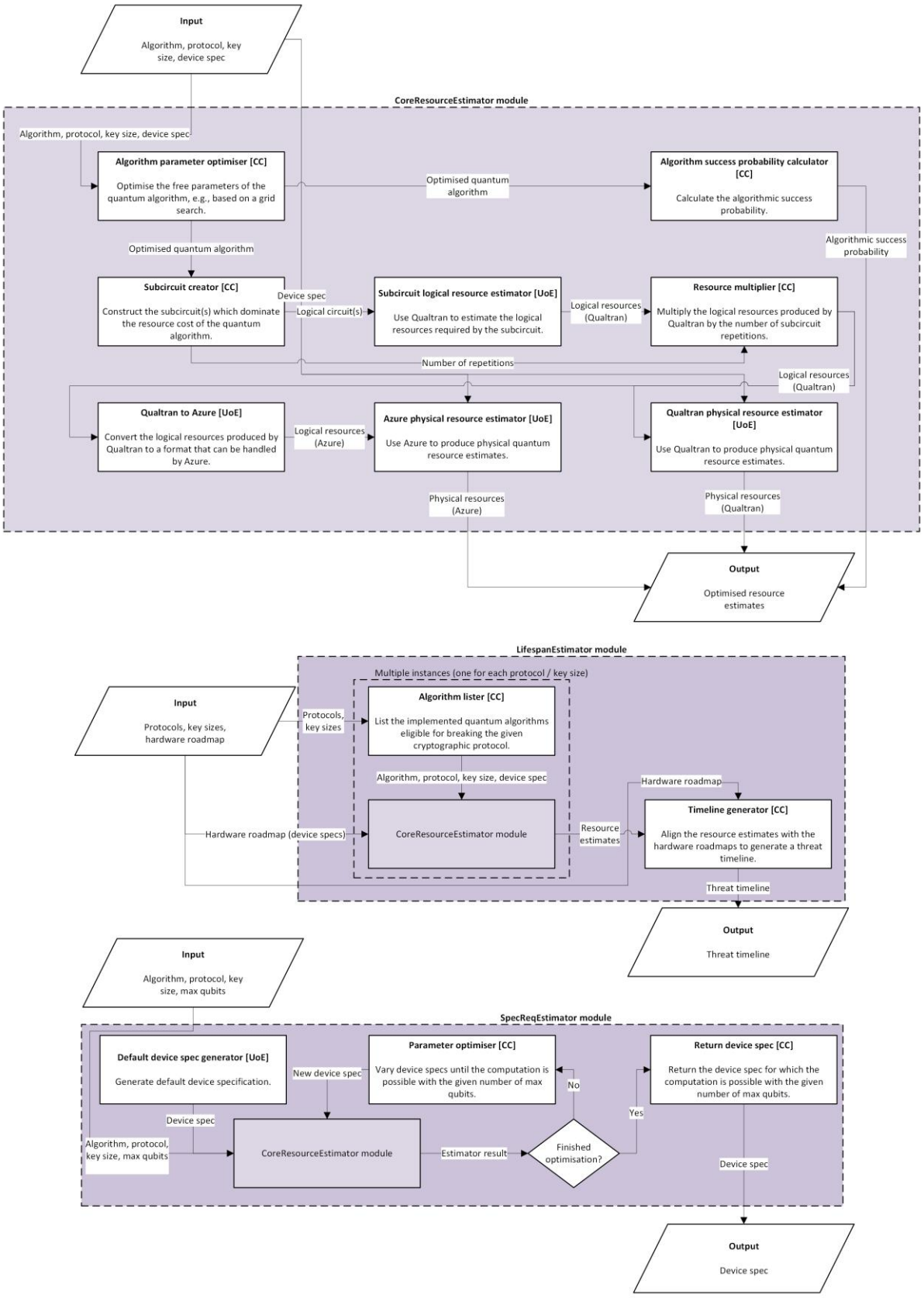

Figure 4: Architecture diagram for the QTT

Core Modules

As discussed in the Quantum Resource Estimates in the Literature section, existing resource estimates of Shor's algorithm have the limitation that it is unclear how they will change when there are changes in the algorithmic subroutines, hardware assumptions, QEC codes, etc. The QTT overcomes this limitation by employing a modular architecture, in which modules may be swapped out or modified to quickly explore how the aforementioned changes affect the quantum threat.

Architecturally, the QTT consists of three core modules, as detailed in Figure 4. In this section, we detail the functionality of each of these modules, their inputs and outputs, and their constituent sub-modules.

Core Resource Estimator

The Core Resource Estimator is the largest module in the QTT and forms the foundation for the rest of the functionality of the tool. The other modules are built on top of this module and provide higher level functionality. The purpose of this module is to estimate the number of qubits necessary to break a specified cryptographic protocol on a specified quantum computer.

In addition to the previously discussed flexible modular approach, this module is specifically designed with the goal of creating quantum resource estimates extremely efficiently. This is essential for the latter modules in the QTT to work effectively, which can only function by instantiating many copies of the Core Resource Estimator module. To achieve this highly efficient quantum resource estimation workflow, we use Qualtran to estimate logical resource estimates, then use Azure Quantum to convert these into highly detailed physical estimates based on the given hardware specification.

Inputs:

- A cryptographic protocol and associated key size.
- A quantum algorithm compatible with breaking the cryptographic protocol.
- A quantum computer device specification.

Outputs:

- Success or failure; i.e., can the specified device break the given protocol.
- The number of logical qubits required to break that protocol on a quantum computer.
- The number of physical qubits required to break that protocol on a quantum computer.
- The expected time to break that protocol on a quantum computer.

- Auxiliary information, such as the number of magic state factories, QEC code distance, etc.

Submodules:

- Algorithm parameter optimiser – optimises the free parameters of a given quantum algorithm (if there are any) to minimise the logical resource estimates.
- Algorithm success probability calculator – calculate the algorithmic success probability (assuming perfectly error-free quantum circuits).
- Subcircuit creator– construct the subcircuit which dominates the resource cost of the quantum algorithm. For Shor's algorithm and its variants, this is usually a quantum non-modular adder circuit.
- Subcircuit logical resource estimator – extract logical resource estimates from the subcircuit returned by the subcircuit creator. This may include qubit counts, Toffoli gates, T gates, arbitrary rotations, and Clifford gates.
- Resource multiplier – multiply the logical resource estimates returned by the subcircuit logical resource estimator by the number of times that subcircuit is repeated in the full circuit.
- Qualtran to Azure – convert logical resource estimates from Qualtran's format to Azure's format.
- Qualtran physical resource estimator – use Qualtran to generate physical resource estimates from the logical resource estimates in Qualtran's format.
- Azure physical resource estimator – use Azure's quantum resource estimator to generate physical resource estimates from the logical resource estimates in Azure's format.

Lifespan Estimator

The Lifespan Estimator generates the primary output of the QTT (and subsequently the input for the Q-ARM tool). When given a roadmap for quantum computing hardware options and the specifications of cryptographic protocols of interest, this module outputs a threat timeline; that is, the time in the future when the cryptographic protocols are predicted to be broken, and the expected resource requirements through time. An example output of the QTT, highlighting the most important aspects of the quantum threat, is shown in a graphical form in Figure 3.

Inputs:

- A list of cryptographic protocols and key sizes.

- Quantum hardware roadmap, i.e. a table of quantum platform specifications (including qubit count, error rates) against expected year of arrival.

Outputs:

- The estimated date at which each given protocol will be broken, given the assumptions in the quantum hardware roadmap (Time to/until attack).
- Series of expected times to break a single key for the given cryptological protocol for quantum platform specifications on the quantum hardware roadmap labelled by expected year.

Submodules:

- Algorithm lister – list all quantum algorithms eligible for breaking the given cryptographic protocol to be considered in the resource estimates. For example, for RSA, this may include the baseline implementation of Shor''s algorithm, Gidney and Ekerå''s implementation, and Chevignard's algorithm.
- Timeline generator – aligns the resource estimates for each cryptographic protocol and key size with the hardware roadmap to estimate when each protocol will be broken.

Specification Requirement Estimator Module

The Specification Requirement Estimator module will estimate the minimum device specification (e.g., gate error rate) required to break a given cryptographic protocol on a specified number of physical qubits.

Inputs:

- A cryptographic protocol and associated key size.
- The number of physical qubits and the architecture.
- The maximum expected time to break the specified cryptographic protocol.

Outputs:

- The quantum computing hardware specification needed to break the given protocol.

Submodules:

- Default device spec generator – generates an initial set of device parameters.
- Parameter optimiser – optimises over the device specifications, until the output of the Core Resource Estimator closely matches the target qubit count.

- Return device spec – returns the device specifications for which the target qubit count is achieved; that is, the solution to the optimisation problem solved by the parameter optimiser.

4. Comparison With Resource Estimates in the Literature

This section summarises and highlights how the QTT resource estimates compare to existing benchmarks in the literature, emphasising our focus on minimising physical qubit requirements to determine the earliest possible vulnerability timelines for cryptographic systems. This approach directly addresses the critical question: In which year will each cryptographic protocol first become vulnerable?

Since quantum hardware development is primarily constrained by physical qubit availability, qubit minimisation provides the most conservative estimate of attack timelines. While the numbers presented here focus on qubit minimisation, the QTT framework is inherently capable of targeting alternative objectives, such as minimal runtime. In practice, the QTT can output both qubit- and runtime-optimised configurations (see the "Threat Report Generation" section for details).

RSA

For RSA-2048, our implementation of the algorithm presented in Gidney & Ekerå, 2021 yields:

- Physical qubits: 16.17 million (19% fewer than Gidney & Ekerå, 2021's estimate of 20 million).

As mentioned previously, this represents an intentional trade-off to identify the earliest vulnerability point. While Gidney & Ekerå optimised for the overall spacetime volume of the computation (balancing runtime and qubit count), we allowed for a lower success probability and more repetitions in favour of achieving a lower qubit count.

Our implementation of Chevignard's algorithm yields:

- Physical qubits: 5.75 million.
- Runtime: 13 days.

To our knowledge, this is the first quantum resource estimate of the physical requirements for executing Chevignard's algorithm. This shows that the logical qubit reduction presented in their work does indeed translate into a reduction in the physical number of qubits, and that RSA may be broken with fewer qubits at the cost of a much-increased runtime.

ECC

For ECC-256, our implementation achieves:

- Physical qubit requirements: Consistent with (Litinski, 2023).

The QTT confirms that ECC-256 will likely be the first major public-key system to fall to quantum attacks, requiring only ~6 million physical qubits.

Discrete Log

For discrete logarithm estimates, we specifically analyse Schnorr groups and Safe Prime groups with short exponents. Our minimum results for 2048-bit moduli are as follows:

- Schnorr groups: 14 million physical qubits (30% fewer than literature benchmarks).
- Safe Prime groups: 16.17 million physical qubits (19% fewer than literature benchmarks).

Simplifying Assumptions

Our current estimates make certain simplifying assumptions about arithmetic subroutines that do not significantly affect the final estimates. For example:

We assume basic carry propagation for addition rather than incorporating detailed modelling of parallelised carry runways or coset representations, as explored in Gidney & Ekerå's work. While these details can refine runtime estimates, they have minimal impact on physical qubit counts, which remain our primary focus. By contrast, other works in the literature delve deeper into such optimisations. For instance, Gidney & Ekerå introduce carry runways for parallelised addition, which results in a lower runtime for running their factoring algorithm.

5. Review of Alpha Phase Objectives

Having completed the development of the proof-of-concept demonstrator detailed in this report there are a few key findings related to the de-risking of the future QTT development that we bring together here. Through these findings we also highlight how we have been able to meet the objectives set out in the introduction.

- Our initial approach of building the QTT on top of Microsoft's Azure Quantum resource estimation tool was not possible given the inability of current quantum programming languages to deal with quantum circuits of the size required. Instead, a hybrid approach was taken using both parts of Google's Qualtran resource estimation and Microsoft's Azure Quantum resource estimation tool. This enabled us to obtain the required estimates while also providing the flexibility need for dealing with hardware roadmaps.
- We were able to generate estimates for time-to-attack and time-of-attack for RSA, Diffie-Hellman and Elliptic Curve cryptography for common key sizes which have been successfully integrated into the proof-of-concept demonstrator of the Q-ARM tool.
- We have developed a modular architecture for the QTT (detailed in section 3), which will allow new algorithms to be quickly implemented and integrated into the data provided to future versions of the Q-ARM tool.
- We have benchmarked the estimates provided by the developed proof-of-concept demonstrator against those provided in the literature (see section 4). This provides confidence in the developed tools and the accuracy of the results. This confidence will be further reinforced in future versions of the QTT by making the code open source to enable the wider community to also verify the modelling approach.
- Chevignard et. al's variant has been implemented and we have been able to generate estimates for both time-to-attack and time-of-attack. This demonstrates the important feature of being able to incorporate future algorithmic improvements. See section 4 for details.
- The proof-of-concept demonstrator accepts a hardware roadmap, as an input that the user can edit and change. This enables users to easily change the assumptions on the quantum hardware development path and generate new estimates. For more details see the description of the Lifespan Estimator module.

The proof-of-concept demonstrator of the QTT has successfully demonstrated the possibility of building a flexible and modular quantum resource estimation tool tailored to practical quantum cryptanalysis.

6. Future Directions

Turning the proof-of-concept demonstrator developed in this phase of the project into a full software suite in the following phase, we foresee several potential future directions for expansion and improvement:

1. Improvements to Provide More Insight to Q-ARM.
2. Improvements to Data Accuracy.
3. Improvements to Usability.
4. Features to Support the Development of an Open-Source Community.

Improvements to Provide More Insight to Q-ARM

The QTT's primary functionality is to generate data to drive informed decisions regarding cryptosystems; in this case, data to be input into the Q-ARM tool. The first and highest-priority category of improvements we consider is the possible enrichments to the data output by the QTT.

Uncertainty in Estimates

The QTT's current model of the progression of quantum computing hardware is that it will evolve exactly according to a hardware roadmap specified by the user with no deviation from that roadmap. This model could be generalised and improved by allowing the specification of uncertainty, giving probabilistic outcomes for when cryptographic protocols will be broken, rather than fixed points in time. This information would be especially valuable to security experts wishing to make informed decisions based on how likely certain protocols are to be broken at certain points in time.

Uncertainty in the estimates of when cryptographic protocols will be broken may arise from several sources. One of the primary sources of uncertainty in the QTT's estimates is the timeline for achieving specific hardware milestones, particularly the availability of a certain number of qubits by a given date. This uncertainty stems from the unpredictable nature of experimental breakthroughs, or setbacks in scaling quantum hardware. Another such source is the sensitivity of resource estimates to changes in the underlying assumptions. For example, if there is some small chance that future quantum computers have gate error rates that are an order of magnitude below what we expect, this should be reflected in the QTT's output. Therefore, sensitivity analysis will play a vital role in incorporating uncertainty into the QTT.

Another potential source of uncertainty is the collection of potential theoretical improvements that may occur in quantum computing over the future, as further discussed in the Modelling Theoretical Improvements section. Although it is impossible to precisely model QEC codes and quantum algorithms which do not yet exist, one can

hypothesise abstractly about subroutines that may decrease resource costs such as qubit counts, gate depths, and coding rates. Given the rapid improvements to the theory of quantum computing over the last few decades, such potential future improvements should also be captured when attempting to quantify the quantum threat.

Modelling Theoretical Improvements

The QTT's threat forecasting capabilities developed in the Alpha phase operate under the basic assumption that quantum error correcting codes, algorithms, and subroutines will remain unchanged for the foreseeable future, while the only factor which affects the risk level of the considered cryptographic protocol is the progression of quantum computing hardware. The accuracy of the QTT's estimates could therefore be improved by estimating not only the development of quantum computing hardware, but also what theoretical improvements in the field could occur in the coming years/decades.

These models will require a less fine-grained approach compared to much of what is currently used by the QTT. For example, if one hypothesises that QEC will achieve a 10x better coding rate in the future, or that circuit gate depths of leading algorithms will decrease by 10%, it will only be possible to create resource estimates for these improvements by reasoning very abstractly about quantum circuit costs.

Improvements to Data Accuracy

The QTT currently produces its estimates using the algorithms, QEC codes, and qubit modalities that are the most well-studied and considered to be the current state-of-the-art. There are, nonetheless, rapid developments happening in these areas, and implementing the alternative options will further enhance the quantity and accuracy of the data output by the QTT.

Algorithm Expansion

This project has focused on demonstrating the modular approach of the QTT and, as such, has only been a proof-of-concept rather than providing a complete library of available cryptographically relevant quantum algorithms. The proof-of-concept nature of the work means we have primarily focused on building a carefully considered framework rather than integrating the most cutting-edge techniques. Therefore, a key piece of future work is taking the solid foundation set up in this phase of work and integrating alternative algorithm approaches with recent theoretical improvements.

Recent work (Kahanamoku-Meyer & Yao, 2024; Litinski, 2024) has introduced quantum arithmetic circuits that are far more qubit- and/or gate-efficient than previous implementations. These circuits have the potential to reduce resource estimates for breaking cryptographic protocols. Integrating these methods into the QTT would be a

natural and impactful next step for the tool. Updating the QTT with the latest advancements in quantum arithmetic is essential to ensure its accuracy and relevance in estimating cryptographic vulnerabilities.

Fundamentally, new variants of factoring algorithms, such as Regev's algorithm (Regev, 2023), offer alternative approaches to breaking cryptographic protocols. These algorithms have yet to be fully analysed in terms of practical resource requirements, partly because their implementation involves overcoming theoretical barriers and adapting them to realistic quantum hardware constraints. The QTT provides a unique opportunity to simplify this analysis by offering a modular framework for building resource estimates for new algorithms. Incorporating these factoring variants into the QTT will not only expand its capabilities but also provide valuable insights into their feasibility and impact on cryptographic security.

Another important extension for the QTT would be the analysis of symmetric cryptography, particularly for small key sizes like AES-128. While Grover's algorithm reduces the query complexity of breaking AES-128 keys from 2^{128} classical queries to just 2^{64} quantum queries (Sarah & Peter, 2024), translating this quadratic speedup in query complexity into practical resource estimates remains a critical gap.

Error Correction Code Analysis

The QTT currently focuses on resource estimation for surface codes (Fowler et al., 2012; Horsman et al., 2012) and quantum error correction codes (QECCs), which are well understood in terms of logical operations, and resource costs. The surface code is the current industry-standard approach to quantum error correction, favoured for its straightforward implementation using only local operations between neighbouring qubits. However, from an information-theory perspective, it is highly inefficient. Results from the QTT suggest that surface codes with a distance of up to 30 will be required to implement Shor's algorithm for factoring, yielding an encoding rate (the ratio of logical to physical qubits) of $r \approx 10^{-3}$.

Recent advancements have introduced alternative quantum low-density parity-check codes (qLDPC) as a promising alternative to surface codes. While these codes require long-range connectivity between qubits, they offer a significant reduction in qubit requirements; simulations suggest qubit counts could be reduced by an order of magnitude compared to surface codes. As a result, qLDPC architectures have gained traction, with four quantum computing companies - QuEra, IBM, Alice & Bob, and Photonic - adopting them to date.

A bottleneck in the development of qLDPC codes is that there is currently no resource estimation software that supports them. The QTT is designed in a modular fashion, making it well-suited for extension to qLDPC codes. Extending the package to support

qLDPC codes would provide a high-impact use case. Specifically, it will allow developers to understand how the choice of QEC protocol changes the resources required to implement algorithms in a fault-tolerant way, bridging the gap between QECC development and practical applications.

More Enriched Hardware Roadmaps

The hardware roadmaps currently used by the QTT focus on superconducting and trapped ion qubit architectures, since these are the most well-understood and promising qubit modalities. Nonetheless, in the future it may be useful to understand the quantum threat on other qubit modalities, such as neutral atoms and photonics.

The development of a roadmap generation methodology would also have the potential to improve the accuracy of the currently used roadmaps. At present, the roadmaps used by the QTT assume a steady, exponential increase in the number of qubits (following Moore's law), while the other parameters such as gate error rates and operation times stay constant. These other parameters could also change in the future, and the hardware roadmaps could be enhanced by reflecting such possible changes.

Improvements to Usability

The QTT currently takes the form of a standard Python package. While this is ideal for users wanting granular control or developers wanting to incorporate the QTT's functionality into their own software, it presents usability challenges for those unfamiliar with Python. Furthermore, the QTT's output primarily takes the form of text-heavy JSON files, which, while human-readable, favour detail over conciseness. We thus foresee possible room for improvement in the QTT's usability, both in executing the software and in presenting the data output by it.

Usability Enhancement: Adding a GUI and a Browser-based Frontend

The current version of the QTT offers flexibility through its Python package interface, allowing users to modify components such as device specifications, quantum algorithms, algorithmic parameters, and QECC implementations. However, this interface provides a significant barrier to engagement by users, especially in cases where this level of detail may not be necessary.

For users who simply want high-level insights into protocol vulnerabilities or resource estimates, alternative interfaces are important. Developing a GUI would make the QTT more accessible by providing an intuitive way to interact with its features without requiring extensive technical expertise. The GUI would serve as a front-end layer that simplifies complex analyses while retaining the full power of the QTT backend.

Developing a browser-based frontend for the QTT would further enhance accessibility and user engagement. This would combine the advantages of a graphical user interface with the convenience of browser-based access, removing the need for users to install bespoke software or manage complex dependencies. A web app would also make it possible to deliver the QTT in a lightweight, user-friendly format accessible from any device.

Data Presentation

While the raw data output by the QTT is comprehensible, the tool prioritises outputting all relevant data over conciseness. The QTT is also able to convert its output data into concise graphical representations, but these sacrifice some of the data relevant to fully understanding the quantum threat. Therefore, the QTT's data presentation could be improved by striking a middle ground between readability and detail. We foresee two immediate possible areas for improvement: space-time trade-offs and uncertainty:

- Space-time trade-offs: In general, the QTT foresees multiple threats for each given cryptographic protocol. In the context of quantum algorithms, these offer various trade-offs between qubit count and runtime. From a practical standpoint, this means a given cryptographic protocol may be broken at various points in the future, with points further in the future offering potentially shorter runtimes. Therefore, each data point in the graphic showcasing the quantum threat could be better represented as a curve, or a series of points, rather than a single point.
- Uncertainty: As previously discussed, future versions of the QTT will incorporate uncertainty modelling to account for advances in hardware, algorithms, and QEC primitives. These uncertainties will be incorporated into the QTT output as error bars on each data point (or curve).

Features to Support the Development of an Open-Source Community

While the QTT is completely open-source and developers are free to clone and modify code from the repository, the tool in its current state favours simple code over user-expandability. As discussed in the Core Modules section, the QTT is built in a modular fashion with future expansion in mind; however, the development of more tools for this task will help others contribute to the QTT and foster an open-source community.

Quantum Algorithm Profiling

Classical computing has well-established tools for profiling programs to identify bottlenecks and optimise performance. In quantum computing, such tools are virtually non-existent, leaving algorithm developers with limited insights into where

computational costs arise. Adding profiling capabilities to the QTT would allow users to analyse resource consumption at a granular level—identifying which parts of an algorithm dominate qubit usage, gate count, or execution time. This feature would be invaluable for both researchers seeking to optimise algorithms and security professionals looking to understand how resource estimates for cryptographic attacks could be reduced further.

Computation Blueprint

Currently, the QTT handles quantum circuits abstractly without explicitly presenting how one would build and execute such a circuit in practice. An ideal QRE output of the QTT (or any resource estimation framework in general) would be to produce an exact blueprint of quantum computations required to break cryptographic protocols – that is, a complete end-to-end compilation from high-level algorithm design down to error correction code implementation and finally to the physical qubit operations.

Achieving this requires solving difficult problems in quantum compilation, including precise gate decomposition, synchronisation of error correction cycles, and hardware-specific instruction mapping. While challenging, this capability would provide near-perfect resource estimates for cryptographic attacks, offering unparalleled precision in assessing quantum threats and guiding long-term security planning.

7. Glossary of Terms

Term	Meaning
Classical computer	A conventional non-quantum computer, such as widely used laptops and desktop computers.
Computational efficiency	The computational resource required to complete a calculation.
Connectivity	A description of how the qubits are connected in a quantum computer.
Diffie -Hellman (DH)	A public key cryptosystem which relies on the difficulty of the discrete log problem.
Discrete log problem	The computational challenge of finding the exponent which solves a specific equivalence relation over a given finite group. Used to attack Diffie-Hellman protocol.
Elliptic Curve Cryptography (ECC)	A method of public key cryptography based on elliptic curves which allows for smaller key sizes when compared to non-ECC schemes.
Elliptic Curve Diffe-Hellman (ECDH)	An elliptic curve-based version of the DH cryptographic scheme, an example of an ECC.
Gate error rate	The probability of an error occurring upon execution of a (quantum) logic gate.
Gate operation time	The real-world time required to execute a (quantum) logic gate.
Integer Factorisation	The computational problem of taking a composite (non-prime) integer and outputting its prime factors.
JSON file format	An open standard file format that uses human-readable text to store and transmit data objects consisting of name–value pairs.
Low density parity check (LDPC) codes	A class of classical error correcting codes.
Logical (Quantum) Bit	The logical unit of information which can be freely programmed by the user.

Term	Meaning
One-Way Function	A mathematical function which is computationally easy to compute given an input but hard to invert given an output.
Physical (Quantum) Bit	The physical units of information which encode information in the device.
Public Key Cryptography (PKC)	A class of cryptographic protocols based on one-way functions that allow users to agree upon a shared secret with no prior shared information.
Quantum Algorithm	An algorithm which runs on a quantum computer.
Quantum Bit (Qubit)	An elementary unit of information in quantum computing, generalising the binary digit (bit).
Quantum Circuit	A set of instructions defining an algorithm to execute on a quantum computer.
Quantum Circuit Depth	The total number of time steps required to execute a quantum circuit, where each time step may contain multiple quantum gates being executed in parallel.
Quantum Computer	A computer which can utilise quantum mechanical phenomena in its computations.
Quantum Error Correction (QEC)	A method for protecting quantum information against errors by encoding the information of logical qubits onto many more physical qubits.
Quantum Gate	An elementary instruction for a quantum computer, analogous to classical gates such as AND and NOT.
Quantum Low Density Parity Check (QLDPC) codes	A QEC code family based on ideas from classical LDPC codes.
Rivest–Shamir–Adleman (RSA)	A widely used public key cryptosystem that relies on the difficulty of integer factorisation.
RSA Number	The product of two (large) prime numbers, also called a semiprime.
Wall Clock Time	The real-world time required to execute a (quantum) algorithm, as distinct from the concept of time in

Term	Meaning
	complexity theory which counts the number of operations.

References

Chevignard, C., Fouque, P.-A., & Schrottenloher, A. (2024). Reducing the Number of Qubits in Quantum Factoring. *Cryptology EPrint Archive*.

Fowler, A. G., Mariantoni, M., Martinis, J. M., & Cleland, A. N. (2012). Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, *86*(3), 32324.

Gidney, C., & Ekerå, M. (2021). How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, *5*, 433.

Horsman, D., Fowler, A. G., Devitt, S., & Van Meter, R. (2012). Surface code quantum computing by lattice surgery. *New Journal of Physics*, *14*(12), 123011.

Kahanamoku-Meyer, G. D., & Yao, N. Y. (2024). Fast quantum integer multiplication with zero ancillas. *ArXiv Preprint ArXiv:2403.18006*.

Litinski, D. (2023). How to compute a 256-bit elliptic curve private key with only 50 million Toffoli gates. *ArXiv Preprint ArXiv:2306.08585*.

Litinski, D. (2024). Quantum schoolbook multiplication with fewer Toffoli gates. *ArXiv Preprint ArXiv:2410.00899*.

May, A., & Schlieper, L. (2019). Quantum period finding is compression robust. *ArXiv Preprint ArXiv:1905.10074*.

Quantum Threat Tracker. (2024). https://github.com/qec-codes/QuantumThreatTracker

Regev, O. (2023). An efficient quantum factoring algorithm. *ArXiv Preprint ArXiv:2308.06572*.

Sarah, D., & Peter, C. (2024). On the practical cost of Grover for AES key recovery. *Presentation at the 5th NIST PQC Standardization Conference*.

Shor, P. W. (1997). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer