


# qkdSim, a Simulation Toolkit for Quantum Key Distribution Including Imperfections: Performance Analysis and Demonstration of the B92 Protocol Using Heralded Photons

Rishab Chatterjee,<sup>1</sup> Kaushik Joarder,<sup>1</sup> Sourav Chatterjee<sup>1</sup>,<sup>1</sup> Barry C. Sanders<sup>1,2</sup>,<sup>1,2</sup> and Urbasi Sinha<sup>1,\*</sup>

<sup>1</sup>*Raman Research Institute, C. V. Raman Avenue, Sadashivanagar, Bengaluru, Karnataka 560080, India*

<sup>2</sup>*Institute for Quantum Science and Technology, University of Calgary, Alberta T2N 1N4, Canada*

 (Received 29 January 2020; revised 22 May 2020; accepted 23 June 2020; published 13 August 2020)

Quantum key distribution (QKD) is one of the most important aspects of quantum cryptography. Using laws of quantum mechanics as the basis for security, the key-distribution process makes information theoretically secure in QKD. With the advancement and commercialization of QKD, an end-to-end QKD simulation software is required that can include experimental imperfections. Software of this kind will ensure that resources are invested only after prior performance analysis, and is faithful to experimental capacities and limitations. In this work, we introduce our QKD simulation toolkit *qkdSim*, which is ultimately aimed at being developed into such a software package that can precisely model and analyze any generic QKD protocol. We present the design, implementation, and testing of a prototype of *qkdSim* that can accurately simulate our own experimental demonstration of the B92 protocol. The simulation results match well with experiment; a representative key rate and the quantum bit error rate from experiment is  $51 \pm 0.5$  kbit/sec and  $4.79\% \pm 0.01\%$  respectively, wherein the simulation yields  $52.83 \pm 0.36$  kbit/sec and  $4.79\% \pm 0.01\%$ , respectively.

DOI: [10.1103/PhysRevApplied.14.024036](https://doi.org/10.1103/PhysRevApplied.14.024036)

## I. INTRODUCTION

Quantum key distribution (QKD) is a promising technology that allows two distant parties, popularly referred to as Alice (sender) and Bob (receiver), to share a sequence of secret bits called the “key” [1]. Unlike the state-of-the-art classical public key cryptosystems, developed on Rivest-Shamir-Adleman (RSA) algorithm [2], which depend on computational security, i.e., hardness of factoring, the security of QKD systems rely on the laws of quantum physics where eavesdropping introduces detectable errors [1,3]. The secret key generated from a QKD protocol implementation makes information theoretically secure communication when the message is encrypted with the one-time pad symmetric key algorithm [4,5].

With QKD being commercialized [6,7], sophisticated engineering techniques are being developed [8,9]. To include and evaluate these growing techniques in the realization of QKD protocols, there exists a requirement of finding a cost-efficient approach. A useful alternative to the development followed by testing of an actual experimental implementation is to design a QKD simulation toolkit that can accurately model the experimentation of the existing QKD protocols and deliver the required analysis.

In this paper, we present the design, implementation, and testing of a prototype of a QKD simulation toolkit (*qkdSim*), which we develop, that can simulate an experimental demonstration of the B92 protocol, while taking into account experimental imperfections. The prototype is designed with the vision that in future it will be ultimately developed into a complete software package that can precisely model and analyze any generic QKD protocol.

Earlier theoretical research has been performed on the analysis of QKD protocols [10,11], and to model real-world QKD systems [12]. Early stage research activities were limited to idealistic designs of QKD protocols [13–15] and they also considered only a few optical components [16–20]. Web-based toolkits that simulate QKD basics primarily for educational purposes have also been developed [21–23]. The other development in QKD-related modeling is in the simulation of QKD networks [24,25]. The emphasis in these QKD network simulations is on the network structure and the dissemination of the secure key through various levels of the network and not so much on the accurate simulation of the QKD protocol towards the generation of the key [24–27].

Among the existing QKD modeling frameworks, a proper modular structure and detailed modeling architecture for the design, implementation, and analysis of a full

\*[usinha@rri.res.in](mailto:usinha@rri.res.in)

system-level model can only be found in “qkdX” [12]. Although, qkdX has been deployed as a complete software package with a modular architecture, limited attention has been invested in it towards modeling the actual physical processes including photon sources, from first principles, as well as detection module. Additionally, qkdX leaves behind large room for improvement from the implementational viewpoint in the level of imperfections considered while modeling some of its optical components.

With this perspective, we develop qkdSim that supports quick, easy, and precise simulation of physical processes and evaluation of QKD systems, while considering realistic experimental imperfections at a greater detail. For instance, while in nearly all of the earlier works, the modeling of the input photon in the QKD protocol is considered as a sequence of events, the corresponding input of our simulation is a sequence of time-stamping data that follows an ideal single-photon distribution [sub-Poissonian, antibunching,  $g^2(\tau = 0) = 0$ ]. We also model the background noise as a thermal source, in order to perform more realistic error analysis that complies with the experimental implementation. In addition to that, our detection module includes all imperfections like dead time, quantum efficiency, timing resolution etc. Thus, qkdSim aims to simulate the key-generation rate as well as the quantum bit error rate (QBER) for a wide range of QKD protocols while taking into account an exhaustive list of experimental imperfections. This will help in filling the gap between abstract simulations of QKD implementations and the actual experimental performance. With a more accurate prediction of experimental performance, resources could be more confidently allocated towards real-world QKD implementations.

Our paper is organized as follows. In Sec. II, we discuss some elements related to the historical developments in QKD and thereafter we introduce the general stages as well as the evaluation methodology of a QKD protocol. In Sec. III, we discuss the software process models that are used to develop our QKD simulator. In Sec. IV, we present a detailed description of our free-space-based experimental demonstration of the B92 protocol. In Sec. V, we discuss the various modules that have been constructed for the simulation of the B92 protocol implementation. In Sec. VI, we analyze the modeling of the associated physical processes including single-photon generation and time stamping. In Sec. VII, we highlight the modeling of different optical and electrical components that are used in the experimentation. In Sec. VIII, we analyze and evaluate the results simulated with qkdSim against those obtained from the actual experimental implementation using the same setup. Lastly, in Sec. IX, we provide the concluding remarks and discuss the future research efforts that can be made in this direction. The detailed analytical expressions and methodologies used at various stages in the toolkit, are provided as appendices.

## II. BACKGROUND, GENERAL APPROACH, AND PERFORMANCE ANALYSIS OF QKD SYSTEMS

In this section, we first provide a brief summary of the historical advancements in QKD. Thereafter in the second part, we outline the main steps required to distill a secure key in any QKD protocol. Finally in the third part, we define the criteria to evaluate its security.

The first QKD protocol (BB84) is proposed in 1984 by Bennett and Brassard [28] and then demonstrated using an experimental prototype in 1989 over a 30-cm free-space optical channel [29]. The information theoretic security of the BB84 protocol has been proven [1,3,30,31]. Unlike the use of four nonorthogonal states in BB84, QKD is achieved in 1992 using two nonorthogonal states [32]. This is called the B92 protocol and it is experimentally realized with weak coherent pulses (WCPs) in 1998 [33,34]. Over the years, few protocols such as E91 [35] and BBM92 [36], that perform QKD using quantum entanglement instead of the noncommutativity of quantum operators as their resource, have also been proposed and demonstrated [37,38]. In 1998, QKD is first shown to be secure with imperfect devices, more specifically by the proposal of a self-checking photon source [39]. This work initiated developments in the topic of device-independent quantum key distribution (DIQKD), which not only proved that QKD can be fully secure with minimal fundamental assumptions and untrusted devices [40–43], but also experimentally realized it [44]. While the two common QKD protocols, BB84 and B92, used two and four states, respectively, in 2000 Phoenix *et al.* proposed that the addition of a third state to the B92 protocol would enhance its security considerably [45]. Later in 2005, this three-state QKD protocol is proven to be unconditionally secure [46].

An important aspect of a QKD demonstration is the type of channel over which the key distribution is performed. Two common choices are the free-space and fiber-based ones.

The fiber-based QKD setup in 1998 is able to communicate up to 100 km [34] following the first experimental implementation of QKD in 1992 [47] and in 2003 using a free-space optical link, QKD could be demonstrated only up to 23.3 km [48]. Since then, substantial progress in research has led to the rapid development of optical quantum technologies and over the last few years, many experiments including the demonstration of the feasibility of ground-to-satellite, satellite-to-ground, and satellite-to-satellite QKD [49–58] have been reported as well as commercial QKD devices [59–62] are available. Besides the implementation of long-haul QKD [63], chip-based integrated QKD systems supporting miniaturization have been designed to enable large-scale deployment of QKD into future telecommunication networks [64]. Lately in 2019, an in-field demonstration of the three-state QKD protocol is performed over a fiber link exhibiting a 21-dB

transmission loss in the metropolitan area [65], with the performance of the scheme being evaluated using finite key analysis [66].

The process of generating a secure key in a QKD protocol can be segregated into (i) authentication, (ii) transmission using single photons or WCPs, (iii) sifting, (iv) error correction, and (v) privacy amplification [53,54]. More specifically, in order to obtain the secure key, the raw key is first generated over the quantum channel, followed by information exchange over the classical channel, that leads to the generation of the sifted key. Thereafter steps (iv) and (v) are implemented; where the error correction estimates the error rate and rectifies the erroneously received information bits, while finally privacy amplification extracts a shorter and even more secure final key.

Performance of QKD systems are evaluated with the QBER and the rate of secure communication [34]. A lower QBER indicates higher security, while a high secure-communication rate implies that the transmission link has a good performance. Any information leakage to an eavesdropper about the generated key leads to an increase in the QBER. Therefore, obtaining a high QBER value reduces the rate of secure communication during error-correction stage of the QKD protocol. If the QBER remains below a certain threshold, then the two parties (sender and receiver) can still distill a secure key string by means of error correction and privacy amplification [67]. In other words, if the QBER of the sifted key is above the information theoretically computed threshold for a given QKD protocol then the key is no longer secure. In that case, any privacy amplification technique becomes ineffective. Thus it is imperative for a QKD protocol to ensure a proper upper bound for the QBER if the privacy amplification techniques are to be employed to eliminate any knowledge gained by the eavesdropper.

### III. SOFTWARE PROCESS OF THE QKD SIMULATOR

In this section, we discuss the software-development procedures on which our qkdSim is designed. As a part of this discussion, we also highlight the salient features of those software processes and how they get associated to our final objective.

A software process generally refers to the set of activities that are used to build a software product [68]. In software engineering, the simplified representation of a software process is known as a software process model or process paradigm [68,69]. Although there can be different processes and process models, each of them must satisfy four activities that are fundamental to software engineering [68]. More specifically, these four activities are that (i) the software specification must be well defined, (ii) the software design and implementation must perfectly suit the requirements, (iii) the implemented software must be

validated, and finally (iv) the software must possess the provision to be easily evolved as per user needs.

#### A. Overview of the process model

In this part, we explain the software process model that is used to construct qkdSim. Our QKD simulation toolkit is built using a hybrid process model [70]. From a top-down perspective, our version of the hybrid architectural model consists of two parts: the “Waterfall” development model, which supports linear and sequential design techniques; and the “Agile” development model, which allows iterative and incremental design procedures [69]. In this sense, it can be categorized as a kind of “Agifall” process model [71]. Agifall merges the best of both worlds, by injecting Agile techniques into loose Waterfall design procedures.

As discussed in Sec. II, any QKD protocol grossly is a five-step sequential process, which experimentally involves propagating the signal generated at the source stage through the preparation, transmission, detection, and lastly postprocessing stage. Therefore, in qkdSim, the outer structure, containing the gross five-step QKD design, is developed using the Waterfall process model, which promotes a clear *flow-down* logic scheme. However, keeping in mind the continuous and rapid evolution of technological advances, precision of handling imperfections and experimental nonidealities; the inner software-development architecture for each of the five experimental stages from the modeling of components and physical processes to data-processing methods are developed using the Agile process model, which supports development at a *sprinter’s* pace. In a nutshell, by using an Agifall process model we ensure that the design pattern remains robust and modular at every step of software development to ease future customization challenges.

The key components from the Agifall process model, which support the “best of both worlds” logic, are highlighted as follows [69]. These components are primarily considered for the design of the inner and outer structure of qkdSim.

(a) Modeling, usability, and organization can be coherently applied on the outer structure of all QKD protocols in general, so the choice of Waterfall process model is suitable.

(b) Incremental updates in small update cycles allow rapid improvement of basic experimental techniques and components, so the choice of Agile process model is appropriate for designing the inner parts of the implementation stage.

#### B. Salient features of the process model

In this part, we first present the salient stages of our version of the Agifall process model in Fig. 1 [69].

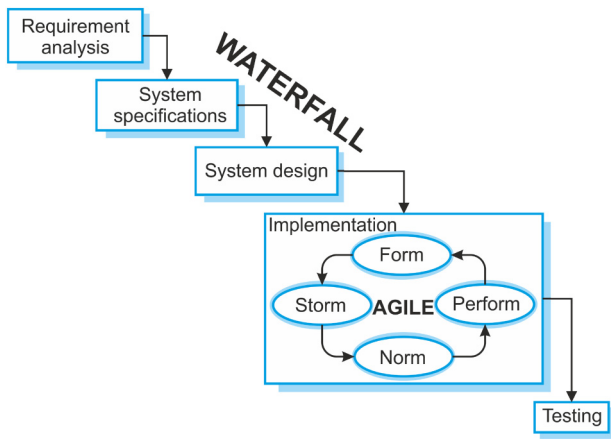


FIG. 1. Schematic of the hybrid process model used to build *qkdSim*. The outline of the simulator is developed on the Waterfall model, while its implementation-based intricacies are modeled using the Agile design procedures.

Thereafter, in the following subsections, we discuss in details the five major steps of the top Waterfall structure, including the requirement analysis performed for developing the toolkit, followed by the specifications of the system, the design process, and finally the implementation and testing stage.

### 1. Requirement analysis

The first stage of our architectural model is “requirement analysis” where the goals and constraints of the software are discussed by consultation with the users. These user requirements then serve as a part of the software specifications. The probable users for the toolkit are identified as experimental physicists and engineers working in the domain of quantum communication and cryptography. The users are required to provide necessary inputs for the toolkit to perform. The requirements of the user that the toolkit will be able to fulfil are analyzed with the help of an user story. The principle user story for the toolkit is

I, as a QKD experimentalist, want to simulate an experimental implementation of a QKD protocol and evaluate its performance through the quantification of the key rate, the QBER and the key symmetry.

The toolkit is also required to provide flexibility to the users to simulate any experimental setup and be able to vary the choice of various components in the setup, so that in principle it may be applicable to any QKD protocol. The toolkit is built keeping in mind that the user requirements may vary with time and hence the toolkit should be extendable to accommodate modification of the various modules that form the system. An important aspect of developing the toolkit is the consideration of the security of the protocol being simulated. Whereas the toolkit does not accommodate any dedicated security analysis of the

given protocol, the algorithms used for calculating QBER are developed to ensure that the value is maintained below known security thresholds for the protocol being implemented. Following the stage of requirement analysis, we move on to the stage where we identify the specific inputs and outputs of the simulation toolkit such that the user requirements are satisfied.

### 2. System specifications

The second stage called “system specifications” is used to identify the inputs and outputs from the user requirements. Figure 2 represents the inputs and outputs of the simulation toolkit. At this stage, we form the overview of the system and identified the functions that the system is enabled to perform without delving into the question “how?.” The inputs take into account the integral components of implementation of a QKD protocol and various choices that the experimentalists possess.

The identification of the general inputs and outputs to the toolkit leads us to the design stage where we develop the architecture of the toolkit.

### 3. System design

At the third or the “system design” stage, the user requirements along with the list of inputs and outputs have to be associated with the hardware and software units to establish an overall system architecture of the toolkit. The independent layers that will form our desired system must contain the level of abstraction and flexibility that we want to provide to the user. An optimal architecture enables us to develop the system in an iterative approach and enhance the system by considering further real-world imperfections.

In our design presented in Fig. 3, the independence of each layer of the architecture exists with respect to their development and modeling. However, there is a hierarchical dependence among the layers in forming the components of the experimental implementation to be simulated. Each layer of the architecture can be explained as follows.

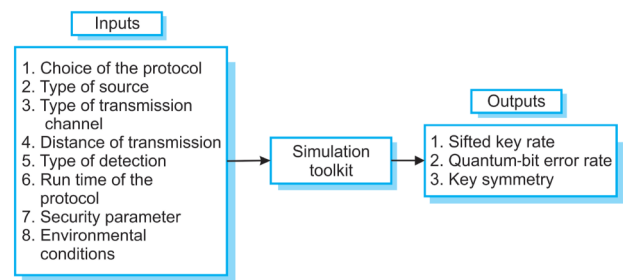


FIG. 2. System specifications of the simulation toolkit.

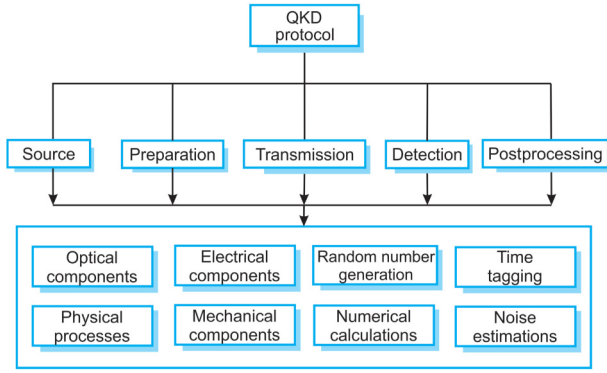


FIG. 3. Architecture of the simulation toolkit.

(a) The QKD protocol chosen by a user forms the top layer of the system. The choices regarding the other inputs to the toolkit will be dependent on the choice of the protocol. As an example, for an entanglement-based QKD protocol, the choice of source is restricted to entangled photon sources. Thus, depending on the choice of the protocol, the respective modules from the second layer will be chosen.

(b) The second layer is formed by the different modules that will be used to simulate the corresponding aspects of the experiment. The modules can be further subdivided into different types to accommodate for the various possible requirements. For example, the source module can accommodate different types of sources such as heralded single-photon source, weak coherent pulse source, entangled photon source, etc. Thus, the user will be able to choose the specific type corresponding to each of the modules as required for the simulation.

(c) The bottom layer is formed by the submodules that are modeled and are used to form the structures of the modules in the upper layer. Each of the modules is made functional by the flow of logic through these submodules. The submodules contain various modeled physical components and processes that can be chosen by the user as required.

The different layers developed in the architecture form the basis for the next stage, which is the implementation, in an iterative manner following the Agile model mentioned previously.

#### 4. Implementation

In this work, an experimental demonstration of the B92 protocol is simulated by using the architecture described in the previous subsection. The simulation toolkit is in a prototype stage, and the modeled physical components and processes are limited to the current aim of simulating the specific implementation scenario. The accuracy of the prototype is limited by the various assumptions that are

TABLE I. List of modeled physical components.

Beam splitter	Polarizing beam splitter	Phase retarder
Single-mode fiber	Single-photon detector	Time-correlated single-photon counting module (TCSPCM)
Periodically poled potassium titanyl phosphate (PPKTP) crystal	SMA cable	Band-pass filter
In-lab free-space channel	Lens	Fiber coupler

considered as well as the set of imperfections that are taken into account while modeling the physical components.

Tables I and II categorize the different physical components and processes that are modeled and tested while considering realistic imperfections. The design methodology employed for modeling these two categories of elements is the Agile development procedure. Such a choice allows the required flexibility for quickly incorporating the future technological advancements, inclusion of further imperfections and thus, improving upon the precision that can be achieved.

The simulation toolkit is currently implemented in Python programming language and interacts via a command user interface. Before we move on to the analysis and testing of the implemented prototype of the simulation toolkit, we make the following observation that besides B92 other similar prepare and measure protocols, such as the BB84 protocol based on SPDC-based heralded single-photon source can be easily accommodated using the different modules available in the current implementation of qkdSim with little modifications. Even for enabling the simulation of other categories of the QKD protocols, we would just need to develop some additional modules and fit them into the pre-existing modular structure of the Agifall framework. For example, in order to simulate a differential phase-shift keying-based QKD protocol, we would need to develop a source module that can simulate a weak coherent pulse, which in fact is simpler in comparison to simulation of a SPDC source. Thereafter, we can just call the pre-existing physical component modules such as beam splitters, fibers, fiber couplers, wave retarders etc., as per our requirement. Lastly, even the parts

TABLE II. List of modeled physical processes.

Type-II SPDC process	Propagation of single photons	Fiber coupling
Single-photon detection	Time stamping	Background estimation and detection

of the detection modules, i.e., detectors and time-correlated single-photon counting submodules, can be mostly reused, with only simpler modifications required on the detector module, when we simulate continuous variable QKD protocols. Similarly, the simulation of a measurement-device-independent QKD protocol would require an additional entangled photon-source module. Such features will be released in the next iterations of qkdSim.

### 5. Testing

At this stage, we test the various modules and submodules that are constructed as a part of the prototype, to simulate the experimental demonstration of the B92 protocol. Each submodule is tested independently to verify whether expected outcomes are obtained. The results from the testing of submodules listed in Table I are compared with experimental outcomes from characterization of the corresponding physical components. The submodules of the physical processes are tested by matching the results with experimental observations corresponding to the same processes. The outputs from each of the modules are then compared with the corresponding sections of the actual experimental setup. The overall verification and testing of the performance of the prototype is done by comparing the simulated outputs with the experimental results obtained from the in-lab free-space demonstration of the protocol. In Sec. IV we provide a detailed discussion on the procedure of our experimental implementation, while the results obtained from the simulation and the experiment are presented in Sec. VIII.

## IV. EXPERIMENTAL DEMONSTRATION OF THE B92 PROTOCOL IN FREE SPACE

In this section, we describe in details our free-space-based experimental realization of the B92 protocol. In the first part, we provide a brief overview of the B92 protocol and a quick history of its various experimental implementations. In the second part, we discuss in details the experimental setup that we use to implement the B92 protocol. Lastly, we highlight the general procedure and the associated techniques, that we develop and use to analyze our experimental data, i.e., estimate the key rate, QBER, and key symmetry for our experimental demonstration.

### A. General procedure of B92

In a standard B92 protocol using polarization encoding, Alice sends Bob a stream of single photons, where the polarization state of each of the photons is randomly selected between any two nonorthogonal polarization bases, say,  $|a\rangle$  and  $|b\rangle$ , where  $\langle a|b\rangle \neq 0$ . These two polarization states are encoded with binary 0 and 1, respectively. When these photons reach Bob, he randomly and independently selects between two projection operators

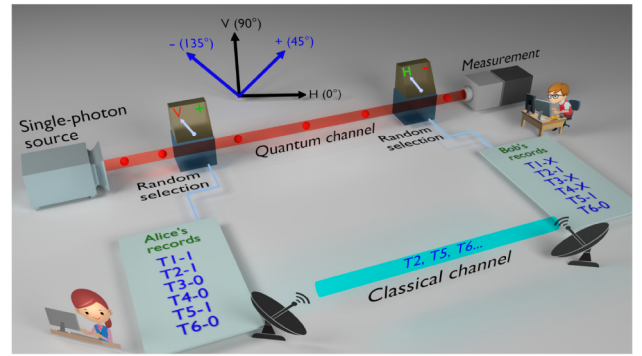


FIG. 4. Schematic of the B92 protocol based on polarization encoding.

$(I - |a\rangle\langle a|)$  or  $(I - |b\rangle\langle b|)$  for each photon, and projects on it. Note that the operator  $(I - |a\rangle\langle a|)$  always gives a null result when it operates on  $|a\rangle$ . Similarly,  $(I - |b\rangle\langle b|)$  always gives a null result for  $|b\rangle$ . A schematic of the B92 protocol based on polarization encoding is presented in Fig. 4. Here, Alice randomly selects the polarization state for each photon coming from a single-photon source to be either vertical (V) or diagonal (D;  $+45^\circ$  with respect to the horizontal polarization basis) and sends it to Bob. She also assigns bit values to all the photons based on their polarization and records them sequentially (T1-1, T2-1, T3-0, etc.). Bob randomly selects his measurement basis to be horizontal (H) or antidiagonal (A;  $-45^\circ$  with respect to the horizontal polarization basis). Bob only considers those events where his measurements give positive (or non-null) results (T2, T5, T6, etc.) and announces only the occurrence (or timing) of these positive events in a public communication channel (that could be prone to eavesdropping), once all the photons are received. Bob never shares the choice of measurement operations for these positive events. Based on the announcement, Alice only keeps those bits that generate positive events in Bob's setup. Thus Alice and Bob generate and share an identical, secure key.

There are many experimental implementations of the B92 protocol. All these experiments can be broadly categorized based on three classification parameters, i.e., type of encoding, medium of transmission and type of source of photons. Though the original B92 protocol is based on phase encoding and many later experiments [72,73] followed similar logic, a number of experiments have also been performed based on polarization encoding [74–77]. In terms of the medium of transmission, there are experiments in free-space transmission channel [78–80] as well as fiber-based channel [76,81,82]. With respect to the type of source, interestingly, most of the experiments to date use WCPs as single photons [73,82–84], and only a handful of experiments have considered heralded single photons generated from a spontaneous parametric down-conversion

(SPDC) process [77,80]. In fact the paucity of existing literature on the B92 protocol motivates its choice as the first protocol that we choose to evaluate using qkdSim as, conceivably, it could reveal fresh insights against which we can test our simulator.

In our implementation, we use heralded single photons generated using the SPDC process. Our choice of source is connected to the security aspects of the protocol and enhances the same. This is explained when we discuss about postprocessing in the following subsection. The photons have been encoded in polarization degree of freedom, and transmitted in a free-space channel inside a lab environment.

## B. Our experimental implementation of the B92 protocol

In this part, we provide a detailed description of the key resources and the various stages of our experimental implementation. In the process of analyzing the different stages of our setup and its associated components, we also identify the different sources of noise and imperfection, that can potentially affect our measurements, and highlight how our resources help to mitigate them.

We use spontaneous parametric down-conversion as a source of heralded single photons. While there are a very small number of SPDC-based implementations of the B92 protocol in the literature, our implementation is significantly different from the existing ones as discussed below.

The schematic in Fig. 5 has details on the components in the source. A blue diode laser of wavelength 405 nm (Cobolt 08-NLD) pumps a PPKTP crystal continuously with 30-mW power. The polarization of the pump beam

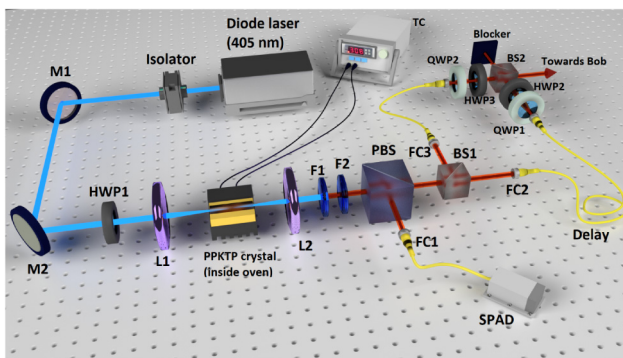


FIG. 5. Schematic of the heralded single-photon source and Alice's module. M1, M2: dielectric mirrors; HWP1: half-wave plate for pump light; L1: focusing lens; L2: collimating lens; F1: long-pass filter; F2: band-pass filter; PBS: polarizing beam splitter; FC1, FC2, FC3: fiber couplers; BS1, BS2: 50:50 nonpolarizing beam splitter; HWP2, HWP3: half-wave plates; QWP1, QWP2: quarter-wave plates; SPAD: single-photon avalanche detector; TC: temperature controller.

is kept horizontal (H), by using a half-wave plate (HWP1). The PPKTP crystal is placed inside an oven (RAICOL), which is connected to a temperature controller. We find that at 40 °C, photon pair generation is optimal for colinear, degenerate, type-II SPDC process in the crystal, wherein a horizontally polarized pump photon of wavelength 405 nm down-converts to a horizontally polarized signal and a vertically polarized idler photon, both with peak wavelength at 810 nm. In order to maximize the photon pair-generation rate, two lenses of focal lengths 100 mm (L1) and 50 mm (L2), respectively, are used; L1 focuses the pump beam at the center of the crystal and L2 is used to collimate the beam. Due to the colinear configuration, both photons in each pair traverse the same path as followed by the residual pump beam. A long-pass filter F1, placed after the crystal, blocks most of the pump beam and allows only the red photons to go through. A band-pass filter F2, allows only photons with wavelength close to 810 nm to go through. A polarizing beam splitter (PBS) separates the two photons in each pair, where “H” polarized photon goes to the transmitted arm and “V” polarized photon goes to the reflected arm of the PBS. A coupler (FC1), placed in the reflected arm, couples “V” polarized photon to a single-mode fiber that is connected to a single-photon avalanche detector (SPAD; COUNT-T-100). So, detection of a “V” polarized photon heralds the “H” polarized photon of the same pair. This is how heralded single photons are generated.

*Alice substation:* The components after the PBS in Fig. 5 constitute the Alice substation of the QKD implementation. Once each heralded single photon is generated, Alice randomly selects between two nonorthogonal polarization states, vertical (V) and diagonal (D), by passing the photon through a 50:50 nonpolarizing beam splitter (BS1), whose transmitted arm has a HWP (HWP2) that rotates the input H polarization to D, and reflected arm has a HWP (HWP3) that rotates the input H polarization to V. Quarter-wave plates (QWP2 and QWP3) are also placed along with HWPs, in order to minimize ellipticity in the polarization. The use of such a 50:50 BS for random selection gives us quantum randomness, which is necessary for the security aspects of the protocol. However, this comes with the caveat that Alice herself does not have knowledge about the polarization state of the photon that she sends to Bob.

In order to resolve this, we come up with a fresh solution wherein Alice makes use of two known properties of SPDC, namely, heralding process and the probabilistic nature of pair generation, where photon pairs are generated randomly in time. If Alice now randomly selects a subset from generated photon pairs and applies a fixed time delay to them, it becomes impossible for an eavesdropper to determine if the photon pair is from the selected subset or the rest, just by looking at the arrival time. So, Alice uses two single-mode fibers of different lengths in the two output arms of the BS1. Photons that traverse the

transmitted arm and become “+45°” polarized later have a fixed time delay ( $\Delta t$ ) as compared to photons that traverse the reflected arm and become “V” polarized. Alice also records the arrival time of each heralding photon detected in her detector using a time tagger (HydraHarp 400) connected to the detector. This is crucial in determining whether the corresponding pair photon is delayed or not, before it is sent to Bob, and hence its polarization state. Thus, by introducing a time delay in the path of one of the photons and knowing the arrival time information of its partner photon (heralding arm), Alice is able to determine the polarization state of the photon that is sent to Bob.

In order to remove any distinguishability in the spatial degree of freedom, outputs from both single-mode fibers recombine at another 50:50 BS (BS2), and only one output arm of the BS2 is used to send both “V” and “+45°” polarized photons to Bob.

*Bob's substation:* In Bob's part of the experimental architecture as shown in Fig. 6, a 50:50 beam splitter (BS) is placed in the path of the incoming photons, where each photon has 50% probability to go to the transmitted arm and 50% to go to the reflected arm. In the transmitted arm of the BS, a polarizing beam splitter (PBS2) is placed and a fiber coupler (FC2) collects any photon that transmits through the PBS and sends it to a single-photon detector (SPAD2). So, in this arm, arm only “+45°” polarized photons have 50% probability to get detected while “V” polarized photons have 0% probability. In the reflected arm of the BS, a similar combination of PBS (PBS1) and fiber coupler (FC1) is placed with an additional half-wave plate (HWP) just after the BS. This HWP converts D photons to V photons and similarly H photons to D photons. So, only the “V” polarized photons sent by Alice pass through the PBS and get detected in this arm with a probability  $\frac{1}{2}$ , but no “+45°” polarized photon is detected. So, to summarize, any detection in the transmitted arm of the

beam splitter definitely means that the photon is “+45°” polarized (assigned bit value 0), and any detection in the reflected arm definitely means that the photon is “V” polarized (assigned bit value 1). Similar to Alice, Bob also records photon time-stamping data from the two detectors at his end.

*Postprocessing:* In the postprocessing stage, Bob shares only his time-stamping data publicly, but does not indicate which time-stamping data comes from which detector. Alice compares her time-stamping data with Bob's data and measures time difference for all detected photon pairs. Let us say the distance between Alice and Bob is  $d$ , then the time difference should be ideally  $T = d/c$ , where  $c$  is the speed of light. For the “+45°” photons, where Alice applied additional time delay  $\Delta t$ , time difference becomes  $T + \Delta t$ . Alice considers those events where time difference is closer to  $T$  (within a small time window around  $T$ ) as bit value 1 (in this case, “V” polarized photons are sent by Alice to Bob), and considers those events where time difference is closer to  $T + \Delta t$  as bit value 0 (“+45°” polarized photons are sent in this case).

Alice then sends back Bob's time-stamping data by omitting all those events that could not make it to Alice's final key. Based on this, Bob generates his final key, which concludes the final key-generation process.

This brings us to an important comment regarding the use of heralded single photons for our implementation and its ramifications. For the purpose of security of the generated key, generation of ideal single photons (with Fock state  $|1\rangle$ ) is essential. In the case of a multiphoton source, an eavesdropper may apply a photon number splitting (PNS) attack. One way to verify the single-photon distribution is to look for the antibunching property where the probability of generating two consecutive photons within the coherence time is negligible. For this purpose, we can measure the normalized second-order coherence or  $g^2$  by performing a Hanbury Brown and Twiss (HBT) type experiment. For an ideal single-photon source  $g^2(\tau = 0) = 0$ , where  $\tau$  is the time interval between the generation of two consecutive photons.

In a real-world experiment, there are stray photons that get detected in Bob's detection module along with single photons that are sent by Alice. There are also other sources of noise like dark noise of the detector, electrical noise, etc. A single-photon detector cannot distinguish noise from the actual signal. These noises increase the quantum bit error rate in the generated key. For noise cancellation, heralded single-photon source plays an important role. In a heralded photon source, correlated photons are always generated as a pair. Detection of one photon in each pair ensures the presence of the other photon. Therefore, Alice and Bob postselect only those events where Alice detects one photon and Bob detects the other photon of the same pair (i.e., coincident events) and consider them as part of the signal.

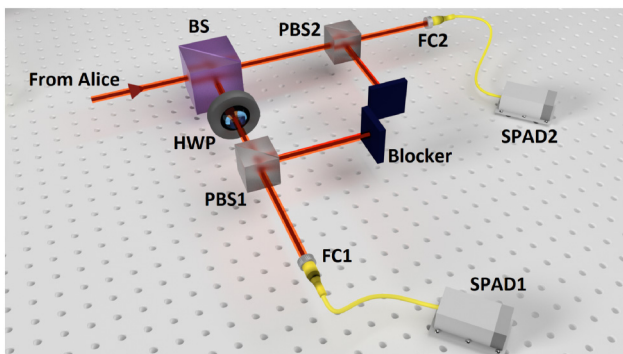


FIG. 6. Schematic of Bob's module. BS: 50:50 nonpolarizing beam splitter; HWP: half-wave plate; PBS1, PBS2: polarizing beam splitters; FC1, FC2: fiber couplers; SPAD1, SPAD2: single-photon avalanche detector.



### C. Data analysis

In the data-processing stage, we measure three important parameters that are the quantifiers of the performance of the QKD experimental setup. These parameters are key rate, quantum bit error rate, and asymmetry of the key.

For key rate, we measure the average number of bits in the sifted key (including error bits) generated per second. In the B92 protocol, sifting includes postprocessing of the dataset, where Alice and Bob selects only those events where Bob’s detectors show positive outcome and the detection occurs within some predefined time window. After the completion of the protocol, Alice and Bob both have a key that should be identical in the absence of any noise in the channel and eavesdropping activity. We measure the number of error bits by comparing each value of the bits for the same bit position in the two keys. The number of error bits divided by the total key length gives the QBER. Asymmetry (or key symmetry) quantifies the disparity between the number of 0 and 1 bits in the final error-free key shared by Alice and Bob.

In order to report the average key rate, we run the protocol for 10 s and repeat the same for 20 iterations. The final key rate is then averaged over the 20 key length values. In order to ensure that we choose a runtime of the protocol such that the standard deviation (SD) by mean ( $M$ ), i.e.,  $SD/M$  of the reported average key rate is very small, we collect data continuously for a longer time duration (say, 100 s), apply the bootstrapping technique (as discussed in Appendix A), and obtain a  $SD/M$  plot (refer to Fig. 29) as a function of the runtime for a fixed number of iterations. We find  $SD/M$  for 10-s runtime and 20 iterations to be 0.016%.

In order to measure key length, QBER, and asymmetry from each dataset, we apply two types of optimization methods, namely A & B, on every dataset. In both methodologies, at the beginning Alice’s and Bob’s recorded time-stamping data are compared and plotted as a function of time difference between the two. The schematic in Fig. 7 shows two distinct coincidence peaks due to the time delay of around 10 ns introduced in Alice’s setup. The first peak (blue) represents those coincidence events where Alice sent a “V” polarized photon and Bob measured correctly. The second peak (red) represents those coincidence events where Alice sent “+45°” polarized (delayed) photon and Bob measured correctly. The extension of the red curve under the blue curve represents those events when Alice sent a “V” photon, but Bob measured it wrongly as a “+45°” polarized photon; due to the noise introduced by optical components as well as the transmission channel. Similarly, the extension of the blue curve under the red peak represents those events when Alice sent a “+45°” photon, but Bob measured it wrongly as a “V” polarized photon. We fix a certain time window around both peaks ( $W_{11}$  to  $W_{r1}$ , and  $W_{12}$  to  $W_{r2}$ ) and measure the area under the curves. The sum of the total area (i.e., both signal and noise portions) under the blue and the red curve for their consecutive time windows gives the key length. The sum of the total area under the red curve from  $W_{11}$  to  $W_{r1}$ , and the blue curve from  $W_{12}$  to  $W_{r2}$  gives the number of total error bits or noise (that contribute to the QBER). In context to our schematic shown in Fig. 7, the definitions of key rate and QBER ( $Q_{err}$ ) can be analytically expressed as

$$\text{key rate} = \frac{[\text{signal} + \text{noise}] \text{ part of both curves}}{\text{runtime of the protocol}} \text{ (Hz)}, \tag{1a}$$

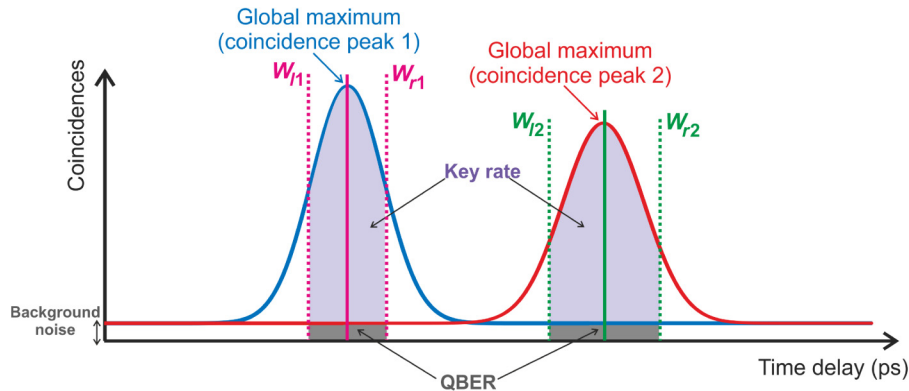


FIG. 7. Simplified schematic of the output of two independent coincidence detection: Alice and Bob’s R basis (coincidence peak 1 in blue) versus Alice and Bob’s D basis (coincidence peak 2 in red). The background noise zone indicated below the flat portions of the blue and red curve represents the unwanted coincident detection from stray light sources, uncorrelated signal and idler photons, leaked pump photons, and dark noise of the photodetector. The symbols  $W_{11(2)}$  and  $W_{r1(2)}$  represent the left and right markers of the time window around the maximal coincidence point for coincidence peak 1 (2). The total coincidences within the chosen window around the central maximum from both curves contribute to the error-free “key rate” (or signal—marked in purple); while those within the background noise zone contribute to the “QBER” (or noise—marked in gray). Note that in reality, the coincidence curves are not typically smooth functions and contain a lot of kinks (local optimal points) around a central global maximum.

$$Q_{\text{err}} = \frac{\text{noise part of both curves} \times 100}{[\text{signal} + \text{noise}] \text{ part of both curves}} (\%), \quad (1b)$$

$$\text{key symmetry} = \frac{\text{signal part of either curve} \times 100}{\text{signal part of both curves}}. \quad (1c)$$

Ideally, for secure key generation the probability of obtaining any key string of  $N$  key bits among the  $2^N$  set of possible key strings should be equal; i.e., any key can be generated with the probability of  $\frac{1}{2^N}$ . If this ideal case is to be realized then all the optical components should behave perfectly, i.e., symmetric beam splitter should have exactly 50% probability of both transmission and reflection, all threshold detectors must have equal efficiency, etc. This would lead to both coincidence peaks also being identical or in other words, the number of 0's and 1's in the final key would then be equal. However, in the real experimental scenario the two peaks could be different. In order to investigate this further, we generate a large number of key strings and find that all of them exhibit asymmetry between the number of 0's and 1's, away from the ideal requirement of 50:50. One of the possible causes for this asymmetry could be a mismatch between the efficiencies of the detectors used in Bob's module. However, we verify independently that the two detectors used in Bob's module have similar efficiency, in the sense that on interchanging them, we observe nearly identical counts. This indicates that the bias (quantified by the asymmetry) is primarily introduced before the detectors, mostly by the beam splitter and other optical components placed in Bob's arm, rather than the detectors. If the device is generating perfectly random outputs, we would have obtained a binomial distribution peaked at 50:50 (i.e., the number 0's or 1's in the generated key string would then be exactly half of the key length).

Nevertheless, in our case, we find that the distribution, in general, is peaked at different values ranging from  $57.59 \pm 0.29 : 42.41 \pm 0.29$  to  $51.49 \pm 0.32 : 48.51 \pm 0.32$ . This bias in the final key can be advantageous to the eavesdropper unless a commensurate step to mitigate this is included in the postprocessing algorithm. More particularly, for generating a perfectly secure key, the probability of a certain key bit to be "0" or "1" should be equal for all the bits in the key string. In this work, along with QBER, the role of asymmetry in the key string is also investigated with respect to security loopholes of a QKD protocol.

As mentioned earlier, in our implementation, we devise two different optimization strategies: "A" and "B" (refer to Appendix B for the detailed procedure of the two strategies) to account for this asymmetry in key string due to device imperfection. In both strategies, our aim is to maintain the QBER below a certain threshold. Additionally, strategy A attempts to counter the biasing issue by compensating for the asymmetry in the key bits; while strategy B does not offer any such compensation on key

asymmetry but enhances the SNR ratio. As a cost for the additional compensation, the former technique produces a lower key rate than the latter. Nevertheless, both strategies have their advantages and shortcomings that are highlighted next.

In strategy A, we optimize the QBER obtained in each run of the experiment individually and impose a symmetry of nearly 50:50 among the key bits. More specifically, the whole purpose of this optimization method is to find the value of the two coincidence time windows or the position of  $W_{11}$ ,  $W_{r1}$ ,  $W_{12}$ , and  $W_{r2}$  (as shown in Fig. 7), such that the QBER remains below the threshold value (4.8%) [67] and the key is symmetric. However, this approach is insecure from the perspective that it imposes a 50:50 key symmetry. In this strategy, ensuring a 50:50 ratio also leads to asymmetric deletion of some bit values and thus lowers the final key length. We recall that the previously stated key-generation probability of  $1/2^N$  in the ideal case, can be associated to Eve's probability of guessing the key string. Hence using strategy A, her guessing probability has now been increased to  $[(N/2)!]^2/N!$ , while reducing her ability to gain additional information due to any asymmetry in the key bits.

In strategy B, we calibrate the time-window markers on the two curves based on maximization of the SNR, while ensuring that the duration remains equal for both of them. More particularly, we move the detection-window markers, in and out around their respective coincidence peaks, to obtain the maximum (error-free) key length within the QBER bound and ensure that the two windows have an equal span, for each key string separately. As a consequence, we find that the optimized key length increased, while the key symmetry deteriorated from 50:50.

In Fig. 8, we exemplify the key symmetries obtained for the generated (error-free) key strings and compare the unoptimized and the two optimized key strings through the distribution of their individual key bits. For this analysis, we consider one of the 20 datasets measured, during the daytime, with the 30-mm crystal. For this representative analysis, we report the distribution of zero bits, which ideally should be equivalent to that for the one bits in a given key string.

In the first case, i.e., in Fig. 8(a), we report the distribution of zero bits obtained in the error-free key string generated by considering a 3-ns time window around the left (blue) and right (red) coincidence peaks (shown in Fig. 7). The duration of 3 ns is chosen to cover the maximum coincidences available on the two curves. This approach being an unoptimized one, produced an error-free key rate of about 73.6 kHz, at the expense of a QBER of 7.27%. The obtained error-free key rate can be inferred from the red subplot on the left, where each value on the  $x$  axis contains 100 key bits of the generated key string. The blue histogram subplot on the right illustrates that the distribution of zero bits in the key string is indeed a binomial peaked at

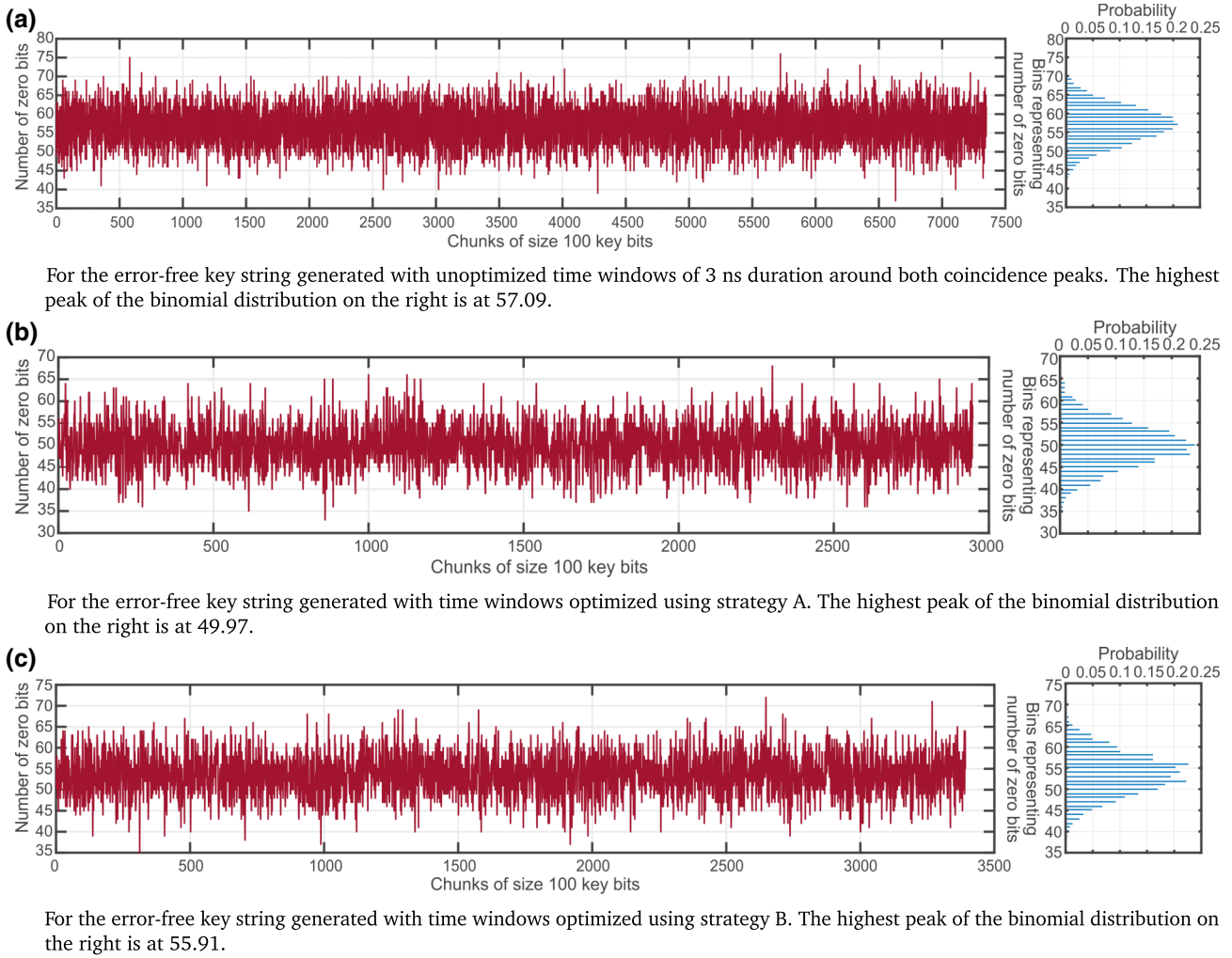


FIG. 8. Distribution of zero bits in the error-free key string obtained with three different strategies: (a) unoptimized, (b) optimized via strategy A, and (c) optimized via strategy B. For all strategies, the left plot represents the number of zero bits present in sequential chunks of size 100 key bits within the key string, and the right plot displays the corresponding histogram of the frequency of zero bits found within those chunks (bins). The key bits plotted on the left plots are collected over a measurement time of 10 s. This representative analysis is performed on one of the 20-day datasets measured with the 30-mm crystal. (a) For the error-free key string generated with unoptimized time windows of 3 ns duration around both coincidence peaks. The highest peak of the binomial distribution on the right is at 57.09. (b) For the error-free key string generated with time windows optimized using strategy A. The highest peak of the binomial distribution on the right is at 49.97. (c) For the error-free key string generated with time windows optimized using strategy B. The highest peak of the binomial distribution on the right is at 55.91.

57.09, indicating that the ratio of 0 to 1 bits is quite asymmetric. More particularly, when we bin the key bits into sets of 100 over the entire key string, then we find that the probability of obtaining a key symmetry of approximately 57 : 43, becomes the maximum, within those bins. This also validates our earlier claim that the distribution of key bits, in general, can possess asymmetric peak values ranging as high as  $57.59 \pm 0.29$  (averaged over 20 datasets). In the next two cases, we contrast the unoptimized analysis against those obtained with the two optimization strategies: A and B. More specifically, in Figs. 8(b) and 8(c), similarly we report the corresponding distribution of zero

bits in the key strings that are generated using the optimization strategies A and B, respectively. From a comparative perspective, we can observe that in Fig. 8(b) the peak of the binomial distribution occurs at 49.95, i.e., the most probable case is free from any significant bias; while in Fig. 8(c) a considerable bias (or asymmetry) exists for the peak of the distribution being at 55.91. It is important to note that the distribution of the zero bits in the generated (error-free) key string for strategy A, in this analysis, comes to being a binomial distribution instead of the otherwise expected delta function peaked at about 50:50, since in the said algorithm we are optimizing the key symmetry

over the entire string and not over the individual chunks being plotted here.

For strategy B, being free from the imposition of any symmetry constraint on the ratio of 0's to 1's bits in the optimized key string, there occurs no increase in Eve's guessing probability unlike that found in strategy A. However, strategy B suffers from possible information leakage to Eve due to the existing bias arising from asymmetry. In [85], the authors have shown how any possible security loophole due to detector efficiency mismatch can be mitigated by including relevant steps in the privacy amplification part of the postprocessing algorithm, thus ensuring that such a bias can also be accounted for, enabling a secure reconciled key. In our example, the mismatch that happens due to the effective asymmetry present in our optical components can also be recast into a similar detection efficiency mismatch problem, thereby enabling a secure reconciled key as shown in Ref. [85]. It would be an interesting open problem to see how modifications could be made to privacy amplification algorithms to account for the possible security loophole posed by optimization strategy A.

We remark here that while the symmetry aspects bring about different security implications as discussed above, the objective of our work is to model practical QKD as accurately as possible, while constraining the QBER below the known security threshold for the given protocol and not comment on any detailed security analysis of the said protocol. In the current B92 implementation, we thus constrain the QBER to be below 4.8% [67] in both optimization strategies that we implement.

## V. SIMULATION TOOLKIT

In this section, we discuss the principle aspects and the current stage of the implementation of the simulation toolkit. Following an overview, we go on to discuss the various assumptions that are considered while simulating the experimental demonstration. In the later subsections, we provide a detailed discussion on the various modules that comprises the toolkit.

### A. Overview

We simulate the experimental demonstration of the B92 protocol discussed in Sec. IV using the simulation toolkit “qkdSim” discussed in Sec. III. Different modules are developed for the respective choices corresponding to the source, detection, and transmission components used in the actual experiment. Table III lists the various choices for the general inputs to the toolkit and Fig. 9 shows the interconnection of the various modules developed for the toolkit.

The modules are interconnected for the flow of logic and mimics the path of the photons in the actual experimental setup. The output from the source module, based

TABLE III. Choice of inputs.

Choices	Inputs
Type of protocol	B92 QKD protocol
Type of source	Type-II colinear degenerate SPDC source
Type of transmission channel	Free space
Distance of transmission	2 m
Type of detection	Fiber-based single-photon detectors and TCSPCM
Protocol runtime	20 runs for 1 s each
Security parameter	QBER threshold
Environmental conditions	In-lab (day time and night time)

on type-II colinear degenerate SPDC process, is passed to Alice's preparation module. The signal states are encoded for transmission and relayed to the transmission module while the heralding states are passed to Alice's detection module. The transmission module simulates the transmission of single photons over the quantum channel between Alice and Bob and the detection of the received signal states is simulated in Bob's detection module. The outputs from Alice's and Bob's detection modules are fed into the classical postprocessing module, which then gives as output the sifted key rate, QBER, and the key symmetry.

Each of the modules is constructed with the help of various submodules corresponding to the various physical components and processes that play important roles in the experimental implementation of the protocol. The inputs to the modules can be categorized as user inputs, set parameters, and outputs obtained from preceding modules. The user inputs refers to certain choices made by the user whereas the set parameters refers to the specification of the various components that the setup consists of. Before we discuss in detail the structure and working of each of the modules, we discuss the various assumptions that are considered for the simulation, in the following subsection.

### B. Assumptions

Though the simulation toolkit takes into account various nonideal aspects of experimental implementation of a QKD protocol, it is not exhaustive. Thus it is essential to provide a detailed list of assumptions that are considered while implementing the architecture at each stage of the

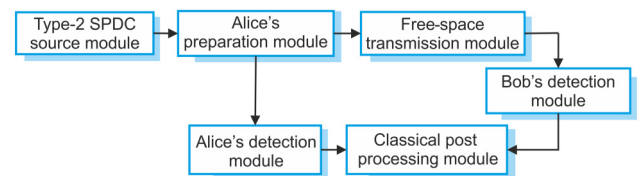


FIG. 9. B92 simulation structure.

iteration. The assumptions considered for the simulation of the B92 protocol are listed as follows.

1. The time stamps associated with each of the photon pairs are evolved through the experimental setup comprising different optical components. However, in the sections of the experimental setup where the photon pairs travel identical path lengths or the possible paths that can be traversed by a single photon have identical length, the time taken to travel is not accounted for. For example, the detectors in Bob's detection module are assumed to be positioned equidistant from the BS (Fig. 6).

2. The pump laser output is assumed to be a symmetric Gaussian beam. The generated photon pairs are also considered to have similar beam properties as that of the pump laser. Thus each photon is associated with a Gaussian distribution for the intensity.

3. It is assumed that the alignment of the optical and mechanical components in the experimental setup is achieved up to maximum precision, limited only by the error introduced by the least count of the screws of the mounting components and stages.

4. In the simulation of the type-II colinear degenerate SPDC process, the frequency linewidth of the pump laser is assumed to be extremely narrow and hence, the frequency distribution of the signal and idler photons is not taken into account. Additionally it is also assumed that the pair generation takes place only at the center of the crystal and the crystal medium is lossless.

5. In the experiment, it is observed that the important parameters of the system such as the laser power, temperature of the crystal etc. does not fluctuate significantly over the period of time for the data acquisition. Thus, in simulation, we assume the system to be time invariant and all the parameters are assumed to be constant over the runtime of the simulation.

6. Any effect in the phase or polarization of the photons due to the transmission over free space and optical fiber channel is neglected.

7. No eavesdropping strategy or attack is considered for the simulation. A security parameter that corresponds to a threshold QBER, derived based on the protocol and independent to the simulation, is taken as an input to the system.

We discuss the various assumptions that are considered in the implementation of the simulation toolkit and now we discuss each of the modules shown in Fig. 9 in order of the path followed by the photons in the actual experimental setup. The source module is discussed first followed by the modules corresponding to the preparation, transmission, and detection of the signal photons and concluding with the postprocessing module. For each of the modules, a brief introduction is followed by a brief discussion on the inputs and outputs of the modules, the structure of the

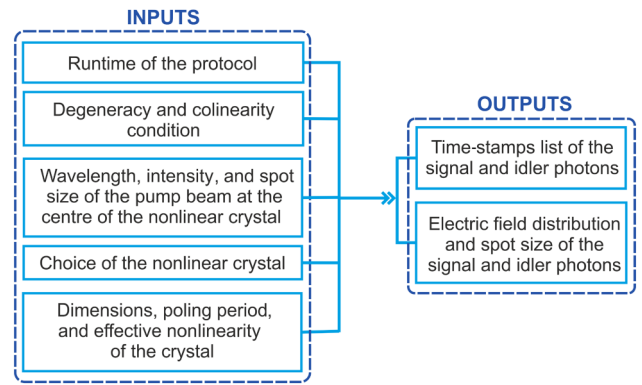


FIG. 10. Source module overview.

module and the algorithmic overview on the logic used. Each module is constructed using relevant submodules and to avoid redundancy, detailed discussions on the various submodules are given in Secs. VI and VII.

### C. Type-II SPDC source module

The type-II SPDC source module simulates the generation of photon pairs in a type-II SPDC process and the temporal distribution of the photon pair-generation events at the crystal. At present, the module simulates the specific case of type-II colinear degenerate SPDC process by quasi-phase-matching (QPM) using a PPKTP crystal. The inputs and the structure of the module are in accordance with this specific case and the submodules are also developed accordingly. Figure 10 lists the user inputs to the source module, the set parameters, and the outputs of the module. [S] denotes set parameters. With respect to the structure, the source module is constructed of submodules that simulate the integral aspects of the type-II degenerate SPDC process by quasi-phase-matching and the layout is shown in Fig. 11.

The optimal QPM condition submodule takes as input the pump wavelength and the poling period of the crystal and calculates the phase-matching temperature for the degenerate condition. The mode overlap function submodule then calculates the pair-generation probability per pump photon incident at the crystal for the given conditions of pump-beam characteristics, crystal dimensions, and temperature for the quasi-phase-matching condition. The total pair-generation rate is then calculated by taking

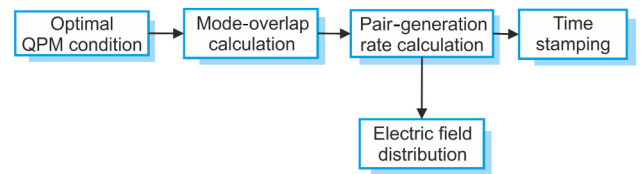


FIG. 11. Type-II SPDC source module layout.

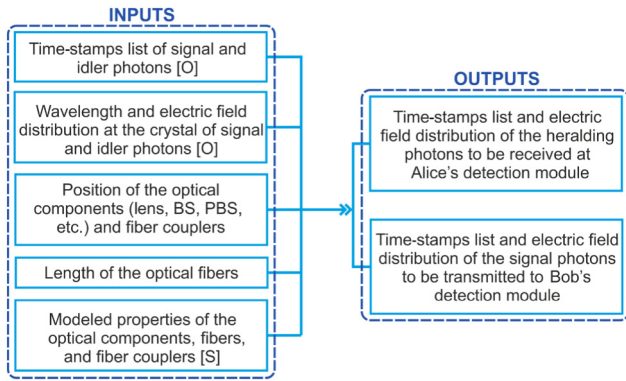


FIG. 12. Alice's preparation module overview.

into account the pump-beam intensity at the crystal and the simulated pair-generation rate obtained per pump photon from the preceding submodules.

The time-stamping submodule takes as input the pair-generation rate and the total runtime of the protocol and creates a list of time stamps of the event of photon pair generation at the crystal in respect to a global clock. The spot size of the signal and idler photons is calculated from the pump-beam spot size and the electric-field-distribution submodule generates a list of electric field amplitudes, at different points along an axis perpendicular to the direction of propagation of photons, that follows a Gaussian distribution. The time stamps for the signal and idler photons along with the Gaussian distribution of their electric field amplitudes are stored in the form of separate arrays and are returned to the main module of the toolkit.

#### D. Alice's preparation module

Alice's preparation module simulates the stage where Alice prepares and encodes the signal states that are to be sent to Bob over the quantum-communication channel. Out of the photon pairs generated at the crystal, the signal photons are encoded for transmission to Bob, whereas the idler photons are transmitted to the detection component of Alice. Figure 12 lists the inputs and outputs of Alice's preparation module. [S] denotes set parameters and [O] denotes inputs that are given as output by the previous module(s).

Alice's preparation module is constructed of various submodules corresponding to different physical processes and components that processes the time stamps and the electric field distribution of the generated photon pairs from the type-II SPDC source module. The structure of the module is shown in Fig. 13. The arrows refer to the flow of logic within the module and the two outputs correspond to the signal and the heralding arms of Alice.

The initial section of the module is common to the stream of photon pairs generated from the crystal (signal and idler). The time stamps and electric-field-distribution

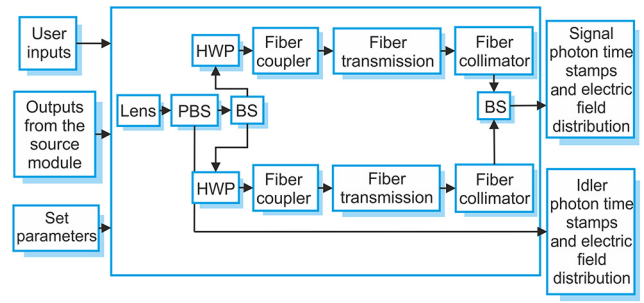


FIG. 13. Alice's preparation module layout.

arrays are first evolved through the lens submodule that provides loss and change of phase to the incident photons. The resultant electric field distribution of the photons at the signal and the heralding end of the module are calculated using the input electric-field-distribution array. The time-stamp arrays are further evolved with the other submodules. The PBS submodule generates the signal and heralding photon time-stamp arrays and while the former is passed to the following submodules, the later is not.

The signal time-stamp array is separated by the BS submodule into two with each of the arrays being further evolved through the HWP, fiber coupler, fiber transmission, and the fiber collimator submodules consecutively, similar to the experimental setup. At the HWP module, the photons are projected to specific polarization desired for transmission and encoded with a bit-value ("0" or "1") corresponding to the polarization. Each element of the time-stamp array of the photons is appended with the corresponding bit value. The resultant arrays from the two different streams are further merged at the BS submodule and the final array is generated and stored as the signal time-stamp array. The resultant arrays are returned to the main module of the toolkit.

#### E. Transmission module

The transmission module simulates the transmission of the photons sent by Alice to Bob over a free-space or fiber-based channel. Figure 14 lists the inputs and outputs of the transmission module. [S] denotes set parameter and [O]

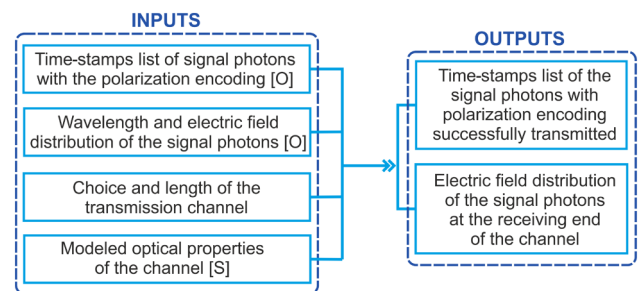


FIG. 14. Transmission module overview.

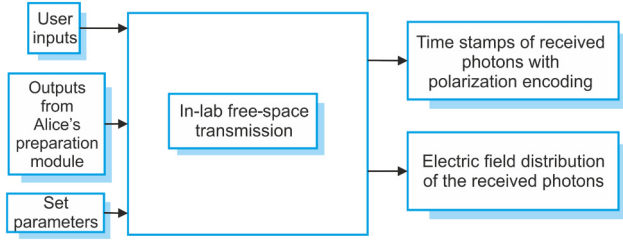


FIG. 15. Transmission module layout.

denotes inputs that are derived as output from the previous module(s). It is important to note that the choice of the channel that forms a general input to the system is specifically used in the transmission module and in accordance with the choice, the respective submodule corresponding to the free-space or optical fiber-based transmission is used. As per the experimental setup being simulated, the transmission module incorporates the in-lab free-space transmission submodule as shown in Fig. 15.

According to the experimental setup being simulated, the transmission model uses the in-lab free-space transmission submodule, which is discussed in detail in Sec. VII. The time stamps of the transmitted signal photons and the electric field distribution are evolved with the in-lab free-space transmission module as shown in Fig. 15. The transmission loss in the channel is accounted for and the transmittance of the channel is calculated, which translates to the transmission probability for the incident photons. If  $\alpha$  is the attenuation in dB per meter and  $l$  is the length of the fiber, then the transmission probability for an incident photon can be calculated as [86]

$$t = 10^{-\alpha l/10}. \quad (2)$$

For each time stamp of the input time-stamp array at the module, a random number is uniformly generated in the range (0,1) and compared with the transmission probability. If the generated random number is less than or equal to the transmission probability, then the photon is considered to be transmitted and the time stamp of the photon is added to the output array of time stamps. It must also be mentioned that the time stamps are added with the time taken for the photon to traverse the channel. If  $c$  is the velocity of light in vacuum and  $n$  is the refractive index of the channel, the time of traversal of the photons is given as

$$\Delta_t = \frac{ln}{c}. \quad (3)$$

As the output of the module, the time stamps of the photons received at Bob's detection module are returned to the main module along with polarization encoding and electric field distribution of the transmitted photons.

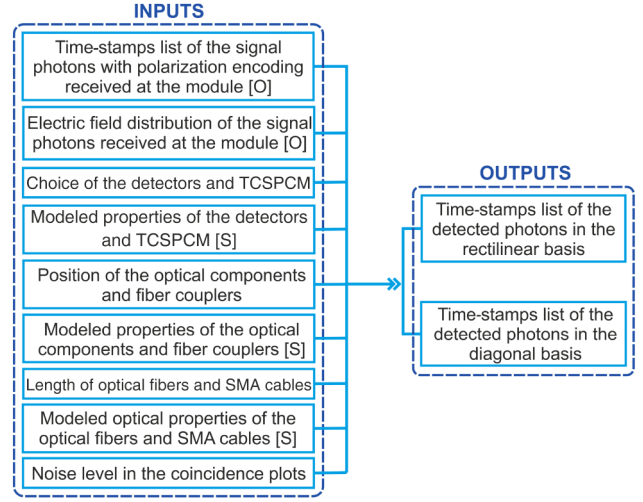


FIG. 16. Bob's detection module overview.

### F. Bob's detection module

This module simulates the detection of the signal photons transmitted by Alice over the quantum channel. Figure 16 lists the inputs and outputs of Bob's detection module. [S] denotes set parameter and [O] denotes inputs that are derived as output from the previous module(s). The choice of the detection components, i.e., the type of the single-photon detector and the time-correlated single-photon counting module dictates the use of respective submodules within this module. Bob's detection module is constructed of the submodules corresponding to the detection components chosen by the user as well as the requirements based on the choice of protocol. In parity with the experimental set up, the submodules for fiber-based single-photon detectors and TCSPCM are used. The structural layout of the module is depicted in Fig. 17. The arrows denote the flow of logic among the different submodules. The detection of single photons in the rectilinear and diagonal basis are simulated separately with the respective

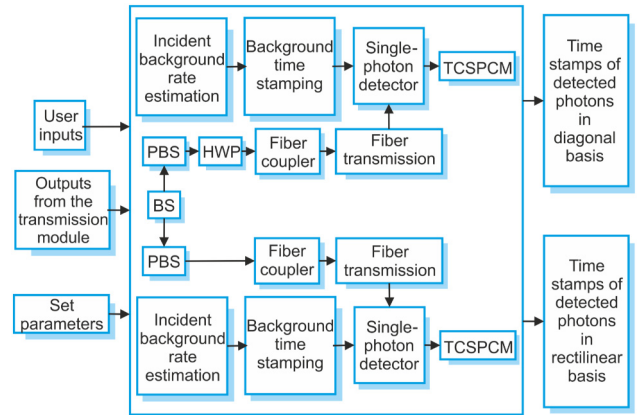


FIG. 17. Bob's detection module layout.

inputs corresponding to the noise level. The time stamps of the received photons along with the polarization encoding are evolved through the BS submodule, which splits the received time-stamp array into two separate arrays corresponding to the random basis choice for measuring the received photons. One of the time-stamp arrays is fed into the submodules, simulating rectilinear-basis measurement, whereas the other is fed into the submodules, simulating diagonal-basis measurements. For the rectilinear basis, the time-stamp array is evolved with the fiber coupler and fiber-transmission submodules to obtain the photons that are received at the detector. In parallel, the electric field distribution of the received photons are evolved according to the user input corresponding to the distance between the fiber coupler and the beam splitter. The coupling efficiency at the coupler is obtained using the fiber coupler submodule. Similar logic is followed for the diagonal basis with an addition of the HWP submodule that simulates the projection of the signal polarization states onto the diagonal basis.

From the submodules corresponding to the background detection, the time stamps of the background photons are obtained and merged with the time stamps of the signal photons received at the detector. The time stamps of the photons are then evolved with the single-photon detector and TCSPCM submodule to finally generate the time-stamp array of the detected photons. A bit value corresponding to the basis in which the photons are detected are appended to each element of the detected time-stamp time array. The resultant arrays are then returned to the main module of the simulation toolkit.

### G. Alice's detection module

Alice's detection module simulates the detection of the heralding photons that are separated from the photon pairs at Alice's preparation module. Figure 18 lists the inputs and outputs of the module. [S] denotes set parameter and [O] denotes inputs that are derived as output from the previous module(s). Similar to Bob's detection module, the module is constructed of submodules corresponding to the choice of the detection components provided by the user as depicted in Fig. 19.

The time stamps and the electric field distribution of the heralding photons at the position of the fiber coupler in the heralding arm of Alice's detection module are evolved through the relevant submodules in series. The fiber-coupler submodule uses the electric field distribution of the incident photons to simulate the coupling efficiency while the fiber-transmission submodule simulates the loss through the fiber and generates the time-stamp array of the photons received at the detector. Similar to Bob's detection module, the incident background rate is estimated and time stamps are generated with the help of the background time-stamping submodule and are merged with the time stamps

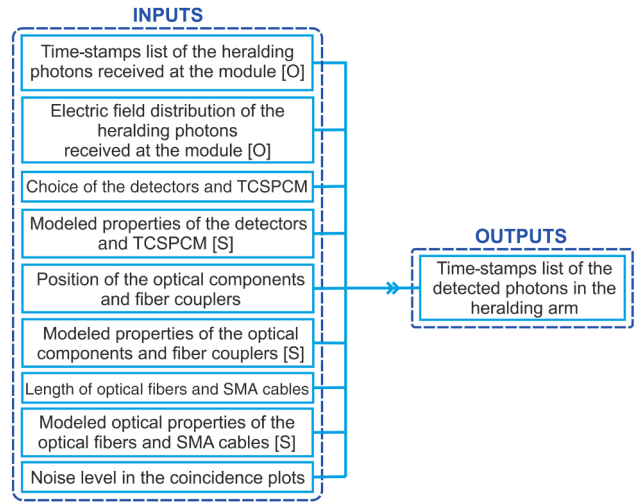


FIG. 18. Alice's detection-module overview.

of the heralded photons received at the detector. The detection of the heralding photons is then simulated with the single-photon detector and the TCSPCM submodule and the time-stamp array of the detected photons is generated and returned to the main module of the toolkit.

### H. Classical postprocessing module

The classical postprocessing module simulates the post-processing of the data after the execution of the protocol. Fig. 20 lists the inputs and outputs of the classical postprocessing module, where [O] denotes inputs that are derived as output from the previous module(s). The module incorporates the optimization strategies that are developed for implementation of the B92 QKD protocol based on single-photon sources as discussed in Secs. IV C. Depending on the choice of the user for the security parameters, the corresponding optimization algorithm submodule is used as depicted in Fig. 21.

## VI. MODELING PHYSICAL PROCESSES

In the current prototype version of the qkdSim, various physical processes are simulated as listed in Table II. The physical processes include generation of photon pairs in

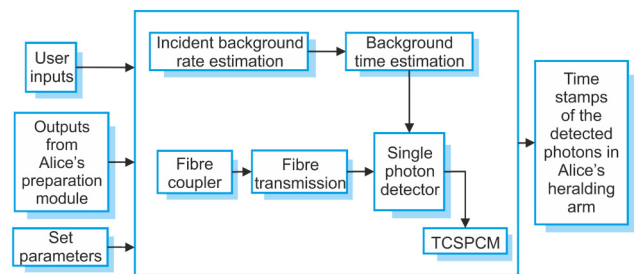


FIG. 19. Alice's detection-module layout.



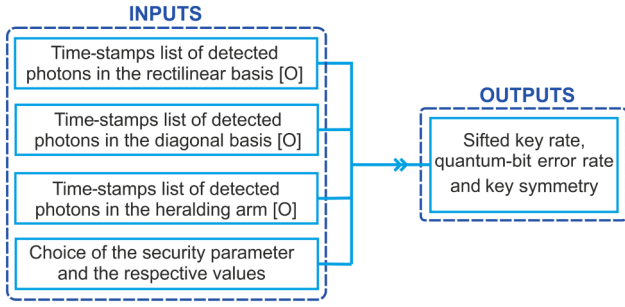


FIG. 20. Classical postprocessing module overview.

SPDC process, time stamping of generated photon pairs as well as background thermal photons, propagation of photons, and fiber coupling and collimation of the same. In this section, the simulation techniques used for these processes are discussed in detail.

### A. Spontaneous parametric down-conversion-based source

#### 1. Background

Spontaneous parametric down-conversion refers to a process of amplification of vacuum uncertainties (or fluctuations) of the optical field in the low-gain regime [87]. In a SPDC process, a photon from the pump ( $p$ ) laser beam incident on a nonlinear (type-II) crystal such as BBO, PPKTP etc. can originate two other photons: signal ( $s$ ) and idler ( $i$ ) [88]; as shown in schematic Fig. 22. Given that the index of refraction changes with frequency, only certain triplets of frequencies will be phase matched such that law of conservation of momentum [refer to Fig. 23(a)] and energy [refer to Fig. 23(b)] are satisfied. In order to achieve phase matching through the use of birefringent crystals, the highest frequency wave  $\omega_p = \omega_s + \omega_i$  is polarized in the direction that gives it the lower of the two possible refractive indices [87]. For the type-II crystal, this choice corresponds to the extraordinary polarization [87]. Also, the polarization of the pump photon should be the same (extraordinary:  $e$ ) as the signal, while the idler should have orthogonal (ordinary:  $o$ ) polarization (refer to the green encircles and arrows in Fig. 22). Thus, for type-II crystals,  $e_p = e_s + o_i$ .

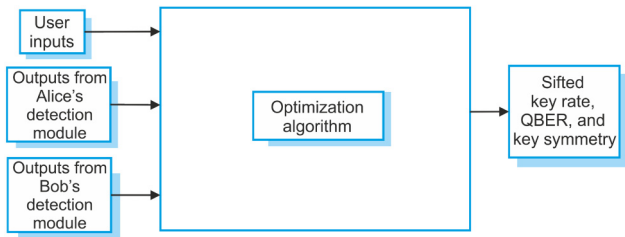


FIG. 21. Classical postprocessing module layout.

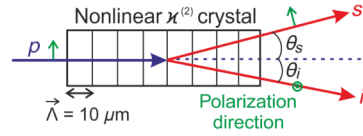


FIG. 22. Schematic of the SPDC process through a PPKTP crystal of poling period ( $\vec{\Lambda} = 10 \mu m$ ). In SPDC, the input pump ( $p$ ) photon at 405 nm (in blue) undergoes frequency down-conversion and outputs two near-infrared photons [signal ( $s$ ) and idler ( $i$ ), highlighted in red] at double its wavelength (i.e., 810 nm). The green colored dot and arrows represent orthogonal polarization directions.

From the above considerations for the conservation of energy and momentum in a SPDC process, the quasi-phase-matching condition for a periodically poled KTP crystal can be obtained by solving the Sellmeier equations using the values of the constants given in Refs. [89–91]. Numerically, the phase-matching temperature is calculated to be 44.4 °C considering a pump wavelength of 405 nm along with signal and idler wavelengths of 810 nm (refer to Appendix C for detailed expressions).

#### 2. Pair-generation probability and pair-generation rate

The pair-generation probability density (or joint spectral density) is the square modulus of the probability amplitude of the SPDC process (or joint spectral amplitude), i.e.,  $|\psi(\omega_s, \omega_i)|^2$ . Now since  $\omega_p = \omega_s + \omega_i$  and  $\omega_p$  is the coherent state of the laser source, if we numerically integrate  $\psi(\omega_s, \omega_i)$  over a possible range of signal frequencies ( $\omega_s$ ) then we can obtain a  $\text{sinc}^2$  nature plot for this pair-generation probability density function plotted over a spectrum of signal mode frequencies as illustrated through a schematic in Fig. 24.

The maximum of this probability distribution provides the corresponding wavelength information at which the SPDC pair-generation rate is maximal. The maximum value of pair-generation probability obtained numerically

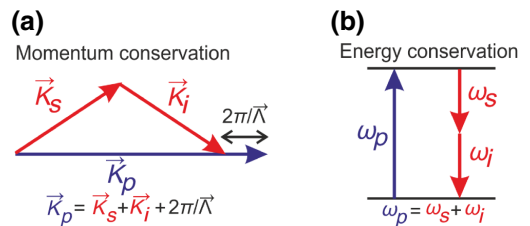


FIG. 23. Schematic describing the relations for the law of conservation of momentum (left) and energy (right), where  $\vec{\Lambda}$  is the poling period.

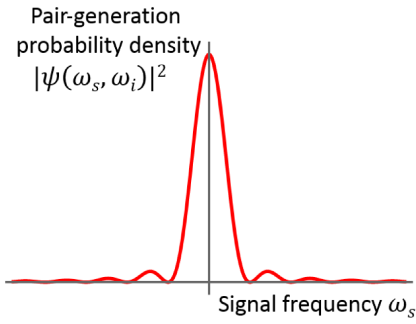


FIG. 24. Schematic describing the  $\text{sinc}^2$  nature of pair-generation probability density function corresponding to a spectrum of mode frequencies.

can be directly verify with the experimental data. Consequently, the pair-generation rate is given by

$$R_T \propto \langle \psi | \psi \rangle = \int |\psi(\omega_s, \omega_i)|^2 d\omega_s d\omega_i. \quad (4)$$

For detailed analytical derivation of the pair-generation rate, refer to Appendix D and for the numerical approach implemented the simulate the same, refer to Appendix E.

### B. Time stamping of single and background photons

In an idealistic picture, we consider a photon source that generates perfect (only) single-photon events (Fock state  $|1\rangle$ ). The quantum uncertainty of detecting each output photon with frequency  $\omega$  at time  $t$  is  $\Delta t \Delta \omega \geq \frac{1}{2}$ . The probability distribution of measuring a given photon at time  $t$  (or with frequency  $\omega$ ) then becomes the modulus square of its wave function (or its probability amplitude). The distribution also depends on the source properties as well as the filtering conditions. For example, in a SPDC source (as presented through a schematic in Fig. 25), the probability distribution can be Gaussian or Sinc-squared, depending on the nonlinearity profile of the crystal as well as the spectral profile of the filter.

Defining the probability of a photon generated at time  $t_1 + \tau$  is  $\Pr(t_1 + \tau | t_1)$  given that the earlier photon is generated at time  $t_1$ ; then for an ideal single-photon source, for  $\tau = 0$ , this probability becomes zero, i.e.,  $\Pr(t_1 | t_1) = 0$ . Now for all  $\tau \gg t_{\text{coh}}$ , where  $t_{\text{coh}}$  is the coherence time of the single photon, this probability will have a constant value  $p$ , i.e.,  $\Pr(t_1 + \tau | t_1) = p$ . This is because of the fact that for very large values of  $\tau$  beyond the coherence time  $t_{\text{coh}}$ , the source behaves truly randomly and emits single photons at any arbitrary interval with equal probability. Therefore as shown in Fig. 26(b), if we plot this probability as a function of  $\tau$ , it will smoothly increase from 0 and saturate at  $p$ .

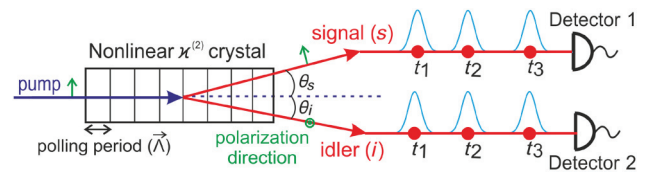


FIG. 25. Probabilistic ideal single-photon source emitting a stream of photon pairs at times  $t_1, t_2, t_3$ . Detection of the idler photon heralds the signal photon of the same pair. The detection-time uncertainty for each photon is depicted with a Gaussian distribution.

Let us assume that  $10^7$  photons are generated per second. So, we divide the time span of 1 s into equal bins of size of 1 ps. This provides us with  $10^{12}$  bins in 1 s. When  $10^7$  photons are randomly distributed into those  $10^{12}$  bins, the probability of having a photon in each bin becomes  $10^{-5}$ . However, we enforce an idealistic restriction that if one photon has already been assigned to a bin, say  $x$ , then the probability of another photon to be assigned to the same bin ( $x$ ) is zero. Now as illustrated earlier in Fig. 26, this probability of assignment for each bin slowly increases for the subsequent bins, i.e.,  $x + 1, x + 2, x + 3, \dots$  and saturates at  $p = 10^{-5}$  for some bin number, say in this case  $x + n$ , where  $n$  is related to the coherence time. However, it is important to note that for a SPDC-based (nonideal) single-photon source the probability of assigning more than one photon is negligible [ $\Pr(t_1 | t_1) \rightarrow 0$ ]. The algorithm for time stamping and the comparison between the simulated and the experimental time-stamping data can be found in Appendix F.

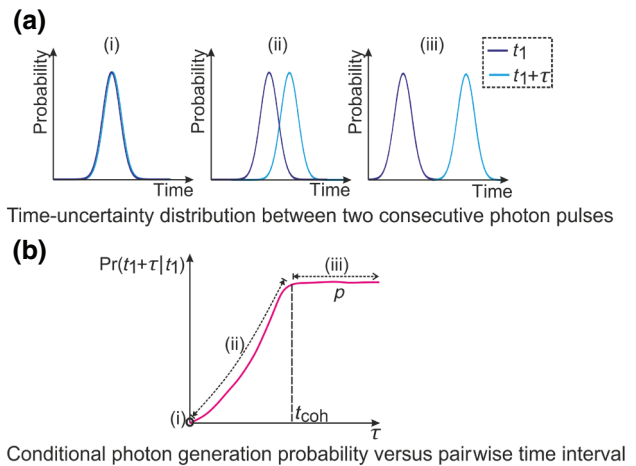


FIG. 26. (a) Schematics comparing the positions of time-uncertainty distributions for two single-photon events with three different relative time intervals  $\tau$ : (i)  $\tau = 0$  where  $\Pr(t_1 + \tau | t_1) = 0$  (ii)  $0 < \tau < t_{\text{coh}}$  where  $0 < \Pr(t_1 + \tau | t_1) < p$  (iii)  $\tau \gg t_{\text{coh}}$  where  $\Pr(t_1 + \tau | t_1) = p$ . (b) Schematic of the nature of the probability distribution  $\Pr(t_1 + \tau | t_1)$  as a function of  $\tau$  for any arbitrary  $t_1$ .

While generating the time-stamp list for each run of the protocol, it is observed that the process increases the runtime of the simulation significantly. In order to speed up the process, we adapt the data resampling technique to generate the time-stamp list. For an arbitrary simulated single-photon pair-generation rate, the time-stamp list is generated only for the first instance of the simulation. In further runs of the simulation, where the parameters for the type-II SPDC source module remain unchanged and hence a similar pair rate is obtained, the resampling technique is used to generate the time-stamp list. The application of the resampling technique results in variation in the number of time stamps generated for a fixed time interval. The number of time stamps generated determines the effective number of photon pairs that form the input to the subsequent modules in the simulation toolkit. In Secs. V B, we mention that it is assumed that all the parameters remain invariant over the runtime of the simulation. Thus, for multiple runs of the simulation with fixed parameters, the simulated pair-generation rate remains constant over all such instances but the effective number of photon pairs generated varies because of the usage of resampling technique. This effectively mimics the variation in the single-photon pair-generation rate in the experimental setup. Refer to Appendix H for a detailed discussion and algorithm for the time-stamp resampling method.

### C. Detection of background photons

Besides the single photons from a SPDC source, the experimental detections also consists of other background photons that commonly originate from any surrounding thermal source. From the literature, the photon-number ( $n$ ) distribution for such a source is commonly *super-Poissonian*, where  $\text{Pr}(n) = \mu^n / (1 + \mu)^{n+1}$ . If we measure the second-order coherence  $g^{(2)}$  for such a source, then it exhibits bunching property, where  $g^{(2)}$  at  $\tau = 0$  is greater than that at  $\tau \gg 0$ . In our simplified model to simulate these background photon statistics we consider multiphoton events only up to two photons. Also, we assume that  $\text{Pr}(n=2) = \text{Pr}(n=1)^2$ , which implies that  $\text{Pr}(2)/\text{Pr}(1) \ll 1$  since  $0 \leq \text{Pr}(n) \leq 1$ . Here, it is important to note that for an ideal single-photon distribution  $\text{Pr}(2)/\text{Pr}(1) \approx 0$ .

Let us assume that  $10^5$  background photons are incident at the single-photon detector per second. As in the case of single-photon time stamping, we divide the time span of 1 s into bins of equal time intervals, each of 1-ps time resolution. Thus we get  $10^{12}$  bins in 1 s. Now, let us consider, the probability of assigning a single thermal photon-generation event to an empty bin is  $P_1$ , then the probability of assigning a multiphoton event to an empty bin becomes  $P_1^2$  from the idea discussed above. Therefore, the probability of assigning at least one photon in each bin is  $P_1 + 2 \times P_1^2$ . Thus, from the consideration that

$10^5$  background photons are incident per second, the value of  $P_1$  becomes approximately  $10^{-7}$ . We use the considerations made above to generate the time-stamping data for our background contributions from thermal sources. Refer to Appendix G for the algorithm implemented for generating background time stamps.

But before we perform the time stamping of the incident background photons, we need to simulate the background incidence rate. This estimation on the incidence rate is done by taking into account the noise level in the coincidence plots at the detectors for the different basis measurements performed in the protocol. This noise level in the coincidence plots is similar to what is mentioned in Fig. 7 and is taken as a user input to the detection modules. Initially, a dataset is generated by varying the incident background rate at the detectors and obtaining the corresponding background coincidence rate for a fixed signal rate. The dataset is then stored and interpolated to obtain the incident background rate for a given input background coincidence rate at the fixed signal rate.

Now we need to extend the aforementioned logic to calculate the incident background rate for any incident signal rate, i.e., we want to estimate the ratio in which the interpolated background rate will be affected. This is done by the method explained as follows. A dataset is generated by varying the signal rate and obtaining the corresponding background coincidence rate for a fixed background incidence rate at the detectors. Now the ratio in which the incidence background rate changes with the signal rate can be calculated by dividing the background coincidence rate obtained at any arbitrary signal rate with the background coincidence rate at the fixed signal rate for which a dataset has already been generated. Thus the two datasets can be interpolated to obtain the incident background rate for any arbitrary signal rate and background coincidence level. Refer to Appendix G for the algorithm implemented for obtaining the incident background rate at the detector.

### D. Propagation of single photons

In the source module, each of the generated photons are associated with a Gaussian distribution corresponding to their electric field amplitude. The propagation of single photons are thus simulated by using Huygen's principle for propagation of Gaussian beams in two dimensions. Refer to Appendix J for the algorithms implemented for the generation of the electric-field-distribution array as well as the propagation of single photons.

### E. Fiber coupling

In the experimental setup, coupling of single photons into a single-mode fiber is done using mounting stages and fiber couplers that provide certain degrees of freedom to

align the fiber tip with the input beam in order to maximize the coupling. In our simulation toolkit, the fiber-coupling submodule is used to simulate an active coupling mechanism that a user might perform while aligning the experimental setup. As mentioned in Secs. VB, the position of the fiber tip is simulated accurately for the maximum coupling limited only by the errors generated from the least count of the coupling apparatus. The error in the position of the fiber tip is simulated by randomly selecting a value from a Gaussian distribution with the least count being twice the sigma of the distribution. The submodule takes in as input the specifications of the physical components such as the fibers and the aspheric lenses within the couplers and the electric field distribution of the incident photons and returns the effective coupling efficiency as the output. The efficiency is calculated as the overlap between the intensity distribution of the received photons at the fiber tip and the Gaussian distribution considering the mode-field diameter of the fiber. By further considering the associated losses, the effective coupling efficiency is obtained. Refer to Appendix K for the detailed methodology and algorithms implemented to simulate the fiber-coupling process.

## VII. MODELING PHYSICAL COMPONENTS

In this section, we discuss briefly the various physical components that are simulated, as listed in Table I, for the current version of qkdSim. The physical components that form the experimental setup for demonstration of the B92 protocol includes both optical and electrical components such as lenses, filters and beam splitters, free-space and optical fiber-based channel, detectors, and TCSPCM. The simulation methodology for the submodules corresponding to these physical components are discussed as follows.

### A. Lens

The working of a lens is simulated by using the lens-transfer function calculated from the specifications of the lenses used in the setup and applying that on the electric field corresponding to the incident photons. As specified in Secs. VB, the photons are considered to have a Gaussian electric field distribution, so the effect of the lens is simulated taking the case of Gaussian beams only. The lens submodule takes in as input the specifications corresponding to the type of material and its dimensions, etc., the electric field distribution and the time-stamp array of the incident photons. A part of the total number of incident photons gets lost during transmission because of the loss in the medium. The submodule returns as output the electric field distribution of the photons after they pass through the lens and the corresponding time stamps.

### B. Half-wave plate (HWP)

The HWP submodule simulates the the effect of phase shift on the polarization of the photons transmitted through the HWPs in the experimental setup. The orientation of the fast axis of the HWP with respect to the polarization of the incoming photons is simulated by the choice of basis for the required projection. The accuracy of the orientation is limited only by the least count of the mounting component and the error is simulated by randomly choosing a value from a Gaussian distribution with the least count being twice the standard deviation of the distribution. The parameters regarding the loss through the medium, least count of the mounting component etc. are set within the submodule. For each element of the time-stamp array evolved with the submodule, the rotation of polarization is calculated by taking into account the simulated orientation of the fast axis of the HWP, and then the polarization angle is appended to the corresponding element of the array.

### C. Filter

The filter submodule simulates the effect of filter on a beam or photons of certain wavelength. For now we model a very simplified filter inspired from the real components used in the experimental setup and thus can be further enhanced to capture more practical scenarios. The submodule takes into account the insertion and the transmission losses incurred by the photons incident on the filter. From these losses, the transmission probability of an incident photon is calculated. When the time-stamp array is evolved with the submodule, the elements are randomly selected based on the transmission probability and a new array for time stamps corresponding to the transmitted photons is created and returned as output.

### D. Polarizing beam splitter (PBS)

The PBS submodule simulates the transmission of a linearly polarized beam through a PBS. The extinction ratio and the transmission loss associated with the PBS is set within the submodule as set parameters by taking data from the specifications' sheet provided by the manufacturer. The submodule takes as input the time stamps of the incident photons and the polarization angle of the photons. Depending on the extinction ratio, each element of the input time-stamp array is randomly selected for either transmission or reflection. By further considering the loss through the medium, separate time-stamp arrays are created corresponding to the transmission and reflection arms of the PBS and returned as outputs.

### E. Beam splitter

The BS submodule simulates the transmission of a beam through a beam splitter. The transmission loss and the splitting ratio associated with the BS is set within the

submodule as set parameters by taking data from the specifications' sheet provided by the manufacturer. The submodule takes as input the time stamps of the incident photons and similar to the PBS submodule, depending on the splitting ratio and loss through the medium, separate time-stamp arrays are created corresponding to the transmission and reflection arms of the BS and returned as outputs.

### F. Fiber transmission

The fiber-transmission submodule simulates the transmission of photons through an optical fiber of a certain length. The submodule takes into account only the losses encountered by the beam during transmission and does not consider any other effect due to the fiber. This takes as input the length of the fiber channel and the time-stamp array of the coupled photons into the fiber. The loss associated with the transmission as well as the insertion loss at the mating sleeves are taken from the specification sheet of the fiber and defined within the submodule. The total loss calculated determines the transmission probability of the photons through the fiber. If  $\beta$  is the loss in dB due to the connectors of the fiber and  $\alpha$  is the transmission loss in dB per meter, then the total loss ( $\gamma$ ) can be calculated as

$$\gamma = \beta + \alpha l, \quad (5)$$

where  $l$  is the length of the fiber. The transmission probability can then be calculated as [86]

$$t = 10^{-\gamma/10}. \quad (6)$$

The refractive index of the material of the fiber is also accounted for from the specification sheet of the fiber and is used to calculate the traversal time of photon using Eq. (3). Each element of the input time-stamp array is randomly selected, depending on the transmission probability, to form the time-stamp array corresponding to the transmitted photons. This array is returned as output by the submodule.

### G. In-lab free-space transmission

The in-lab free-space transmission submodule simulates the transmission of photons through free space of a certain length in laboratory conditions where temperature, humidity, and lighting are controlled. It takes into account only the losses encountered by the beam during transmission and does not consider any other effect due to the free-space channel. The submodule takes in as input the length of the free-space channel and the time-stamp array of the photons traveling through the channel. For the in-lab free-space transmission channel, the attenuation is taken as 0.2 dB per km and the transmission probability is calculated using Eq. (2). The refractive index of the

medium is taken as unity and the time of traversal for the photons is obtained using Eq. (3). Similar to the fiber-transmission submodule, depending on the transmission probability, elements of the input time-stamp array are randomly selected to form the time-stamp array corresponding to the transmitted photons.

### H. Fiber-based single-photon detector

The fiber-based single-photon detector submodule simulates the detection process of single photons with fiber-based detectors. In other words, we consider the case where single photons are coupled to the detector with a fiber and get detected via an avalanche breakdown process at the photo diode of the detector. For each detection event, the detector outputs a transistor-transistor logic (TTL) pulse corresponding to the detected photon. The submodule provides a simplistic approach towards modeling of single-photon avalanche detectors and the scope is restricted to only SPADs. It takes the time-stamp array of the stream of photons incident at the detector as an input. The primary detector imperfections that we consider for simulation are the quantum efficiency  $r$ , dead time, and timing jitter of the detector. The values corresponding to these parameters are taken from the specification sheet of the detector and set within the submodule. The imperfections are discussed as follows.

1. *Quantum efficiency*: A single-photon detector has a certain efficiency of detecting photons incident on it, i.e., for each of the photons received at the detector, a TTL pulse is not generated. The probability of generation of the TTL pulse on receiving an incident photon is quantified using the parameter quantum efficiency of the detector. The quantum efficiency of the detector or the detector efficiency depends on the wavelength of the incident photons.

2. *Detector dead time*: The dead time of a detector is the time interval after a detection event, followed by an avalanche breakdown, during which the detector is unresponsive to any photon incident at the detector. It defines the time required by the detector to restore the quenching circuit. Thus the minimum time interval possible between two detection events is the dead time of the detector.

3. *Timing jitter*: Due to imperfections in the detector circuit, there is a time uncertainty between receiving a photon at the detector and generating the TTL pulse. The time interval between generation of the TTL pulse corresponding to a photon detection and the time at which the photon is actually received at the detector is not constant and has a Gaussian distribution. This uncertainty is quantified by the timing jitter of the detector, i.e., the FWHM of the distribution.

The logic used for simulating the single-photon detection event at the detectors considering the specified imperfections can be explained as follows. The quantum efficiency affects the difference in the total number of photons incident at the detector and those which actually get detected. For each photon incident at the detector, a random number is generated in the range  $[0, 1]$  as in principle the efficiency of the detector is  $\leq 1$ . If the random variable has a value less than the quantum efficiency of the detector, the instance is considered as the receiving event. The detector dead time is then accounted for by checking the difference in the time stamp of the consecutive photons received at the detector. If the difference is less than the dead time, the photon corresponding to the larger time-stamp value is discarded and the time difference with the next photon is checked and the process continues for all the received photons. To simulate the timing jitter of the detector, each of the generated TTL pulses corresponding to detected photons is adjusted with a time delay chosen randomly from a Gaussian distribution with a mean ( $\mu$ ) of zero and standard deviation ( $\sigma$ ) value equalling 2.3 times the timing jitter of the detector. The time-stamp array corresponding to the detected photons is then returned as output.

### I. Time-correlated single-photon counting module

A TCSPCM receives the TTL pulses generated at the detector corresponding to the detection events and registers time stamps for the TTL pulses. The TCSPCM submodule simulates this process by taking as input the time stamps of the TTL pulses generated at the detector corresponding to the detection events. The primary TCSPCM imperfections that we consider for simulation are the losses in the Sub-Miniature version A (SMA) cables that connects it to the detectors, dead time, and timing jitter of the TCSPCM. In the following, we explain the various parameters that are accounted for to model the TCSPCM.

1. *SMA cable losses*: SMA cables are used to connect the detector to the TCSPCM. These cables have some inherent losses at the connectors and as a result of that,

some of the TTL pulses generated from the detectors are lost.

2. *TCSPCM dead time*: Similar to the detectors, the dead time of a TCSPCM is the minimum time interval between registering two TTL pulses received from the detector. The value of the dead time is set within the submodule as per the specification sheet.

3. *Timing jitter*: Due to imperfections in the TCSPCM circuit, the time stamps associated to a received pulse is not the same as the time at which the TTL pulse is received. The distribution of this time delay is a Gaussian distribution, centered at the time at which the pulse is actually received by the TCSPCM. This deviation is quantified using the timing jitter of the TCSPCM, which forms the FWHM of the Gaussian distribution.

The dead time of the TCSPC module and the timing jitter is simulated in the same way as done for the detector. The channel efficiency is calculated using the specified losses of the SMA cable and is simulated in a similar manner as the detection efficiency. Refer to Appendix I for the algorithms implemented for the simulation of single-photon detection with the methodology discussed for fiber-based single-photon detector and TCSPCM sub-modules

## VIII. RESULTS AND DISCUSSION

Our paper introduces a simulation toolkit called qkdSim. While in the future, we aim to develop this into a software that will be able to simulate any QKD protocol along with consideration of the associated experimental imperfections, in the present work, we show details of the B92-protocol simulation and its performance analysis in comparison to the results obtained from our experimental demonstration of the B92 protocol using a heralded single-photon source. We find a reasonably good match between our simulation and experiment as discussed below.

The comparative results from the experiment and simulations, using the two optimization strategies A and B, are presented in Tables IV and V, respectively. The results

TABLE IV. Optimized results of average key rate, QBER and asymmetry (i.e., key symmetry), obtained using strategy A, from the experiment and the simulation. Note that an asymmetry value of “x” implies that the ratio of “0” bits to “1” bits in the key is x:(100-x).

Crystal length (mm)		Optimization strategy A						
		Time of the day	From experiment			From simulation		
			key rate (kHz)	QBER (%)	asymmetry	key rate (kHz)	QBER (%)	asymmetry
20	Day	$47.6 \pm 0.6$	$4.79 \pm 0.01$	$49.82 \pm 0.01$	$53.1 \pm 0.3$	$4.79 \pm 0.01$	$50.1 \pm 0.06$	
	Night	$51.0 \pm 0.5$	$4.79 \pm 0.01$	$50.15 \pm 0.02$	$52.8 \pm 0.4$	$4.79 \pm 0.01$	$50.1 \pm 0.05$	
30	Day	$33 \pm 2$	$4.78 \pm 0.01$	$50.07 \pm 0.02$	$64 \pm 1$	$4.78 \pm 0.01$	$50.05 \pm 0.08$	
	Night	$36 \pm 3$	$4.78 \pm 0.01$	$50.08 \pm 0.02$	$60 \pm 2$	$4.79 \pm 0.01$	$50.01 \pm 0.11$	

TABLE V. Optimized results of average key rate, QBER, and asymmetry (i.e., key symmetry), obtained using strategy B, from the experiment and the simulation. Note that an asymmetry value of “x” implies that the ratio of “0” bits to “1” bits in the key is x:(100-x).

Crystal length (mm)	Time of the day	Optimization strategy B					
		From experiment			From simulation		
		key rate (kHz)	QBER (%)	asymmetry	key rate (kHz)	QBER (%)	asymmetry
20	Day	47.8 ± 0.6	4.79 ± 0.01	50.2 ± 0.3	60.0 ± 0.2	4.79 ± 0.01	57.0 ± 0.2
	Night	53.8 ± 0.4	4.79 ± 0.01	53.7 ± 0.3	59.8 ± 0.2	4.79 ± 0.01	57.0 ± 0.2
30	Day	36 ± 2	4.79 ± 0.01	54.0 ± 0.3	71 ± 1	4.79 ± 0.01	57.1 ± 0.3
	Night	38 ± 3	4.78 ± 0.01	54.1 ± 0.4	66 ± 2	4.79 ± 0.01	57.1 ± 0.3

reported for the experiment are obtained by averaging the measurement outcomes over 20 measurement sets, wherein each set involved a measurement time of 10 s. The choice for this measurement time is motivated from the result of the bootstrapping analysis discussed at the end of Secs. IV C and described in Appendix A. For the simulation, while the number of datasets considered for averaging remained the same, the runtime of the protocol is considered to be 1 s. Here, the consideration of a smaller runtime for the simulation can be motivated from the assumption of time invariance that is considered for the system. The error values quoted correspond to the standard deviations in the results obtained from 20 iterative runs of the protocol, for both simulation and experiment.

In Figs. 27 and 28, we exemplify the distribution of the key rate, QBER, and key symmetry values corresponding to a range of coincidence window sizes, considering a representative daytime dataset (chosen from those 20 measurement sets), for both crystals of length 20 and 30 mm, respectively. More particularly, in both figures, we vary the coincidence window size (i.e., the gap between  $W_{11}$  and  $W_{r1}$  or  $W_{12}$  and  $W_{r2}$ ) in steps of 10 ps from the global maximum point (shown in Fig. 7) and then use two unoptimized strategies: A and B, that are slight variations to the optimization strategies presented in Appendix B. In the unoptimized strategy A, we calculate the key rate (plotted with blue “+” symbol) for the (unrestricted) QBER (plotted with red “+” symbol) corresponding to each size of the coincidence window, while ensuring that the key symmetry remains 50:50 (approximately). This is done by slightly readjusting the coincidence window size on the coincidence peak 2 (i.e., the positions of windows:  $W_{12}$  and  $W_{r2}$ ). Thus, it can be observed in Figs. 27(b) and 28(b) that the distribution of key symmetry (marked with green “+” symbol) remains 50 over the whole range of coincidence window sizes. In the unoptimized strategy B, we only slightly deviate from its optimized strategic version, by calculating the key rate (plotted with blue dots) for the (unrestricted) QBER (plotted with red dots) corresponding to each size of the coincidence window; however, in this case apart from removing the threshold on QBER values, we also relax the

50:50 constraint on the key symmetry. For this strategy, the corresponding variations in key symmetry values are shown in Figs. 27(b) and 28(b) with pink dots. We can

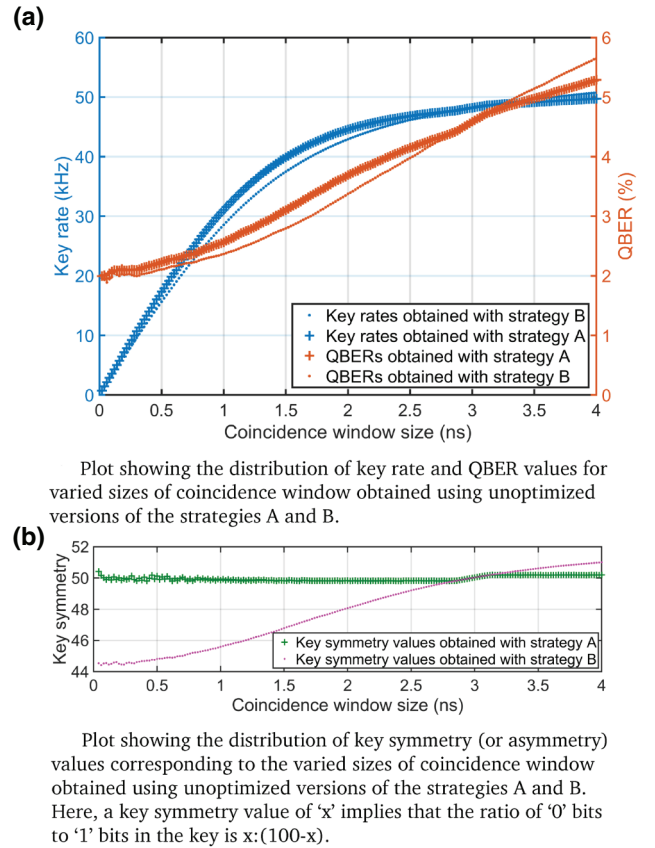


FIG. 27. Distribution of key rates, QBERs, and key symmetries plotted against a range of coincidence window sizes for the crystal of length 20 mm. (a) Plot showing the distribution of key rate and QBER values for varied sizes of coincidence window obtained using unoptimized versions of strategies A and B. (b) Plot showing the distribution of key symmetry (or asymmetry) values corresponding to the varied sizes of coincidence window obtained using unoptimized versions of strategies A and B. Here, a key symmetry value of “x” implies that the ratio of “0” bits to “1” bits in the key is x:(100-x).

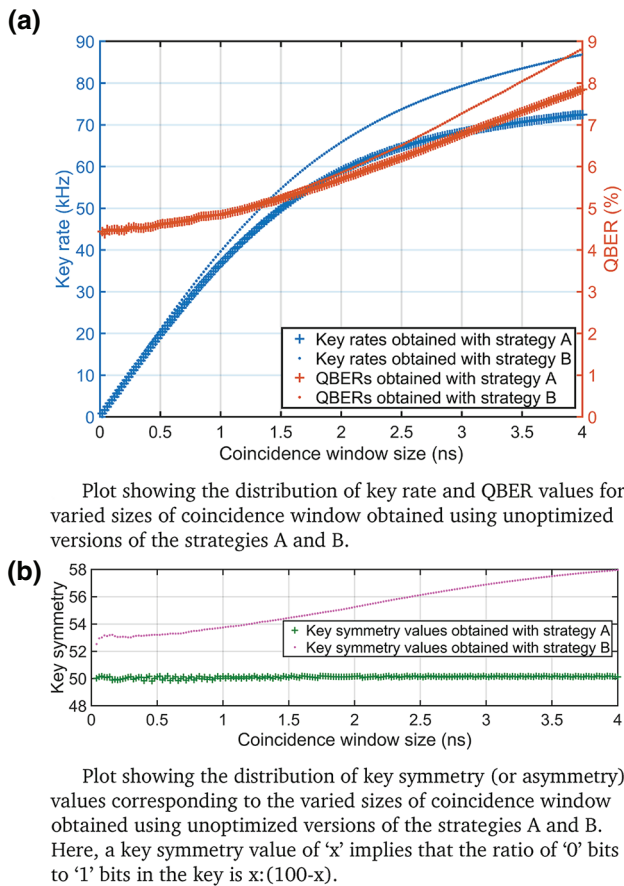


FIG. 28. Distribution of key rates, QBERs, and key symmetries plotted against a range of coincidence window sizes for the crystal of length 30 mm. (a) Plot showing the distribution of key rate and QBER values for varied sizes of coincidence window obtained using unoptimized versions of strategies A and B. (b) Plot showing the distribution of key symmetry (or asymmetry) values corresponding to the varied sizes of coincidence window obtained using unoptimized versions of strategies A and B. Here, a key symmetry value of “x” implies that the ratio of “0” bits to “1” bits in the key is  $x:(100-x)$ .

observe that in strategy B, for the two coincidence peaks (1 and 2) being asymmetric (in height and width) and also slightly skewed due to realistic imperfections, the key symmetry keeps constantly decreasing in a given direction over the range of coincidence window sizes. Intuitively, in both Figs. 27(a) and 28(a), the key rate and QBER values are directly proportional to the size of the coincidence window, as they result from the area under the curve estimates depicted in Fig. 7. In Fig. 28(a), the key rate and QBER values grow more steeply compared to that in Fig. 27(a), since both pair production rate and background noise is higher for the longer crystal of 30 mm. This is because we do not focus our Gaussian pump beam very tightly to ensure that we achieve a good heralding ratio [92] (refer to Appendix D for details). Also, we note that towards the

larger sizes of the coincidence window, the growth of the key rate reduces and becomes sublinear, compared to the linearity of the slope for the QBER distribution, since at those points the coincidence window span has expanded and entered the flat zone where the background noise is increasing more rapidly than the SNR. Most importantly, we observe that beyond a certain choice of coincidence window size the information theoretic QBER threshold of 4.8% [67] for the B92 protocol gets violated for both crystals, reaching upto approximately 88 kHz key rate with approximately 8.9% QBER in Fig. 28(a) and upto approximately 50 kHz key rate with approximately 5.7% QBER in Fig. 27(a). Lastly, we observe that the key-rate values corresponding to 4.8% QBER in Fig. 28(a) for the shorter crystal and Fig. 28(b) for the longer crystal is around 48 and 34 kHz, respectively; along with the fact that the key-rate curve for strategy B shoots above that for strategy A in Fig. 28(b), unlike that in Fig. 28(a) where they actually overlap. These observations help us to infer that the optimization results presented in Tables IV and V are well in agreement with these unoptimized results. These results also bring out the importance, utility, and performance analysis of the optimization methods that have been introduced in this paper.

One of the key features of the results reported in Tables IV and V is that irrespective of the optimization strategy used, the simulated key rates are seen to be higher than the experimentally measured ones. Whereas, in the qkdSim framework, we indeed taken into account realistic imperfections associated with several optical components and processes, thus enabling a close match between simulation and experiment, there are more effects that we will be incorporating in future versions of the toolkit, which will enable an even closer match. One of the key factors, which plays a major role in the reported key rate is the fiber coupling of the photons. The efficiency of coupling is affected by numerous factors such as the coupling lens arrangement, the degrees of freedom provided by the lens mount, the least count of the mounts, direction of the incoming beam as well as human error. In the qkdSim module, which simulates coupling, we take into account the lens specifications, the fibers as well as the mounts. However, the incoming beam is considered to be perfectly perpendicular to the spherical front surface of the lens. Any deviation from this decreases the coupling efficiency. We also assume that the beams are perfectly Gaussian in the spatial domain whereas in experiment, the beam may deviate from this assumption. These factors create a difference in experimental and simulated coupling efficiency, which inevitably means that the simulated efficiency is always higher. As a result, the simulated key rate is higher than the experimentally measured one.

All the reported results in Tables IV and V are obtained using the optimization strategies (introduced in Secs. IV C) and as a consequence of it, the results are dependent on the



background noise in the coincidence plots. As discussed, the source of this noise is the background photons incident at the detector. We observe in Table IV that the error values for both estimated QBER and asymmetry parameters, are at least one order of magnitude lower than those for the key rates, irrespective of their origin: i.e., from experiment or simulation. The reasons for this are the constraints on QBER (approximately 4.8%) and key symmetry (approximately 50), which are fixed in case of the optimization strategy A to obtain the key rate, as discussed earlier in Secs. IV C. Therefore, all the fluctuations in the measured data get reflected on the key-rate results. However, in Table V, we observe that both key rate and key symmetry (or asymmetry in the key) have one order of magnitude higher-error values than QBER. This is in accordance with the fact that in optimization strategy B, the key symmetry, along with key rate, is also considered as an unconstrained parameter for optimization. It is important to note that such choices of optimization, results in different implications on the security of the resultant secure key as discussed earlier in Secs. IV C.

Moreover, for both tables, our in-lab free-space experimental demonstration of the B92 protocol (discussed in Secs. IV B) is conducted during day and night time. For both time periods, the similarity of the experimental results reflects that the in-lab conditions varied indistinctly at different times of the day. In principle, for in-lab conditions, the estimated key rate should be the same for both day and night time measurement. However, we find that irrespective of optimization strategy, the measured key rate in the night data is slightly higher than that measured during the day time. We compare the raw unoptimized data to investigate this observation and find that the night-time key rates are higher than the day time ones. For the 20-mm crystal and considering a 3-ns time window, we obtain a key rate of 47.47 and 51.1 kHz for the experiment during the day and night time, respectively. Thus, this observation is not an artefact of a given optimization strategy nor is it simply correlated with higher or lower measured background. We believe that the source of this slight increase in night-time key rate may be related to the overall temperature conditions. As our source is a PPKTP source, the phase matching is governed by the temperature of the crystal. As explained in Secs. IV B, the crystal temperature is maintained at the phase-matching temperature through a feedback mechanism involving an oven and a temperature controller. Ideally, the oven temperature is the same as the one displayed on the temperature controller. However, during the day time, the outside temperature fluctuates quite a lot, which results in slight lab temperature fluctuations at a fractional level. The temperature that is experienced by the crystal is a homogenized combination of the oven temperature as well as the overall room temperature. From the analytical relationship between the quasi-phase-matching temperature and SPDC

pair rate in a type-II PPKTP crystal, as highlighted over Appendices C and D, along with its numerical calculation discussed in Appendix E, it can be observed that even a 0.1 degree change in crystal temperature may lead to change in phase-matching condition and hence, pair-generation rate. Thus, the small difference in homogenized temperature may cause slightly higher night-time key rates than day time ones. However, as the background noise level forms a key input to the simulation toolkit, the negligible variation of the in-lab conditions at different times of the day gets reflected in the simulation results, in which the night- and day-time key rates are almost the same, within respective error bars.

Another observation from both tables is that although for our case of not tightly focused Gaussian pump beam, having a longer crystal length should potentially lead to a higher key rate [as depicted in Fig. 28(a)] due to an increased pair production rate [92], the crystal of length 30 mm in fact produces a lower key-rate estimate, for the given threshold QBER value, than the 20-mm one for the experimental results unlike the simulation part, where the logic is rather consistent. This is because in the experimentally measured datasets, the 30-mm crystal besides having higher signal level also possesses an increased noise level due to the detection of more background photons, which then lowers the key rate to ensure that the QBER optimization remains within the threshold value of 4.8%. However, since the experimentally observed noise level forms an input to the simulation toolkit, one expects to observe a better match of the simulated results with that of the experiment for the crystal of 30-mm length. Nevertheless, we observe a contradiction there! As per our understanding, the origin of this discrepancy is due to the assumptions involved with the pair-generation-rate calculation while simulating the type-II colinear degenerate SPDC process. The simulated pair-generation rate is directly proportional to the crystal length as well as the input pump intensity at the crystal, which is not observed experimentally. Through separate experimental tests we verify that the singles and the coincidence rates observed at the detectors does not increase linearly with increase in the pump-beam intensity and the crystal length. From these test results, it can be inferred that the pair-generation rate at the crystal does not increase linearly as well. Thus the pair-generation rate obtained from the simulation differs from the actual value obtained from the experiment, resulting in a discrepancy between the final results for the 30-mm crystal length. Additionally, in the case of the key rate estimated from the experiment with the 30-mm crystal, the error values are also higher owing to the increased fluctuations in the measured data points on the coincidence plot. As exemplified for a representative dataset in Fig. 8(a), we observe via the unoptimized window duration of 3 ns around the coincidence peaks that the 30-mm crystal indeed offers

a higher key rate owing to its increased pair rate compared to the 20-mm crystal. More particularly, in this daytime data analysis over 20 runs of the protocol we find the key rate and QBER to be  $47.47 \pm 0.52$  kHz and  $4.67 \pm 0.06\%$ , respectively, for the 20-mm crystal, and  $78.9 \pm 0.48$  kHz and  $7.26 \pm 0.03\%$ , respectively, for the 30-mm crystal. However, a significant lowering of this considerably high QBER (below the threshold) present in the 30-mm datasets due to an increased background noise level, via the optimization strategies, leads to a significant loss of key rate as reported in Tables IV and V. In a nutshell, due to these reasons, the results from the experiment and those from the simulation offer a better match when the length of the crystal is 20 mm compared to the 30-mm case that involves both higher background noise and more fluctuations.

Furthermore, from Tables IV and V, we observe that the key rates obtained via the two optimization algorithms differ given a certain time of the day and length of the crystal. As discussed previously, this increase in the corresponding key rates for strategy B, occurs due to the offered relaxation in key symmetry, over strategy A. For the simulations, however, the increase is more pronounced than that for the experiment. From this observation, we can compare and conclude that the asymmetry in simulated key bits is intrinsically higher than in the experiment. In order to understand the reason behind this intrinsic asymmetry, we need to take an account of the factors that affect the asymmetry of the key bits in both simulation and the experiment. In simulation, the asymmetry is mainly introduced by the asymmetric splitting ratio of the beam splitters. However, in the experiment, apart from the asymmetry in the beam-splitting ratio, the asymmetry in coupling efficiency also plays an important role. In our experiment, the asymmetry due to coupling efficiency effectively reduces the asymmetry introduced by the beam splitter to some extent. This counter error propagation happens as all the components involved in the experiment do not behave identically, thus we can get a different coupling efficiency even when the specification of the fibers and lenses remain the same. As a consequence, the overall asymmetry in the setup decreases and that gets reflected in the results as well. In the simulated version of the coupling, the asymmetry is found to be less pronounced as the various parameters that affect the behavior of the components are directly taken from the specification sheet of the components. For this reason we observe that the difference in the key rates obtained with the two optimization strategies are higher for the simulation than for the experiment.

Lastly, it is important to point out that while the optimization strategies focus on fixed values of QBER and asymmetry in the key string, the estimated key rate is itself associated with a standard deviation in the case of both experiment and simulation. While the sources of

this deviation for the experiment are imperfections of the source, devices, components etc., that for the simulation are primarily captured by the methodology of the simulation, which is based on random number generation from both uniform and normal distributions. While certain aspects of the experiment such as loss through the medium, generation of time stamps, and detection efficiency are simulated using random number generation from a uniform distribution; the alignment errors, timing jitter, etc. are simulated using random numbers from a normal distribution. This, explained in detail in Secs. VI and VII, results in deviation in the outcomes for multiple simulation runs of the protocol, which are shown in the tables.

## IX. CONCLUSION AND OUTLOOK

In this paper, we discuss in detail our in-house developed simulator `qkdSim`, which is created to specifically provide the QKD community with a simulation toolkit that takes into account practical imperfections that could be encountered in an actual experiment. Software available earlier, including `qkdX`, do not contain detailed discussion on attendant physical processes and/or physical components. The `qkdSim` aims to bring in more practical considerations to QKD simulations so that realistic predictions about the key rate and the QBER can be made before investment of resources in developing the physical QKD system. To this end, in our current work, we show how `qkdSim` simulates the free-space-based in-lab demonstration of the B92 protocol in detail. We discuss the development of the `qkdSim` architecture following the Agifall model as well as the simulation methodology involved in modeling the various physical processes and components. A representative key rate from the experiment is  $51 \pm 0.5$  kbit/sec whereas that from the `qkdSim` simulated value is  $52.8 \pm 0.4$  kbit/sec, corresponding to a representative QBER of  $4.79\% \pm 0.01\%$  from both. Having successfully simulated the B92 protocol, which is an example of a QKD protocol that does not use entanglement as the basis for security, we will, in future work, address the applicability of `qkdSim` to entanglement-based QKD. This will bring us a step closer to the desired all-purpose QKD software, which is capable of simulating arbitrary QKD protocols, giving due importance to experimental imperfections and conditions.

## ACKNOWLEDGMENTS

U.S. thanks the Indian Space Research Organisation for support through the QuEST-ISRO research grant. S.C. thanks Professor Ryan S. Bennink for his suggestions on analytical calculation of the SPDC pair rate. The authors thank Professor Gregor Weihs for useful suggestions and discussions; and Ms. Debadrita Ghosh for initial assistance in experimental design.

## APPENDIX A: BOOTSTRAPPING ON MEASUREMENT OUTCOMES

### 1. Motivation

To estimate an optimal choice for the data-acquisition rate (measurement time for each dataset versus number of datasets measured) for an appropriate estimation of key rate, QBER, and asymmetry (or key symmetry) obtained in our experimental demonstration of the B92 protocol.

### 2. Methodology

For a given measurement run of  $M$  sets, each of time  $T$  ps:

1. We choose the start and end time-window marker positions for both coincidence curves (sketched in Fig. 7), i.e., between Alice and Bob V basis (vertical) and Alice and Bob + basis (diagonal). This provides us with three lists of time stamps sorted in ascending order corresponding to each measurement set.
2. We choose a uniform random sequence of length  $M$ .
3. Using a fast binary search algorithm we collect  $k$  ( $\leq T$ ) seconds of data from each of the above three lists, starting from the time-stamp entry found in the random sequence. For those three output lists we calculate the key rate.
4. We repeat step 3 for each element of the uniform random sequence and then obtain the average key rate over those elements.
5. To remove any bias, we randomize (or repeat) steps 2–4 over many iterations. We store the average key rate obtained in each iteration.
6. We finally calculate the ratio: standard deviation (SD) over mean (M) of all the average key rates.
7. We repeat steps 2–6 for a range of  $k$  values and finally obtain a plot over that range.

We use Algorithm 1 to implement the above methodology.

---

#### Algorithm 1 Bootstrap analysis on QKD datasets

---

##### Require:

UNSIGNED INTEGER WINLEFTCURVE1	▷ Left-window marker position (in ps) for the first curve
UNSIGNED INTEGER WINRIGHTCURVE1	▷ Right-window marker position (in ps) for the first curve
UNSIGNED INTEGER WINLEFTCURVE2	▷ Left-window marker position (in ps) for the second curve
UNSIGNED INTEGER WINRIGHTCURVE2	▷ Right-window marker position (in ps) for the second curve
UNSIGNED INTEGER TOTTIME	▷ Measurement runtime for each dataset in ps
UNSIGNED INTEGER TOTDATASETS	▷ Total number of datasets measured
UNSIGNED INTEGER MINCHUNK	▷ Minimum time chunk size (in ps) for bootstrap analysis
UNSIGNED INTEGER MAXCHUNK	▷ Maximum time chunk size (in ps) for bootstrap analysis
UNSIGNED INTEGER STEPSIZE	▷ Stepsize (in ps) to increase the time chunk
UNSIGNED INTEGER TOTITER	▷ Number of iterations
UNSIGNED INTEGER LENDETTIMESTAMPSALICE	▷ Length of Alice’s detected photon time-stamp array
UNSIGNED INTEGER [LENDETTIMESTAMPSALICE] DETTIMESTAMPSALICE	▷ Alice’s detected photon time-stamp array
UNSIGNED INTEGER LENDETTIMESTAMPSBOBPLUS	▷ Length of the time-stamp array for Bob’s “+” polarized photon detection
UNSIGNED INTEGER [LENDETTIMESTAMPSBOBPLUS] DETTIMESTAMPSBOBPLUS	▷ Time-stamp array for Bob’s detection of “+” polarized photons
UNSIGNED INTEGER LENDETTIMESTAMPSBOBVERT	▷ Length of the time-stamp array for Bob’s “V” polarized photon detection
UNSIGNED INTEGER [LENDETTIMESTAMPSBOBVERT] DETTIMESTAMPSBOBVERT	▷ Time-stamp array for Bob’s detection of “V” polarized photons

---

**Ensure:**


---

```

UNSigned INTEGER COUNTER                                ▷ Counts the number of time chunk sizes used for the analysis
REAL [COUNTER] RATIOS                                  ▷ Array of standard deviation over mean values for each time chunk size
UNSigned INTEGER [COUNTER] CHUNKSIZE                  ▷ Array of the corresponding time chunk sizes
1: procedure BOOTSTRAP(WINLEFTCURVE1, WINRIGHTCURVE1, WINLEFTCURVE2, WINRIGHTCURVE2, TOTTIME, TOT-
  DATASETS, MINCHUNK, MAXCHUNK, STEPSIZE, TOTITER, LENDETTIMESTAMPSALICE, DETTIMESTAMPSALICE, LENDET-
  TIMESTAMPSBOBPLUS, DETTIMESTAMPSBOBPLUS, LENDETTIMESTAMPSBOBVERT, DETTIMESTAMPSBOBVERT)
2:   REAL [] RATIOS                                    ▷ Dynamic array for storing standard deviation over mean values
3:   UNSigned INTEGER [] CHUNKSIZE                    ▷ Dynamic array for storing the corresponding time chunk sizes
4:   UNSigned INTEGER COUNTER                          ▷ Counts the number of time chunk sizes
5:   UNSigned INTEGER K                                ▷ Iterator for time chunk sizes
6:   UNSigned INTEGER J                                ▷ Iterator for total number of iterations
7:   UNSigned INTEGER I                                ▷ Iterator for randomly chosen sequence of time stamps
8:   UNSigned INTEGER [TOTDATASETS] SEQUENCE          ▷ Array to store TOTDATASETS uniform random positive
  integers
9:   REAL CUMSUM                                       ▷ Stores cumulative sum of all key rates
10:  REAL [TOTITER] AVGKEYRATE                         ▷ Array to store intermediate average key rates
11:  UNSigned INTEGER [] TEMPALICE                    ▷ Dynamic array to store the temporary list of Alice's detected photon
  time stamps
12:  UNSigned INTEGER [] TEMPBOBPLUS                 ▷ Dynamic array to store the temporary list of Bob's detected ("+"
  polarized) photon time stamps
13:  UNSigned INTEGER [] TEMPBOBVERT                 ▷ Dynamic array to store the temporary list of Bob's detected ("V"
  polarized) photon time stamps
14:  REAL ESTKEYRATE                                   ▷ Stores the estimated values of key rate
15:  COUNTER ← 0                                       ▷ Counter for the processed time chunk sizes initialized to zero
16:  RATIOS ← ∅                                        ▷ Initializes the dynamic array RATIOS as a NULL set
17:  CHUNKSIZE ← ∅                                     ▷ Initializes the dynamic array CHUNKSIZE as a NULL set
18:  for K = MINCHUNK, MAXCHUNK do                  ▷ Loops over the time-chunk sizes
19:    for J = 1, TOTITER do                          ▷ Loops over the specified set of iterations
20:      SEQUENCE ← RANDOMSEQ(0, TOTTIME - K, TOTDATASETS) ▷ Stores TOTDATASETS uniform random
  integers
21:      CUMSUM ← 0                                     ▷ Initializes CUMSUM variable to zero
22:      for I = 1, LENGTH(SEQUENCE) do              ▷ Loops over the uniform random positive integers
23:        TEMPALICE ← GETTIMESTAMPS(I, SEQUENCE, K, DETTIMESTAMPSALICE) ▷ Stores a random
  chunk of time stamps from DETTIMESTAMPSALICE array
24:        TEMPBOBPLUS ← GETTIMESTAMPS(I, SEQUENCE, K, DETTIMESTAMPSBOBPLUS) ▷ Stores a
  random chunk of time stamps from DETTIMESTAMPSBOBPLUS array
25:        TEMPBOBVERT ← GETTIMESTAMPS(I, SEQUENCE, K, DETTIMESTAMPSBOBVERT) ▷ Stores a
  random chunk of time stamps from DETTIMESTAMPSBOBVERT array
26:        ESTKEYRATE ← QKDRATE(WINLEFTCURVE1, WINRIGHTCURVE1, WINLEFTCURVE2, WINRIGHTCURVE2,
  TEMPALICE, TEMPBOBPLUS, TEMPBOBVERT) ▷ Stores the estimated key rate for the given window positions
27:        CUMSUM ← CUMSUM + ESTKEYRATE                ▷ Updates the cumulative summation of key rates
28:      end for
29:      AVGKEYRATE[J] ← CUMSUM/LENGTH(SEQUENCE)      ▷ Stores the key rate averaged over the current
  random sequence length
30:    end for
31:    RATIOS ← RATIOS ∪ { SD(AVGKEYRATE)/MEAN(AVGKEYRATE) } ▷ Appends the SD and M value of
  the average key rates for each chunk size
32:    CHUNKSIZE ← CHUNKSIZE ∪ { SEC(K) }             ▷ Stores the corresponding time chunk size, i.e., K s
33:    COUNTER ← COUNTER + 1                           ▷ Updates the counter by unity
34:  end for
35:  return COUNTER, RATIOS, CHUNKSIZE                ▷ Returns the specified outputs
36: end procedure

```

---

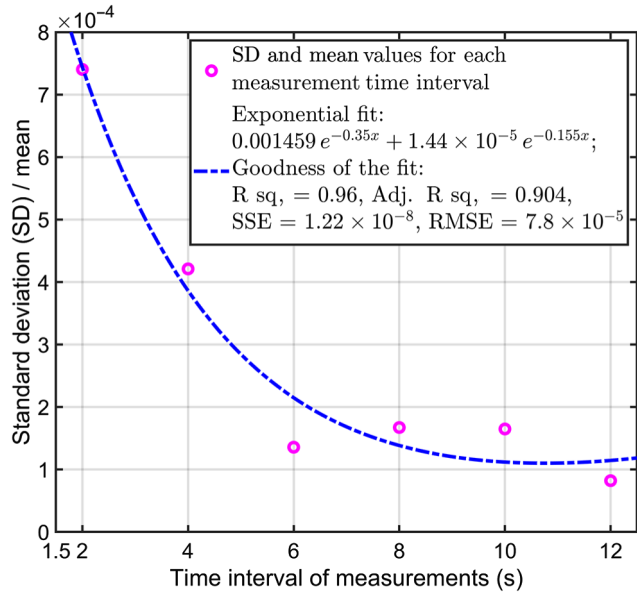


FIG. 29. Bootstrap analysis (SD and M of key rate) plot on a 15-s dataset from a series of 20 measurement runs. Here each run consists of 10 000 iterations. The saturation or convergence effect is supported by the exponential fit to the simulated data points. The goodness of the fit is guaranteed by the low values for sum-of-squares error (SSE) and root-mean-square error (RMSE) as well as close to 1 values for R squared (R sq.) and adjusted R squared (Adj. R sq.).

Please refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

### 3. Result

We test our bootstrap method over a measurement run where ( $M =$ ) 20 sets of ( $T =$ )  $15 \times 10^{12}$  ps data are collected. In order to remove bias, for each set and corresponding to every chosen size of time window, the estimated values of key rate are averaged over 10 000 iterations. The result obtained is presented in Fig. 29. As expected, from this result, we observe that SD and M decreases with increase in the data-collection time of  $k$  s for each measurement and finally saturates beyond a certain value (here say, approximately 9 s) of  $k$  as it approaches  $T$ . To have an appropriate estimate of the key rate, QBER and key symmetry, the measurement runtime (which for our experiment is fixed to 20 sets of 10 s each) should belong to this saturation region.

## APPENDIX B: OPTIMIZATION METHODS FOR DATA ANALYSIS

### 1. Overview

In classical cryptography, asymmetry quantifies the disparity between the number of 0 bits and 1 bits in the key shared by Alice and Bob. For perfect secrecy of the key string, the probability of a certain key bit to be 0 or 1 should be equal for all the bits in the key string. Thus, we can consider that asymmetry in key string can give rise to security issues in QKD protocols. In the implementation of a QKD protocol, the sifted key generated can be asymmetric since the various optical components induce some imperfections.

To account for this asymmetry in key string due to device imperfection, we define two types of optimization strategies, namely A and B, to obtain the optimal values for the key rate, QBER, and key symmetry. Both strategies have their advantages and shortcomings. In strategy A, we maximize the key rate while keeping the QBER below a certain threshold and maintaining approximately 50:50 key symmetry. Here, the asymmetry obtained in the key string is negligible, however the fixing of asymmetry value introduces the possibility for leakage of additional information to the eavesdropper. On the other hand in strategy B, we maximize the key rate with similar constraints on QBER but not on key symmetry. With this technique, the key rate gets increased; however now the security gets compromised to some extent since the probability for obtaining any key string out of  $2^N$  possibilities, where  $N$  is the number of key bits in each key string, remains no longer  $1/2^N$ .

## 2. Methodology

In our experimental version of the B92 protocol, we measure two coincidence curves, which includes coincidences between Alice and Bob's V basis and Alice and Bob's + basis, as sketched in Fig. 7.

*Strategy A:* We use this method to maximize the output key rate while maintaining an asymmetry (also referred to as key symmetry) value of approximately 50% and simultaneously ensuring a quantum bit error rate  $\leq 4.8\%$  in our protocol. For a given dataset in a measurement run of  $S$  datasets, the main steps of this strategy are listed as follows.

1. Detect and mark the coincidence maximum points in both plots.
2. In the first plot, consider an almost wide window, i.e., place the window markers on the left and the right of the coincidence maximum. Ensure that they are located far beyond the FWHM points.
3. Move both window markers with equal step size towards the coincidence maximum and in turn maximize the key rate (i.e., whole area under the curve) within the considered window span. Ensure that the QBER remains below the threshold value of 4.8% during maximization.
4. Retain the optimized window position in the left coincidence plot and in a similar way optimize the left and the right-window marker positions on the right coincidence plot. During this optimization, ensure that approximately 50% symmetry exists between the key rates obtained from both curves and also that the overall QBER from both curves lies within 4.8%.

Lastly, store the optimized key rates, QBERs, and key symmetry (or asymmetry) values for all the  $S$  datasets in three different lists. Calculate the mean value for each of the three lists to obtain the optimal key rate, QBER, and key symmetry for the entire measurement run over  $S$  datasets.

*Strategy B:* We use this method to maximize the output key rate while only ensuring a quantum bit error rate  $\leq 4.8\%$  in our protocol. For a given dataset in a measurement run of  $S$  datasets, the main steps of this strategy are listed as follows.

1. Detect and mark the coincidence maximum points in both plots.
2. In the first plot, consider an almost wide window, i.e., place the window markers on the left and the right of the coincidence maximum. Ensure that they are located far beyond the FWHM points.
3. Move both window markers with equal step size towards the coincidence maximum and in turn maximize the SNR, i.e., the ratio of the areas under the curve (above and below background noise level), within the considered window span.
4. Retain the optimized window position in the left coincidence plot and in a similar way optimize the left and the right-window marker positions on the right (or delayed) coincidence plot.
5. After SNR optimization on the second (or delayed) coincidence plot, move both its window markers in (or out) to achieve the current window span on the first coincidence plot.
6. Alter the window size by moving the slightly markers in (or out) to ensure that the overall QBER does not cross the threshold value of 4.8%.

In the similar approach as used in strategy A, obtain the mean value from each of the three optimized lists for key rates, QBERs, and asymmetry values to report the optimal key rate, QBER, and key symmetry for the entire measurement run.

We use Algorithms 2 and 3 to implement the strategies A and B, respectively.

---

### Algorithm 2 Optimization of QKD results on measured datasets by strategy A

---

#### Require:

- |                                 |  |
|---------------------------------|--|
| REAL STEPSIZEQBER               | ▷ Step size for updating the threshold QBER                  |
| REAL SEEDQBERBOUND              | ▷ Threshold for the seed QBER value to initiate optimization |
| UNSIGNED INTEGER WINLEFTCURVE1  | ▷ Left-window marker position (in ps) for the first curve    |
| UNSIGNED INTEGER WINRIGHTCURVE1 | ▷ Right-window marker position (in ps) for the first curve   |
| UNSIGNED INTEGER WINLEFTCURVE2  | ▷ Left-window marker position (in ps) for the second curve   |
| UNSIGNED INTEGER WINRIGHTCURVE2 | ▷ Right-window marker position (in ps) for the second curve  |
| UNSIGNED INTEGER TOTTIME        | ▷ Measurement runtime for each dataset in seconds            |
| UNSIGNED INTEGER TOTDATASETS    | ▷ Total number of datasets measured in a run                 |
| REAL MINASYBOUND                | ▷ Minimum threshold value for asymmetry estimation           |
| REAL MAXASYBOUND                | ▷ Maximum threshold value for asymmetry estimation           |
| REAL QBERBOUND                  | ▷ Actual threshold value for the QBER estimation             |
-

UNSIGNED INTEGER STEPSIZEWIN	▷ Step size (in ps) for shrinking the window span
UNSIGNED INTEGER LENDETTIMESTAMPSALICE	▷ Length of Alice’s detected photon time-stamp array
UNSIGNED INTEGER [LENDETTIMESTAMPSALICE] DETTIMESTAMPSALICE	▷ Alice’s detected photon time-stamp array
CHAR D	▷ Input for selecting Bob’s measurement in diagonal basis
CHAR R	▷ Input for selecting Bob’s measurement in rectilinear basis
UNSIGNED INTEGER LENDETTIMESTAMPSBOBPLUS	▷ Length of the time-stamp array for Bob’s detection in diagonal basis
UNSIGNED INTEGER [LENDETTIMESTAMPSBOBPLUS] DETTIMESTAMPSBOBPLUS	▷ Time-stamp array for Bob’s photons detected in diagonal basis
UNSIGNED INTEGER LENDETTIMESTAMPSBOBVERT	▷ Length of the time-stamp array for Bob’s detection in rectilinear basis
UNSIGNED INTEGER [LENDETTIMESTAMPSBOBVERT] DETTIMESTAMPSBOBVERT	▷ Time-stamp array for Bob’s photons detected in rectilinear basis
REAL [TOTDATASETS] OPTKEY	▷ Array containing all optimized key rates
REAL [TOTDATASETS] OPTQBER	▷ Array containing all optimized QBERs
REAL [TOTDATASETS] OPTASYMMETRY	▷ Array containing all optimized asymmetry values
UNSIGNED INTEGER CURRDATASET	▷ Pointer to the current dataset in a given run
<b>Ensure:</b>	
REAL [TOTDATASETS] OPTKEY	▷ Updated array with optimized key rates upto the current dataset
REAL [TOTDATASETS] OPTQBER	▷ Updated array with optimized QBERs upto the current dataset
REAL [TOTDATASETS] OPTASYMMETRY	▷ Updated array with optimized asymmetry values upto the current dataset
UNSIGNED INTEGER NEXTDATASET	▷ Pointer to the next dataset in a given run
1: <b>procedure</b> OPTIMIZATIONSTRATEGYA(WINLEFTCURVE1, WINRIGHTCURVE1, WINLEFTCURVE2, WINRIGHTCURVE2, TOTTIME, TOTDATASETS, QBERBOUND, SEEDQBERBOUND, STEPSIZEQBER, MINASYBOUND, MAXASYBOUND, STEPSIZEWIN, LENDETTIMESTAMPSALICE, DETTIMESTAMPSALICE, LENDETTIMESTAMPSBOBPLUS, DETTIMESTAMPSBOBPLUS, LENDETTIMESTAMPSBOBVERT, DETTIMESTAMPSBOBVERT, OPTKEY, OPTQBER, OPTASYMMETRY, CURRDATASET, D, R)	
2: REAL MINQBERBOUND	▷ Minimum threshold value for QBER estimation
3: REAL MAXQBERBOUND	▷ Maximum threshold value for QBER estimation
4: UNSIGNED INTEGER FLAG	▷ Indicator to hasten or delay the convergence of optimized results
5: UNSIGNED INTEGER TEMPSTEPSIZE1	▷ Temporary optimization step size for the first curve
6: UNSIGNED INTEGER TEMPSTEPSIZE2	▷ Temporary optimization step size for the second curve
7: REAL [5] RESULTSCURVE1	▷ Array to store the estimated results for the first curve
8: REAL [5] RESULTSCURVE2	▷ Array to store the estimated results for the second curve
9: REAL KEY1	▷ Stores the optimized key rate for the first curve
10: REAL KEY2	▷ Stores the optimized key rate for the second curve
11: REAL QBER	▷ Stores the optimized QBER for both curves
12: REAL ASYMMETRY	▷ Stores the optimized asymmetry value for both curves
13: REAL UPKEY1	▷ Stores the updated value of the key rate for the first curve
14: REAL UPKEY2	▷ Stores the updated value of the key rate for the second curve
15: REAL UPQBER	▷ Stores the updated value of the QBER for both curves
16: REAL UPASYMMETRY	▷ Stores the updated value of the asymmetry for both curves
17: UNSIGNED INTEGER ITERLEFT	▷ Iterator for the left-window position
18: UNSIGNED INTEGER ITERRIGHT	▷ Iterator for the right-window position
19: UNSIGNED INTEGER LOOPCOUNT	▷ Loop counter
20: KEY1 ← 0	▷ Initializes the optimized key rate for the first curve to zero
21: KEY2 ← 0	▷ Initializes the optimized key rate for the second curve to zero
22: QBER ← 0	▷ Initializes the optimized QBER to zero
23: ASYMMETRY ← 0	▷ Initializes the optimized asymmetry value to zero
24: UPKEY1 ← 0	▷ Initializes the updated key rate for the first curve to zero
25: UPKEY2 ← 0	▷ Initializes the updated key rate for the second curve to zero
26: UPQBER ← 0	▷ Initializes the updated QBER to zero

---

```

27:   UPASYMMETRY ← 0                                ▷ Initializes the updated asymmetry value to zero
28:   LOOPCOUNT ← 0                                  ▷ Initializes the loop counter to zero
29:   while QBER < QBERBOUND do                    ▷ Checks that the QBER remains within threshold
30:     MINQBERBOUND ← SEEDQBERBOUND − STEPSIZEQBER  ▷ Updates the minimum threshold value for
QBER estimation
31:     MAXQBERBOUND ← SEEDQBERBOUND + STEPSIZEQBER  ▷ Updates the maximum threshold value for
QBER estimation
32:     LOOPCOUNT ← LOOPCOUNT + 1                  ▷ Updates the loop counter at each iteration
33:     FLAG ← 0                                       ▷ Initializes the flag to zero
34:     TEMPSTEPSIZE1 ← STEPSIZEWIN                    ▷ Initializes the step-size iterator for the first curve with the
user-input-window step size
35:     ITERLEFT ← WINLEFTCURVE1                       ▷ Initializes the iterator with left-window start position for the first curve
36:     ITERRIGHT ← WINRIGHTCURVE1                     ▷ Initializes the iterator with right-window start position for the first
curve
37:     while ITERRIGHT − ITERLEFT > 0 do           ▷ Iterates until the two windows cross each other
38:       RESULTCURVE1 ← QKD (ITERLEFT, ITERRIGHT, DETTIMESTAMPSALICE, DETTIMESTAMPSBOBPLUS, DET-
TIMESTAMPSBOBVERT, TOTTIME, D)                    ▷ Assigns the estimated results of key rate, QBER, signal, noise, and SNR
for the first curve
39:       if RESULTCURVE1[2] > MAXQBERBOUND then    ▷ Checks if the estimated QBER is above the threshold
maximum
40:         if FLAG = 2 then
41:           TEMPSTEPSIZE1 ← TEMPSTEPSIZE1 − 1      ▷ Hastens the convergence rate by 1 ps
42:         end if
43:         ITERRIGHT ← ITERRIGHT − TEMPSTEPSIZE1    ▷ Shifts the right-window iterator towards the global
or central maximum
44:         ITERLEFT ← ITERLEFT + TEMPSTEPSIZE1      ▷ Shifts the left-window iterator towards the global or
central maximum
45:         FLAG ← 1                                   ▷ Updates the flag
46:       else
47:         if RESULTCURVE1[2] < MINQBERBOUND then  ▷ Checks if estimated QBER is below the threshold
minima
48:           if FLAG = 1 then
49:             TEMPSTEPSIZE1 ← TEMPSTEPSIZE1 − 1    ▷ Hastens the convergence rate by 1 ps
50:           end if
51:           ITERRIGHT ← ITERRIGHT + TEMPSTEPSIZE1  ▷ Shifts the right-window iterator away from the
global or central maximum
52:           ITERLEFT ← ITERLEFT − TEMPSTEPSIZE1    ▷ Shifts the left-window iterator away from the
global or central maximum
53:           FLAG ← 2                                   ▷ Updates the flag
54:         else
55:           break                                     ▷ Quits due to saturation of the QBER at the threshold
56:         end if
57:       end if
58:     end while
59:     TEMPSTEPSIZE2 ← STEPSIZEWIN                    ▷ Initializes the step-size iterator for the second curve with the
user-input-window step size
60:     FLAG ← 0                                       ▷ Resets the flag to zero
61:     ITERLEFT ← WINLEFTCURVE2                       ▷ Reinitializes the iterator with left-window start position for the second
curve
62:     ITERRIGHT ← WINRIGHTCURVE2                     ▷ Reinitializes the iterator with right-window start position for the
second curve
63:     while ITERRIGHT − ITERLEFT > 0 and TEMPSTEPSIZE2 > 0 do  ▷ Iterates until the two windows cross
each other

```

---



---



---

```

64:      RESULTCURVE2 ← QKD(ITERLEFT, ITERRIGHT, DETTIMESTAMPSALICE, DETTIMESTAMPSBOBPLUS, DET-
      TIMESTAMPSBOBVERT, TOTTIME, R)   ▷ Assigns the estimated values of key rate, QBER, signal, noise, and SNR for
      the second curve
65:      KEY2 ← RESULTCURVE2 [1]           ▷ Stores the current key rate obtained for the second curve
66:      QBER ←  $\frac{(\text{RESULTCURVE1}_{[4]}+\text{RESULTCURVE2}_{[4]})\times 100}{(\text{RESULTCURVE1}_{[3]}+\text{RESULTCURVE1}_{[4]}+\text{RESULTCURVE2}_{[3]}+\text{RESULTCURVE2}_{[4]})}$    ▷ Calculates
      the total QBER from the estimated signal and noise values
67:      ASYMMETRY ←  $\frac{(\text{RESULTCURVE2}_{[3]}\times 100)}{(\text{RESULTCURVE1}_{[3]}+\text{RESULTCURVE2}_{[3]})}$    ▷ Calculate the asymmetry value from the
      estimated signal values
68:      if ASYMMETRY > MAXASYBOUND then           ▷ Checks if estimated asymmetry value is above the
      threshold maximum
69:          if FLAG = 2 then
70:              TEMPSTEPSIZE2 ← TEMPSTEPSIZE2 - 1           ▷ Hasten the convergence rate by 1 ps
71:          end if
72:          ITERRIGHT ← ITERRIGHT - TEMPSTEPSIZE2           ▷ Shifts the right window towards global or central
      maximum
73:          ITERLEFT ← ITERLEFT + TEMPSTEPSIZE2             ▷ Shifts the left window towards global or central
      maximum
74:          FLAG ← 1                                         ▷ Updates the flag
75:      else
76:          if ASYMMETRY < MINASYBOUND then           ▷ Checks if estimated asymmetry value is below the
      threshold minimum
77:              if FLAG = 1 then
78:                  TEMPSTEPSIZE2 ← TEMPSTEPSIZE2 - 1           ▷ Hasten the convergence rate by 1 ps
79:              end if
80:              ITERRIGHT ← ITERRIGHT + TEMPSTEPSIZE2       ▷ Shifts the right window away from the global or
      central maximum
81:              ITERLEFT ← ITERLEFT - TEMPSTEPSIZE2         ▷ Shifts the left window away from the global or
      central maximum
82:              FLAG ← 2                                     ▷ Updates the flag
83:          else
84:              break                                       ▷ Quits due to the saturation of the asymmetry value at the threshold
85:          end if
86:      end if
87:  end while
88:  if QBER < QBERBOUND then
89:      UPKEY1 ← KEY1                                       ▷ Stores the previous value of key rate for the first curve
90:      UPKEY2 ← KEY2                                       ▷ Stores the previous key rate obtained for the second curve
91:      UPQBER ← QBER                                       ▷ Stores the previous QBER obtained for both curves
92:      UPASYMMETRY ← ASYMMETRY                             ▷ Stores the previous asymmetry value obtained for both curves
93:  end if
94:  end while
95:  OPTKEY [CURRDATASET] ← UPKEY1 + UPKEY2           ▷ Stores the optimized key rates for the current measurement
      set
96:  OPTQBER [CURRDATASET] ← UPQBER                       ▷ Stores the optimized QBER for the current measurement set
97:  OPTASYMMETRY [CURRDATASET] ← UPASYMMETRY           ▷ Stores the asymmetry value for the current
      measurement set
98:  NEXTDATASET ← CURRDATASET + 1                       ▷ Updates the pointer to the next dataset
99:  return OPTKEY, OPTQBER, OPTASYMMETRY, NEXTDATASET   ▷ Returns the specified outputs
100: end procedure

```

---

**Algorithm 3** Optimization of QKD results on measured datasets by strategy B**Require:**

REAL STEPSIZEQBER	▷ Step size for updating the threshold QBER
REAL GLOBALMAX1	▷ Global maximum point of the first curve
REAL GLOBALMAX2	▷ Global maximum point of the second curve
REAL MINQBERBOUND	▷ Minimum threshold value for QBER estimation
REAL MAXQBERBOUND	▷ Maximum threshold value for QBER estimation
UNSIGNED INTEGER WINLEFTCURVE1	▷ Left-window marker position (in ps) for the first curve
UNSIGNED INTEGER WINRIGHTCURVE1	▷ Right-window marker position (in ps) for the first curve
UNSIGNED INTEGER WINLEFTCURVE2	▷ Left-window marker position (in ps) for the second curve
UNSIGNED INTEGER WINRIGHTCURVE2	▷ Right-window marker position (in ps) for the second curve
UNSIGNED INTEGER TOTTIME	▷ Measurement runtime for each dataset in seconds
UNSIGNED INTEGER TOTDATASETS	▷ Total number of datasets measured in a run
REAL QBERBOUND	▷ Actual threshold value for the QBER estimation
UNSIGNED INTEGER STEPSIZEWIN	▷ Step size (in ps) for shrinking the window span
UNSIGNED INTEGER LENDETTIMESTAMPSALICE	▷ Length of Alice's detected photon time-stamp array
UNSIGNED INTEGER [LENDETTIMESTAMPSALICE] DETTIMESTAMPSALICE	▷ Alice's detected photon time-stamp array
UNSIGNED INTEGER LENDETTIMESTAMPSBOBPLUS	▷ Length of the time-stamp array for Bob's detection in diagonal basis
UNSIGNED INTEGER [LENDETTIMESTAMPSBOBPLUS] DETTIMESTAMPSBOBPLUS	▷ Time-stamp array for Bob's photons detected in diagonal basis
UNSIGNED INTEGER LENDETTIMESTAMPSBOBVERT	▷ Length of the time-stamp array for Bob's detection in rectilinear basis
UNSIGNED INTEGER [LENDETTIMESTAMPSBOBVERT] DETTIMESTAMPSBOBVERT	▷ Time-stamp array for Bob's photons detected in rectilinear basis
REAL [TOTDATASETS] OPTKEY	▷ Array containing all optimized key rates
REAL [TOTDATASETS] OPTQBER	▷ Array containing all optimized QBERs
REAL [TOTDATASETS] OPTASYMMETRY	▷ Array containing all optimized asymmetry values
UNSIGNED INTEGER CURRDATASET	▷ Pointer to the current dataset in a given run
CHAR D	▷ Input for selecting Bob's measurement in diagonal basis
CHAR R	▷ Input for selecting Bob's measurement in rectilinear basis

**Ensure:**

REAL [TOTDATASETS] OPTKEY	▷ Updated array with optimized key rates upto the current dataset
REAL [TOTDATASETS] OPTQBER	▷ Updated array with optimized QBERs upto the current dataset
REAL [TOTDATASETS] OPTASYMMETRY	▷ Updated array with optimized asymmetry values upto the current dataset
UNSIGNED INTEGER NEXTDATASET	▷ Pointer to the next dataset in a given run
1: <b>procedure</b> OPTIMIZATIONSTRATEGYB(GLOBALMAX1, GLOBALMAX2, MINQBERBOUND, MAXQBERBOUND, WINLEFTCURVE1, WINRIGHTCURVE1, WINLEFTCURVE2, WINRIGHTCURVE2, TOTTIME, TOTDATASETS, QBERBOUND, SEEDQBERBOUND, STEPSIZEQBER, STEPSIZEWIN, LENDETTIMESTAMPSALICE, DETTIMESTAMPSALICE, LENDETTIMESTAMPSBOBPLUS, DETTIMESTAMPSBOBPLUS, LENDETTIMESTAMPSBOBVERT, DETTIMESTAMPSBOBVERT, OPTKEY, OPTQBER, OPTASYMMETRY, CURRDATASET, D, R)	
2: UNSIGNED INTEGER FLAG	▷ Indicator to hasten or delay the convergence of optimized results
3: UNSIGNED INTEGER TEMPSTEPSIZE1	▷ Temporary optimization step size for the first curve
4: UNSIGNED INTEGER TEMPSTEPSIZE2	▷ Temporary optimization step size for the second curve
5: REAL [5] RESULTSCURVE1	▷ Array to store the estimated results for the first curve
6: REAL [5] RESULTSCURVE2	▷ Array to store the estimated results for the second curve
7: REAL SNR1	▷ Stores the optimized signal-to-noise ratio for the first curve
8: REAL SNR2	▷ Store the optimized signal-to-noise ratio for the second curve
9: REAL QBER	▷ Stores the optimized QBER for both curves
10: REAL ASYMMETRY	▷ Stores the optimized asymmetry value for both curves

---

```

11: REAL REMSNR1           ▷ Stores the previous value of the signal-to-noise ratio obtained for the first curve
12: REAL REMSNR2           ▷ Stores the previous value of the signal-to-noise ratio obtained for the second curve
13: UNSIGNED INTEGER WINSPAN1           ▷ Stores the window span for the first curve
14: UNSIGNED INTEGER WINSPAN2           ▷ Stores the window span for the second curve
15: UNSIGNED INTEGER ITERLEFT1           ▷ Iterator for the left-window marker position for first curve
16: UNSIGNED INTEGER ITERRIGHT1          ▷ Iterator for the right-window marker position for first curve
17: UNSIGNED INTEGER ITERLEFT2           ▷ Iterator for the left-window marker position for second curve
18: UNSIGNED INTEGER ITERRIGHT2          ▷ Iterator for the right-window marker position for second curve
19: REMSNR2 ← 0             ▷ Initializes the previously optimized SNR for the second curve to zero
20: FLAG ← 0                ▷ Initializes the flag to zero
21: TEMPSTEPSIZE1 ← STEPSIZEWIN           ▷ Initializes the step-size iterator for the first curve with the
user-input-window step size
22: ITERLEFT1 ← WINLEFTCURVE1           ▷ Initializes the iterator with left-window start position for the first curve
23: ITERRIGHT1 ← WINRIGHTCURVE1        ▷ Initializes the iterator with right-window start position for the first curve
24: REMSNR1 ← 0                ▷ Initializes the previously optimized SNR for the first curve to zero
25: while ITERRIGHT1 – ITERLEFT1 > 0 do           ▷ Loops until the two window markers cross each other
26:     RESULTCURVE1 ← QKD (ITERLEFT1, ITERRIGHT1, DETTIMESTAMPSALICE, DETTIMESTAMPSBOBPLUS, DET-
TIMESTAMPSBOBVERT, TOTTIME, D)           ▷ Assigns the estimated results of key rate, QBER, signal, noise, and SNR
for the first curve
27:     if RESULTCURVE1 [5] > REMSNR1 then           ▷ Checks if the estimated SNR is increasing
28:         if FLAG = 2 then
29:             TEMPSTEPSIZE1 ← TEMPSTEPSIZE1 – 1           ▷ Hastens the convergence rate by 1 ps
30:         end if
31:         ITERRIGHT1 ← ITERRIGHT1 – TEMPSTEPSIZE1           ▷ Shifts the right-window marker towards global
maxima
32:         ITERLEFT1 ← ITERLEFT1 + TEMPSTEPSIZE1           ▷ Shifts the left-window marker towards global maxima
33:         FLAG ← 1                                           ▷ Updates the flag
34:     else
35:         if RESULTCURVE1 [5] < REMSNR1 then           ▷ Checks if the estimated SNR is decreasing
36:             if FLAG = 1 then
37:                 TEMPSTEPSIZE1 ← TEMPSTEPSIZE1 – 1           ▷ Hastens the convergence rate by 1 ps
38:             end if
39:             ITERRIGHT1 ← ITERRIGHT1 + TEMPSTEPSIZE1           ▷ Shifts the right-window marker away from the
global maxima
40:             ITERLEFT1 ← ITERLEFT1 – TEMPSTEPSIZE1           ▷ Shifts the left-window marker away from the global
maxima
41:             FLAG ← 2                                           ▷ Updates the flag
42:         else
43:             break                                           ▷ Quits as further maximization of SNR is not possible
44:         end if
45:     end if
46:     REMSNR1 ← SNR1           ▷ Saves the current optimized SNR for the first curve
47: end while
48: WINSPAN1 ← (GLOBALMAX1 – ITERLEFT1) + (ITERRIGHT1 – GLOBALMAX1)           ▷ Saves the optimized window
span for the first curve
49: TEMPSTEPSIZE2 ← STEPSIZEWIN           ▷ Initializes the step-size iterator for the second curve with the user-input
step size
50: FLAG ← 0                ▷ Reinitializing the flag to zero
51: ITERLEFT2 ← WINLEFTCURVE2           ▷ Initializes the iterator with left-window start position for the second curve
52: ITERRIGHT2 ← WINRIGHTCURVE2        ▷ Initializes the iterator with right-window start position for the second
curve
53: REMSNR2 ← 0             ▷ Initializes the previously optimized SNR for the second curve to zero

```

---

---



---

```

54:   while ITERRIGHT2 – ITERLEFT2 > 0 and TEMPSTEPSIZE2 > 0 do           ▷ Loops until the two window markers
      cross each other
55:     RESULTCURVE2 ← QKD(ITERLEFT2, ITERRIGHT2, DETTIMESTAMPSALICE, DETTIMESTAMPSBOBPLUS, DET-
      TIMESTAMPSBOBVERT, TOTTIME, R)   ▷ Assigns the estimated results of key rate, QBER, signal, noise, and SNR for
      the second curve
56:     if RESULTCURVE2 [5] > REMSNR2 then                                   ▷ Checks if the estimated SNR is increasing
57:       if FLAG = 2 then
58:         TEMPSTEPSIZE2 ← TEMPSTEPSIZE2 – 1                               ▷ Hastens the convergence rate by 1 ps
59:       end if
60:       ITERRIGHT2 ← ITERRIGHT2 – TEMPSTEPSIZE2                         ▷ Shifts the right window towards global maxima
61:       ITERLEFT2 ← ITERLEFT2 + TEMPSTEPSIZE2                           ▷ Shifts the left window towards global maxima
62:       FLAG ← 1                                                         ▷ Updates the flag
63:     else
64:       if RESULTCURVE2 [5] < REMSNR2 then                               ▷ Checks if estimated SNR is decreasing
65:         if FLAG = 1 then
66:           TEMPSTEPSIZE2 ← TEMPSTEPSIZE2 – 1                           ▷ Hastens the convergence rate by 1 ps
67:         end if
68:         ITERRIGHT2 ← ITERRIGHT2 + TEMPSTEPSIZE2                       ▷ Shifts the right-window marker away from the
      global maxima
69:         ITERLEFT2 ← ITERLEFT2 – TEMPSTEPSIZE2                         ▷ Shifts the left-window marker away from the global
      maxima
70:         FLAG ← 2                                                         ▷ Updates the flag
71:       else
72:         break                                                         ▷ Quits as further maximization of SNR is not possible
73:       end if
74:     end if
75:   end while
76:   WINSPAN2 ← (GLOBALMAX2 – ITERLEFT2) + (ITERRIGHT2 – GLOBALMAX2)   ▷ Saves the optimized window
      span for the second curve
77:   ADJUSTBY ←  $\frac{(\text{WINSPAN1} - \text{WINSPAN2})}{2}$                                ▷ Stores the difference in window span sizes between the first and
      second curve
78:   ITERLEFT2 ← ITERLEFT2 – CEIL (ADJUSTBY)                               ▷ Reinitializes the iterator with adjusted position for the
      left-window marker of the second curve
79:   ITERRIGHT2 ← ITERRIGHT2 + FLOOR (ADJUSTBY)                           ▷ Reinitializes the iterator with adjusted position for the
      right-window marker of the second curve
80:   RESULTCURVE2 ← QKD(ITERLEFT2, ITERRIGHT2, DETTIMESTAMPSALICE, DETTIMESTAMPSBOBPLUS, DETTIMES-
      TAMPBOBVERT, TOTTIME, R)   ▷ Reassigns the estimated results of key rate, QBER, signal, noise, and SNR for the
      second curve
81:   QBER ←  $\frac{(\text{RESULTCURVE1} [4] + \text{RESULTCURVE2} [4]) \times 100}{(\text{RESULTCURVE1} [3] + \text{RESULTCURVE1} [4] + \text{RESULTCURVE2} [3] + \text{RESULTCURVE2} [4])}$    ▷ Calculates the total
      QBER from the estimated signal and noise values
82:   ASYMMETRY ←  $\frac{(\text{RESULTCURVE2} [3] \times 100)}{(\text{RESULTCURVE1} [3] + \text{RESULTCURVE2} [3])}$    ▷ Calculates the asymmetry value from the
      estimated signal values
83:   while QBER > MAXQBERBOUND or QBER < MINQBERBOUND do               ▷ Optimizes both window spans to restrict
      the overall QBER within the threshold value
84:     if QBER < MINQBERBOUND then                                       ▷ Checks if the estimated QBER is below the desired range
85:       ITERLEFT1 ← ITERLEFT1 – STEPSIZEQBER                             ▷ Shift the left window away from the global maxima of
      the first curve
86:       ITERRIGHT1 ← ITERRIGHT1 + STEPSIZEQBER                          ▷ Shift the right window away from the global maxima
      of the first curve
87:       ITERLEFT2 ← ITERLEFT2 – STEPSIZEQBER                             ▷ Shift the left window away from the global maxima of
      the second curve

```

---

---

```

88:         ITERRIGHT2 ← ITERRIGHT2 + STEPSIZEQBER    ▷ Shift the right window away from the global maxima
of the second curve
89:         else
90:         ITERLEFT1 ← ITERLEFT1 + STEPSIZEQBER    ▷ Shifts the left window towards the global maxima of the
first curve
91:         ITERRIGHT1 ← ITERRIGHT1 − STEPSIZEQBER    ▷ Shifts the right window towards the global maxima of
the first curve
92:         ITERLEFT2 ← ITERLEFT2 + STEPSIZEQBER    ▷ Shifts the left window towards global maxima of the
second curve
93:         ITERRIGHT2 ← ITERRIGHT2 − STEPSIZEQBER    ▷ Shifts the right window towards global maxima of the
second curve
94:         end if
95:         RESULTCURVE1 ← QKD(ITERLEFT1, ITERRIGHT1, DETTIMESTAMPSALICE, DETTIMESTAMPSBOBPLUS, DET-
TIMESTAMPSBOBVERT, TOTTIME, D) ▷ Re-assigns the estimated results of Key rate, QBER, signal, noise and SNR for
the first curve
96:         RESULTCURVE2 ← QKD(ITERLEFT2, ITERRIGHT2, DETTIMESTAMPSALICE, DETTIMESTAMPSBOBPLUS, DET-
TIMESTAMPSBOBVERT, TOTTIME, R) ▷ Reassigns the estimated results of key rate, QBER, signal, noise, and SNR for
the second curve
97:         QBER ←  $\frac{(\text{RESULTCURVE1}_{[4]} + \text{RESULTCURVE2}_{[4]}) \times 100}{(\text{RESULTCURVE1}_{[3]} + \text{RESULTCURVE1}_{[4]} + \text{RESULTCURVE2}_{[3]} + \text{RESULTCURVE2}_{[4]})}$     ▷ Updates the
total QBER for the revised window positions
98:         ASYMMETRY ←  $\frac{(\text{RESULTCURVE2}_{[3]} \times 100)}{(\text{RESULTCURVE1}_{[3]} + \text{RESULTCURVE2}_{[3]})}$     ▷ Updates the key symmetry value
corresponding to the revised window positions
99:         end while
100:        OPTKEY [CURRDATASET] ← RESULTCURVE1 [1] + RESULTCURVE2 [1]    ▷ Stores the optimized key rates for
the current measurement set
101:        OPTQBER [CURRDATASET] ← QBER    ▷ Stores the optimized QBER for the current measurement set
102:        OPTASYMMETRY [CURRDATASET] ← ASYMMETRY    ▷ Stores the key symmetry value for the current
measurement set
103:        NEXTDATASET ← CURRDATASET + 1    ▷ Updates the pointer to the next dataset
104:        return OPTKEY, OPTQBER, OPTASYMMETRY, NEXTDATASET    ▷ Returns the specified outputs
105:    end procedure

```

---

Refer to Appendix L for details on the usage of data types and libraries, at various instances, in the above two optimization algorithms.

---

**Algorithm 4** Calculates the estimated values of key rate, QBER, signal, noise, and SNR from detected photon time stamps by Alice and Bob

---

**Require:**

```

UNSIGNED INTEGER WINDOWLEFT    ▷ Left-window marker position (in ps)
UNSIGNED INTEGER WINDOWRIGHT    ▷ Right-window marker position (in ps)
UNSIGNED INTEGER LENDETTIMESTAMPSALICE    ▷ Length of Alice's detected photon time-stamp array
UNSIGNED INTEGER [LENDETTIMESTAMPSALICE] DETTIMESTAMPSALICE    ▷ Alice's detected photon time-stamp
array
UNSIGNED INTEGER LENDETTIMESTAMPSBOBDIAG    ▷ Length of the time-stamp array for Bob's detection in
diagonal basis
CHAR BASISCHOICE    ▷ Choice of measurement basis: rectilinear (R) or diagonal (D)
UNSIGNED INTEGER [LENDETTIMESTAMPSBOBDIAG] DETTIMESTAMPSBOBDIAG    ▷ Time-stamp array for
Bob's detection in diagonal basis
UNSIGNED INTEGER LENDETTIMESTAMPSBOBRECT    ▷ Length of the time-stamp array for Bob's detection in
rectilinear basis

```

---

---

```

UNUNSIGNED INTEGER [LENDETTIMESTAMPSBOBRECT] DETTIMESTAMPSBOBRECT    ▷ Time-stamp array for Bob's
detection in rectilinear basis
UNUNSIGNED INTEGER MEASTIME                                          ▷ Total data collection time in seconds
Ensure:
REAL KEYRATE                                                         ▷ Estimated key rate
REAL QBER                                                            ▷ Estimated QBER
UNUNSIGNED INTEGER SIGNAL                                           ▷ Estimated signal value
UNUNSIGNED INTEGER NOISE                                            ▷ Estimated noise value
REAL SNR                                                             ▷ Estimated signal-to-noise ratio
1: function QKD(WINDOWLEFT, WINDOWRIGHT, LENDETTIMESTAMPSALICE, DETTIMESTAMPSALICE, LENDETTIMES-
TIMESTAMPSBOBDIAG, DETTIMESTAMPSBOBDIAG, LENDETTIMESTAMPSBOBRECT, DETTIMESTAMPSBOBRECT, MEASTIME,
BASISCHOICE)
2:   UNSIGNED INTEGER I                                               ▷ Iterates over Eve's detection list
3:   UNSIGNED INTEGER J                                               ▷ Iterates over Eve's detection list
4:   UNSIGNED INTEGER SIGNAL                                           ▷ Stores the estimated signal value along matched bases
5:   UNSIGNED INTEGER NOISE                                           ▷ Stores the estimated noise value along mismatched bases
6:   REAL KEYRATE                                                     ▷ Stores the estimated key rate
7:   REAL QBER                                                         ▷ Stores the estimated QBER
8:   switch BASISCHOICE do                                           ▷ Collects the measurement basis choice to estimate the signal and noise along it
9:     case D                                                         ▷ Measurements along the chosen diagonal basis
10:      I ← 1                                                         ▷ Initializes the iterator to the first element on Alice's detection list
11:      J ← 1                                                         ▷ Initializes the iterator to the first element on Bob's detection list
12:      SIGNAL ← 0                                                    ▷ Initializes the counter for coincidences along matched bases to zero
13:      while I ≤ LENDETTIMESTAMPSALICE & J ≤ LENDETTIMESTAMPSBOBDIAG do    ▷ Loops over both time
stamp lists
14:        if DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [I] < WINDOWLEFT then    ▷ Rejects any
detection beyond the current left-window position
15:          J ← J + 1                                                 ▷ Increments the detection interval
16:        end if
17:        if DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [I] ≤ WINDOWRIGHT    &
DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [i] ≥ WINDOWLEFT then    ▷ Rejects any detection outside the
current window span
18:          SIGNAL ← SIGNAL + 1                                       ▷ Increments the signal counter
19:          I ← I + 1                                                 ▷ Increments the pointer on Alice's detection list
20:          J ← J + 1                                                 ▷ Increments the pointer on Bob's detection list for the diagonal basis
21:        end if
22:        if DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [I] > WINDOWRIGHT then    ▷ Rejects any
detection beyond the right-window position
23:          I ← I + 1                                                 ▷ Decrements the detection interval
24:        end if
25:      end while
26:      I ← 1                                                         ▷ Reinitializes the iterator to the first element of Eve's detection list
27:      J ← 1                                                         ▷ Reinitializes the iterator to the first element of Eve's detection list
28:      NOISE ← 0                                                     ▷ Initializes the counter for coincidences along mismatched bases to zero
29:      while I ≤ LENDETTIMESTAMPSALICE & J ≤ LENDETTIMESTAMPSBOBRECT do    ▷ Loops over both
time-stamp lists
30:        if DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [I] < WINDOWLEFT then    ▷ Rejects any
detection beyond the current left-window position
31:          J ← J + 1                                                 ▷ Increments the detection interval
32:        end if
33:        if DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [I] ≤ WINDOWRIGHT    &
DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [I] ≥ WINDOWLEFT then    ▷ Rejects any detection outside
the current window span

```

---

---

```

34:          NOISE ← NOISE + 1                                ▷ Increments the noise counter
35:          I ← I + 1                                        ▷ Increments the pointer on Alice's detection list
36:          J ← J + 1                                        ▷ Increments the pointer on Bob's detection list for the rectilinear basis
37:          end if
38:          if DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [I] > WINDOWRIGHT then    ▷ Rejects any
detection beyond the current right-window position
39:          I ← I + 1                                        ▷ Decrements the detection interval
40:          end if
41:          end while
42:          case R                                          ▷ Measurements along the chosen rectilinear basis
43:          I ← 1                                          ▷ Initializes the iterator to the first element on Alice's detection list
44:          J ← 1                                          ▷ Initializes the iterator to the first element on Bob's detection list
45:          SIGNAL ← 0                                     ▷ Initializes the counter for coincidences along matched bases to zero
46:          while I ≤ LENDETTIMESTAMPSALICE & J ≤ LENDETTIMESTAMPSBOBRECT do          ▷ Loops over both
time-stamp lists
47:          if DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [I] < WINDOWLEFT then    ▷ Rejects any
detection beyond the current left-window position
48:          J ← J + 1                                        ▷ Increments the detection interval
49:          end if
50:          if          DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [I] ≤ WINDOWRIGHT      &
DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [i] ≥ WINDOWLEFT then    ▷ Rejects any detection outside the
current window span
51:          SIGNAL ← SIGNAL + 1                            ▷ Increments the signal counter
52:          I ← I + 1                                        ▷ Increments the pointer on Alice's detection list
53:          J ← J + 1                                        ▷ Increments the pointer on Bob's detection list for the rectilinear basis
54:          end if
55:          if DETTIMESTAMPSBOBRECT [J] − DETTIMESTAMPSALICE [I] > WINDOWRIGHT then    ▷ Rejects any
detection beyond the right-window position
56:          I ← I + 1                                        ▷ Decrements the detection interval
57:          end if
58:          end while
59:          I ← 1                                          ▷ Reinitializes the iterator to the first element of Eve's detection list
60:          J ← 1                                          ▷ Reinitializes the iterator to the first element of Eve's detection list
61:          NOISE ← 0                                     ▷ Initializes the counter for coincidences along mismatched bases to zero
62:          while I ≤ LENDETTIMESTAMPSALICE & J ≤ LENDETTIMESTAMPSBOBDIAG do          ▷ Loops over both
time-stamp lists
63:          if DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [I] < WINDOWLEFT then    ▷ Rejects any
detection beyond the current left-window position
64:          J ← J + 1                                        ▷ Increments the detection interval
65:          end if
66:          if          DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [I] ≤ WINDOWRIGHT      &
DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [I] ≥ WINDOWLEFT then    ▷ Rejects any detection outside
the current window span
67:          NOISE ← NOISE + 1                            ▷ Increments the noise counter
68:          I ← I + 1                                        ▷ Increments the pointer on Alice's detection list
69:          J ← J + 1                                        ▷ Increments the pointer on Bob's detection list for the diagonal basis
70:          end if
71:          if DETTIMESTAMPSBOBDIAG [J] − DETTIMESTAMPSALICE [I] > WINDOWRIGHT then    ▷ Rejects any
detection beyond the current right-window position
72:          I ← I + 1                                        ▷ Decrements the detection interval
73:          end if
74:          end while
75:          KEYRATE ←  $\frac{\text{SIGNAL} + \text{NOISE}}{\text{MEASTIME}}$                                 ▷ Stores the final estimated key rate

```

---

---

```

76:   QBER ←  $\frac{\text{NOISE} \times 100}{\text{SIGNAL} + \text{NOISE}}$                                 ▷ Stores the final estimated QBER in %
77:   SNR ←  $\frac{\text{SIGNAL}}{\text{NOISE}}$                                           ▷ Stores the final estimated signal-to-noise ratio
78:   return KEYRATE, QBER, SIGNAL, NOISE, SNR                        ▷ Returns the specified outputs in the order: key rate, QBER,
   signal, noise, and SNR
79: end function

```

---

Refer to Appendix L for details on the usage of various data types and libraries in the above algorithmic function.

### APPENDIX C: QUASI-PHASE-MATCHING TEMPERATURE FOR TYPE-II SPDC IN A PPKTP CRYSTAL

Considering the law of conservation of momentum (sketched in Fig. 23) and colinear parametric interaction (sketched in Fig. 22) for a SPDC process, the quasi-phase-matching condition [87] for a periodically poled crystal can be described with the relations

$$K_p \cos \theta_p = K_s \cos \theta_s + K_i \cos \theta_i + \frac{2\pi}{\Lambda(T)}, \quad (\text{C1})$$

$$K_s \sin \theta_s = K_i \sin \theta_i.$$

Here,  $K_p$ ,  $K_s$ , and  $K_i$  represents the magnitude of the momentum vectors for the pump ( $p$ ), signal ( $s$ ) and idler ( $i$ ) photons, respectively. Also,  $\Lambda(T)$  denotes the poling period of the crystal, which is dependent on temperature  $T$  and  $\theta_{s(i)}$  represents the angle that the signal's (idler's) momentum vector makes with that of the pump propagation along  $z$  direction. Note that the quasi-phase-matching condition described in Eq. (C1) can be derived by solving the set of coupled differential equations for the electric field amplitude of the pump, signal, and idler as shown in Ref. [92].

Substituting the conditions of colinearity (i.e.,  $\theta_s = \theta_i = 0$ ) and degeneracy (i.e.,  $\omega_s = \omega_i = \omega_p/2$ ) in Eq. (C1); and also finally considering  $K_h = 2\pi n_h/\lambda_h$ , where  $h \in \{p, s, i\}$ , we get

$$\frac{2\pi n_p}{\lambda_p} = \frac{2\pi n_s}{\lambda_s} + \frac{2\pi n_i}{\lambda_i} + \frac{2\pi}{\Lambda(T)}, \quad (\text{C2})$$

where  $n_p$ ,  $n_s$ , and  $n_i$  denote the nonlinear refractive indices and  $\lambda_p$ ,  $\lambda_s$ , and  $\lambda_i$  represent the wavelengths of the pump, signal, and idler photons, respectively.

Now, each of these nonlinear refractive indices can be compactly stated as functions of temperature  $T$ , polarization direction  $\hat{s}$ , and respective wavelength  $\lambda_h$ , i.e.,  $n_h = f(T, \hat{s}, \lambda_h)$  where  $h \in \{p, s, i\}$ . So, a thermal expansion of the poling period results in [93]

$$\Lambda(T) = \Lambda_0 \{1 + \alpha(T - T_0) + \beta(T - T_0)^2\}, \quad (\text{C3})$$

where  $T_0 = 25^\circ\text{C}$  (room temperature) and  $\Lambda_0 = 10 \mu\text{m}$  (poling period of our crystal). Also for a KTP crystal,  $\alpha = (6.7 \pm 0.7) \times 10^{-6} \text{ }^\circ\text{C}^{-1}$  and  $\beta = (11 \pm 2) \times 10^{-9} \text{ }^\circ\text{C}^{-1}$  [93]. Additionally, a thermal expansion of the refractive indices gives [93,94]

$$n(\lambda, T) = n(\lambda, T = T_0) + \left. \frac{\partial n}{\partial T} \right|_{(\lambda, T=T_0)} (T - T_0) + \left. \frac{\partial^2 n}{\partial T^2} \right|_{(\lambda, T=T_0)} (T - T_0)^2. \quad (\text{C4})$$

The second and third term in Eq. (C4) can be obtained from the empirical results in Refs. [93,94]. While, the value of the first term, can be obtained by solving either of the following Sellmeier equations for a PPKTP crystal [89–91]:

$$\text{one pole: } n^2(\lambda, T = T_0) = A + \frac{B}{1 - C\lambda^{-2}} - D\lambda^2, \quad \text{or} \quad (\text{C5})$$

$$\text{two pole: } n_z^2(\lambda, T = T_0) = A + \frac{B}{1 - C\lambda^{-2}} - \frac{D}{1 - E\lambda^{-2}} - F\lambda^2. \quad (\text{C6})$$

The value of the constants  $A$  to  $F$  can be again obtained from empirical results in Refs. [89–91]. Here, it is important to note that the two-pole Sellmeier equation provides a better accuracy in the obtained result for the first term than the one-pole version.



Lastly, by considering a pump wavelength of 405 nm and using the above Eqs. (C1)–(C6), we numerically calculate our phase-matching temperature to be 44.4 °C with a signal (or idler) wavelength of 810 nm.

## APPENDIX D: ANALYTICAL DEDUCTION OF SPDC PAIR RATE FROM FIRST PRINCIPLES

In the previous Appendix C, we deduce the phase-matching temperature for our PPKTP crystal. In this appendix, we continue with that result to theoretically derive the expressions and then use it later to numerically simulate the values for the pair-generation probability and the pair-generation rate of our SPDC process.

### 1. Quantization of the electromagnetic field

Using vector potential  $A(\vec{r}, t)$ , the classical electric  $\vec{E}(\vec{r}, t)$  and magnetic field  $\vec{B}(\vec{r}, t)$  can be expressed as [95]

$$\vec{E}(\vec{r}, t) = -\frac{\partial \vec{A}(\vec{r}, t)}{\partial t}, \quad (\text{D1a})$$

$$\vec{B}(\vec{r}, t) = \vec{\nabla} \times \vec{A}(\vec{r}, t), \quad (\text{D1b})$$

where  $\vec{r}$  is the direction vector and  $t$  denotes time.

By substituting Eqs. (D1a) and (D1b) in Maxwell's equations and thereby using  $\vec{\nabla} \cdot \vec{A}(\vec{r}, t) = 0$  from Coulomb gauge, we can arrive at the following wave equation [96]:

$$\vec{\nabla}^2 \vec{A}(\vec{r}, t) = \frac{1}{c^2} \frac{\partial^2 \vec{A}(\vec{r}, t)}{\partial t^2}, \quad (\text{D2})$$

where  $c = 1/\sqrt{\mu_0 \epsilon_0}$  is the speed of light in vacuum, with  $\mu_0$  being the free-space magnetic permeability and  $\epsilon_0$  representing the free-space electric permittivity.

Considering period boundary conditions in a cubic space of side length  $L$ , the contributions from various modes, say  $k$ , to the vector field can be expressed as [97]

$$\vec{A}(\vec{r}, t) = \sum_{k,p} \vec{\epsilon}_{kp} A_{kp}(t) e^{i\vec{K} \cdot \vec{r}} + \text{c.c.}, \quad (\text{D3})$$

where  $\vec{K}$  is the wave vector that takes restricted values owing to the defined boundary conditions,  $\vec{\epsilon}$  represents the polarization vector, c.c. stands for complex conjugate and  $A_{kp}$  denotes the mode function. In this configuration, there are two conditions that must be satisfied: (a)  $\vec{K} \cdot \vec{\epsilon}_{\vec{K}} = 0$ , which arises from Coulomb gauge, implying that the orthogonal between the polarization of light and its direction of propagation; and (b)  $\vec{\epsilon}_{kp} \cdot \vec{\epsilon}_{kp'} = \delta_{p,p'}$  that comes from the orthogonality of the polarization vectors. By substituting Eq. (D3) in Eq. (D2), the following solution to the wave equation can be obtained [97],

$$A_{kp}(t) = A_{kp} e^{-i\omega_k t}, \quad (\text{D4})$$

where  $\omega_k = |\vec{K}| c$  represent the frequency of the  $k$ th mode. Inserting this solution again in Eq. (D3), we get

$$\vec{A}(\vec{r}, t) = \sum_{k,p} \vec{\epsilon}_{kp} A_{kp} e^{i(\vec{K} \cdot \vec{r} - \omega_k t)} + \text{c.c.} \quad (\text{D5})$$

In the classical picture, the energy of the electromagnetic (EM) field is given by [95,96]

$$H_c = \frac{1}{2} \int_V \left( \epsilon_0 |\vec{E}|^2 + \frac{1}{\mu_0} |\vec{B}|^2 \right) dV, \quad (\text{D6})$$

where  $V = L^3$  is the mode volume. Now solving Eqs. (D1a) and (D1b) with the result of Eq. (D5) and substituting the results in Eq. (D6) we obtain [97]

$$H_c = \sum_{k,p} \epsilon_0 V \omega_k^2 \left( A_{kp} A_{kp}^* + A_{kp}^* A_{kp} \right). \quad (\text{D7})$$

Again in the quantum picture, the energy of the EM field is given by [96]

$$H_q = \sum_{k,p} \hbar \omega_k \left( \hat{n}_{kp} + \frac{1}{2} \right), \quad (\text{D8})$$

where  $\hat{n}_{kp} = \hat{a}_{kp}^\dagger \hat{a}_{kp}$  is the photon number operator. Also,  $\hat{a}_{kp}$  and its hermitian conjugate (H.c.)  $\hat{a}_{kp}^\dagger$  are the quantum-mechanical field operators that satisfy the commutation relation:  $[\hat{a}_{kp}, \hat{a}_{k'p'}^\dagger] = \delta_{kk'} \delta_{pp'}$  and  $[\hat{a}_{kp}, \hat{a}_{k'p'}] = [\hat{a}_{kp}^\dagger, \hat{a}_{k'p'}^\dagger] = 0$ .

In EM field quantization, every mode of the classical field is associated to a quantum-mechanical harmonic oscillator. Here, by comparing Eqs. (D8) and (D7), the map from the classical-to-quantum picture can be done with the following substitutions,

$$A_{k,p} = \sqrt{\frac{\hbar}{2\epsilon_0 V \omega_k}} \hat{a}_{k,p}, \quad (\text{D9a})$$

$$A_{k,p}^* = \sqrt{\frac{\hbar}{2\epsilon_0 V \omega_k}} \hat{a}_{k,p}^\dagger. \quad (\text{D9b})$$

Considering the substitution of Eqs. (D9a) and (D9b) in Eq. (D5), we obtain the quantized vector potential for the EM field, which can then be expressed as

$$\hat{A}(\vec{r}, t) = \sum_{k,p} \sqrt{\frac{\hbar}{2\epsilon_0 V \omega_k}} \left[ \vec{\epsilon}_{kp} \hat{a}_{kp} e^{i(\vec{K} \cdot \vec{r} - \omega_k t)} + \text{H.c.} \right]. \quad (\text{D10})$$

Therefore, by substituting Eq. (D10) in Eq. (D1a), we obtain the quantized electric field vector, which when polarized along the unit vector  $\vec{\epsilon}_k$  is given by

$$\hat{E}_k(\vec{r}, t) = \sum_{k,p} \sqrt{\frac{\hbar \omega_k}{2\epsilon_0 V}} \left[ \vec{\epsilon}_k \hat{a}_k e^{i(\vec{K} \cdot \vec{r} - \omega_k t)} + \text{H.c.} \right]. \quad (\text{D11})$$

Here,  $\sqrt{\hbar \omega_k / 2\epsilon_0 V}$  is the amplitude factor with  $V$  being the quantization volume. Also, we can turn the electric field vector into a positive and a negative frequency part, which can then be expressed as

$$\begin{aligned} \hat{E}_k(\vec{r}, t) &= \hat{E}_k^{(+)}(\vec{r}, t) + \hat{E}_k^{(-)}(\vec{r}, t), \quad \text{with} \\ \hat{E}_k^{(-)}(\vec{r}, t) &= \left[ \hat{E}_k^{(+)}(\vec{r}, t) \right]^\dagger. \end{aligned} \quad (\text{D12})$$

By considering Eq. (D12) and assuming that the electric field is horizontally polarized, the positive frequency part of Eq. (D11) can be rewritten as [92]

$$\hat{E}_k^{(+)}(\vec{r}, t) = \sum_k \sqrt{\frac{\hbar \omega_k}{2\epsilon_0 [n(\omega_k)]^2 V}} \mathcal{E}(\vec{K}(\omega_k), \vec{r}) \hat{a}_k e^{-i\omega_k t}, \quad (\text{D13})$$

where using the dispersion relation we substitute  $|\vec{K}(\omega_k)| = \omega_k n(\omega_k)/c$ , with  $n(\omega_k)$  being the frequency-dependent refractive index (squared in this case owing to its nonlinear dependence), and  $\mathcal{E}(\vec{K}(\omega_k), \vec{r}) = e^{i\vec{K}(\omega_k)\cdot\vec{r}}$  represents the *spatial mode function*.

The above form of the electric field vector along with the Gaussian beam approximation [87] provides us the electric field expressions for the pump ( $p$ ), signal ( $s$ ), and idler ( $i$ ) photons in a SPDC process. In the following subsections we illustrate this construction in details.

## 2. Gaussian beam approximation

We consider the solution to the paraxial wave equation (given in Ref. [96]) for the propagation of an optical wave through a nonlinear medium. We also assume that the beam undergoes transverse intensity modulation that is everywhere Gaussian. With this assumption, the solution can be then re-expressed in scalar approximation as [87]

$$A(r, z) = A_0 \frac{w_0}{w(z)} e^{-r^2/w(z)^2} e^{iK r^2/2R(z)} e^{i\Phi(z)}, \quad (\text{D14})$$

where  $K = 2\pi n\omega/c$  is the wave number with  $n$  being the complex refractive index of the nonlinear medium,  $r^2 = x^2 + y^2 + z^2$  represents the spherical beam shape with radius ( $r$ ) in a three-dimensional coordinate system,  $\lambda = 2\pi c/n\omega$  represents the wavelength of the radiation in the medium,  $w(z) = w_0 \left[1 + (\lambda z/\pi n w_0^2)^2\right]^{1/2} = w_0 \left[1 + (z/L_R)^2\right]^{1/2}$  represents the beam waist of the Gaussian beam at a distance  $z$  for  $w_0$  being the input beam waist (i.e., at  $z = 0$ ) and  $L_R = \pi n w_0^2/\lambda$  denotes the Rayleigh length,  $R(z) = z \left[1 + (\pi n w_0^2/\lambda z)^2\right] = z \left[1 + (L_R/z)^2\right]$  is the radius of curvature of the optical wavefront, and  $\Phi(z) = -\arctan(\lambda z/\pi n w_0^2) = -\arctan(z/L_R)$  represents the spatial variation of the phase of the wave (measured with respect to that of an infinite plane wave).

In principle, the output of the pump laser has a Gaussian mode profile propagating in  $z$  direction, which then undergoes SPDC in the experiment. So, the above expression for  $A(r, z)$  serves as the form of *spatial mode function*  $\mathcal{E}[\vec{K}(\omega_k), \vec{r}]$  in the expression of the electric field given by Eq. (D13). The beam undergoes nonlinear interaction through the PPKTP crystal of a certain length, say  $L$ , along its direction of propagation ( $z$ ). However, it remains unaffected along the  $x$  and  $y$  directions. So, in order to calculate the value of the wave amplitude  $A_0$  in Eq. (D14), we can consider the following normalization condition, i.e.,  $\forall z$ :

$$\iint_{-\infty}^{\infty} |A(r, z)|^2 dx dy = 1. \quad (\text{D15})$$

With this condition, we get

$$A_0 = \frac{\sqrt{2}}{\sqrt{\pi} w_0} e^{z^2/w(z)^2} \quad (\text{D16})$$

and so  $A_0$  is also a function of  $z$ . By substituting Eq. (D16) in Eq. (D14) and ignoring the Gouy phase term, we obtain the final scalar approximated version of our spatial mode function  $\mathcal{E}(\vec{K}(\omega_k), \vec{r})$ :

$$\mathcal{E}(\vec{K}(\omega_k), \vec{r}) \equiv \frac{\sqrt{2}}{\sqrt{\pi} w(z)} e^{-(x^2+y^2)/w(z)^2} e^{iK(x^2+y^2+z^2)/2R(z)}. \quad (\text{D17})$$

It is important to note that this expression for the spatial mode function is the same as that provided in Ref. [98] for the case of *focused* Gaussian pump beam.

## 3. Quantum Hamiltonian governing the SPDC process

Under periodic boundary conditions,

$$\mathcal{E}(\vec{K}(\omega_k), \vec{r})|_{z=-L/2} = \mathcal{E}(\vec{K}(\omega_k), \vec{r})|_{z=L/2} \quad \text{for } z \text{ axis}, \quad (\text{D18a})$$

$$\mathcal{E}(\vec{K}(\omega_k), \vec{r})|_{x=-\infty} = \mathcal{E}(\vec{K}(\omega_k), \vec{r})|_{x=\infty} = 0 \quad \text{for } x \text{ axis}, \quad (\text{D18b})$$

$$\mathcal{E} [\vec{K}(\omega_k), \vec{r}]|_{y=-\infty} = \mathcal{E} [\vec{K}(\omega_k), \vec{r}]|_{y=\infty} = 0 \quad \text{for } y \text{ axis,} \quad (\text{D18c})$$

we get  $e^{-ikL/2} = e^{ikL/2}$ , which leads to  $k = 2m\pi/L$  for  $m = 0, \pm 1, \pm 2, \dots$ , with  $L$  being the length of the PPKTP crystal.

Now, the total energy for all the  $k = 2m\pi/L$  modes is contained within Eq. (D8), while the average energy per mode (frequency) is given by  $\hbar\omega(\hat{n} + \frac{1}{2})$ . So, the total energy ( $E$ ) for all frequencies can be expressed as

$$\int_{\omega} \frac{dE}{d\omega} = \int_{\omega} \frac{\delta E}{\delta m} \times \frac{\delta m}{\delta k} \times \frac{\delta k}{\delta \omega} = \int_{\omega} \frac{\hbar\omega(\hat{n} + 1/2)L}{2\pi c} d\omega. \quad (\text{D19})$$

Rewriting the positive frequency part of the electric field in Eq. (D13) with frequency indices instead of mode indices, we get

$$\hat{E}_{\omega}^{(+)}(\vec{r}, t) = \int_{\omega} \mathcal{A} \mathcal{E}_{\omega}(\vec{r}) \hat{a}_{\omega} e^{-i\omega t} d\omega, \quad (\text{D20})$$

where  $\mathcal{A}$  is the transformed amplitude factor. In order to calculate the exact form of this factor, we substitute the value of the electric field operator from Eq. (D22) in Eq. (D6) and compare with the result of Eq. (D19). This gives us the form of the amplitude factor to be,

$$\mathcal{A} = \sqrt{\frac{\hbar\omega}{4\pi\epsilon_0 c}}. \quad (\text{D21})$$

Finally, substituting Eq. (D21) in Eq. (D22), the complete form of the positive frequency part of the electric field operator turns out to be

$$\hat{E}_{\omega}^{(+)}(\vec{r}, t) = \int_0^{\infty} \sqrt{\frac{\hbar\omega}{4\pi\epsilon_0 c}} \mathcal{E}_{\omega}(\vec{r}) \hat{a}_{\omega} e^{-i\omega t} d\omega, \quad (\text{D22})$$

where  $\mathcal{E}_{\omega}(\vec{r})$  denotes a Gaussian spatial mode function of the form given in Eq. (D17) and having a frequency  $\omega$  [98].

A parametric interaction between a pump field ( $p$ ) and two other fields, designated as signal ( $s$ ) and idler ( $i$ ), initially in the vacuum state defines the spontaneous parametric down-conversion process. We restrict our considerations to the electromagnetic fields which describe a linearly (i.e., horizontal in this case) polarized light with paraxial Gaussian beam waist at the origin, propagating along  $z$  direction through a nonlinear medium (PPKTP crystal of length  $L$ ) of second-order nonlinearity.

From Eq. (D22), the quantized version of the electric field vectors for the pump ( $p$ ), signal ( $s$ ) and idler ( $i$ ) photons can be expressed as

$$\hat{E}_{\omega_p}^{(+)}(\vec{r}, t) = \int_0^{\infty} \sqrt{\frac{\hbar\omega_p}{4\pi\epsilon_0 c}} \mathcal{E}_{\omega_p}(\vec{r}) \hat{a}_{\omega_p} e^{-i\omega_p t} d\omega_p \quad \text{for pump } (p) \text{ photon,} \quad (\text{D23a})$$

$$\hat{E}_{\omega_s}^{(-)}(\vec{r}, t) = \int_0^{\infty} \sqrt{\frac{\hbar\omega_s}{4\pi\epsilon_0 c}} \mathcal{E}_{\omega_s}^*(\vec{r}) \hat{a}_{\omega_s}^{\dagger} e^{i\omega_s t} d\omega_s \quad \text{for signal } (s) \text{ photon,} \quad (\text{D23b})$$

$$\hat{E}_{\omega_i}^{(-)}(\vec{r}, t) = \int_0^{\infty} \sqrt{\frac{\hbar\omega_i}{4\pi\epsilon_0 c}} \mathcal{E}_{\omega_i}^*(\vec{r}) \hat{a}_{\omega_i}^{\dagger} e^{i\omega_i t} d\omega_i \quad \text{for idler } (i) \text{ photon,} \quad (\text{D23c})$$

where  $\omega_p$ ,  $\omega_s$ , and  $\omega_i$  represent the frequency of the pump, signal, and idler photons. However, in our case, since we use a bright continuous-wave laser source for the pump beam, we replace Eq. (D23a) with its classical form [92],

$$\hat{E}_{\omega_p}^{(+)}(\vec{r}, t) \rightarrow \vec{E}_{\omega_p}^{(+)}(\vec{r}, t) \equiv \mathcal{A}_p \int_0^{\infty} s(\omega) \mathcal{E}_{\omega_p}(\vec{r}) e^{-i\omega_p t} d\omega_p, \quad (\text{D24})$$

where  $\mathcal{A}_p$  is the pump amplitude. Here for our monochromatic pump  $s(\omega) = \delta(\omega - \omega_p)$ , which represents the pump spectral amplitude. However, for a general spectral amplitude distribution the corresponding normalized power spectral density is given by  $\int_{\omega} d\omega |s(\omega)|^2 = 1$ .

Therefore, the quantum Hamiltonian for this parametric interaction is then given by

$$\hat{H}_I(\vec{r}, t) = \int_V \epsilon_0 \chi^{(2)}(\vec{r}) : \left[ \hat{E}_{\omega_p}^{(+)}(\vec{r}, t) \hat{E}_{\omega_s}^{(-)}(\vec{r}, t) \hat{E}_{\omega_i}^{(-)}(\vec{r}, t) + \text{H.c.} \right] dV, \quad (\text{D25})$$

$\chi^{(2)}(r)$  is the nonlinear susceptibility tensor.

### a. Pair production process

In the interaction picture, we apply the unitary operator  $U = \exp\left[-(i/\hbar) \int_{-\infty}^{\infty} dt \hat{H}_I(\vec{r}, t)\right]$  to the initial vacuum  $(|0\rangle)$  state of the signal and the idler photons  $|\psi_{\text{in}}(t=0)\rangle = |0\rangle_s \otimes |0\rangle_i$  to obtain the resultant state of the SPDC process. The first term of this expansion produces the output state at time  $t$ , that is the creation of a photon pair:

$$|\Psi_{\text{SPDC}}\rangle = -\frac{i}{\hbar} \int_{-\infty}^{\infty} dt \hat{H}_I(\vec{r}, t) |\psi_{\text{in}}(t=0)\rangle. \quad (\text{D26})$$

We now insert Eqs. (D23a), (D23b), (D23c), and (D25) in Eq. (D26) to obtain the final form the output state  $|\Psi_{\text{SPDC}}\rangle$ ,

$$|\Psi_{\text{SPDC}}\rangle = -i \int \int_0^{\infty} \psi(\omega_s, \omega_i) \hat{a}_{\omega_s}^{\dagger} \hat{a}_{\omega_i}^{\dagger} |0\rangle_s |0\rangle_i d\omega_s d\omega_i, \quad (\text{D27})$$

where the amplitude (joint spectral amplitude [92]) of the SPDC process is given by [98]

$$\psi(\omega_s, \omega_i) = \sqrt{\frac{2\pi^2 \hbar N_p}{\epsilon_0 \lambda_p \lambda_s \lambda_i}} s(\omega_p) \mathcal{O}(\omega_s, \omega_i). \quad (\text{D28})$$

In Eq. (D28), the mean photon number of the pump beam is denoted by  $N_p$ , where  $s(\omega_p) \sqrt{N_p}$  replaces the operator  $\hat{a}_{\omega_p}$ , and the free-space wavelength of the pump, signal, and idler photons is given by  $\lambda_h = 2\pi c / \omega_h$ , where  $\omega_p = \omega_s + \omega_i$ , for  $h \in \{p, s, i\}$ . Also,  $\mathcal{O}(\omega_s, \omega_i)$  represents the spatial overlap of the pump, signal, and idler modes in the nonlinear medium. This SPDC amplitude contains all the spatiotemporal structure of the two-photon output state [92]. Conceptually, the joint spectral amplitude is a product of pump spectral amplitude  $s(\omega_p)$  and phase-matching term, which essentially is a sinc function. The intensity of the classical fields for the signal and idler photons is proportional to the square modulus of this phase-matching function [92].

### b. Pair-generation probability density and pair-generation rate

The square modulus of the joint spectral amplitude (or in other words the joint spectral density) of the SPDC process, i.e.,  $|\psi(\omega_s, \omega_i)|^2$ , gives the pair-generation probability density. More particularly, given the pump power, it provides the expected number of photon pairs produced per unit time per signal (or idler) bandwidth.

Moreover, we know that  $\omega_p = \omega_s + \omega_i$  and  $\omega_p$  can be considered to be the coherent state from the laser source. Therefore, if we now numerically integrate  $\psi(\omega_s, \omega_i)$  [given by Eq. (D28)] over a certain bandwidth, i.e., a range of signal (or idler) frequencies, then we obtain a sinc<sup>2</sup> nature trace for this pair-generation probability density plotted over a spectrum of signal frequencies (as sketched in Fig. 24).

The maxima of this probability distribution provides the corresponding wavelength information at which the SPDC pair-generation rate will be maximal. The maximum value of pair-generation probability obtained numerically can be directly verified with the experimental data. Consequently, the pair-generation rate can be defined as [92,98,99]

$$R \propto \langle \psi | \psi \rangle = \int_0^{\infty} d\omega_s \int_0^{\infty} d\omega_i |\psi(\omega_s, \omega_i)|^2. \quad (\text{D29})$$

The pair-production rate  $R$  is proportional to the Boyd-Kleinman factor [92,98]. In the case of weakly focused Gaussian pump beam, the Boyd-Kleinman factor is proportional to the length  $L$  of the nonlinear crystal [92]. However, it is important to note that this relationship of  $R$  and  $L$  is not true for the case of strongly focused Gaussian pump beam; and also that there exists a trade-off between heralding ratio and tightness of focus [92].

## APPENDIX E: NUMERICAL CALCULATION FOR THE GENERATION RATE OF PHOTON PAIRS FROM TYPE-II SPDC PROCESS

Here we describe the numerical approach that we follow to obtain the single-photon pair generation in a type-II SPDC process using the quasi-phase-matching condition in a PPKTP crystal. Algorithm 5 shows the numerical method for calculating refractive indices along the directions of the pump, signal, and idler beams at any given temperature.

---

### Algorithm 5 Calculation of refractive indices using Sellmier equations

---

#### Require:

- FLOAT TEMP ▷ Input temperature value
- FLOAT LAMBDA\_P ▷ Pump-beam wavelength in meters
- FLOAT LAMBDA\_S ▷ Wavelength of signal photons in meters
- FLOAT LAMBDA\_I ▷ Wavelength of idler photons in meters

#### Ensure:

- FLOAT REFP ▷ Refractive index along the pump-beam direction
  - FLOAT REFS ▷ Refractive index along the direction of signal photons
  - FLOAT REFI ▷ Refractive index along the direction of pump photons
- 1: **function** REFRACTIVEINDICES(TEMP, LAMBDA\_P, LAMBDA\_S, LAMBDA\_I)
  - 2:   FLOAT NP ▷ Temperature-independent refractive index along the pump-beam direction
  - 3:   FLOAT NS ▷ Temperature-independent refractive index along the direction of signal photons
  - 4:   FLOAT NI ▷ Temperature-independent refractive index along the direction of idler photons
  - 5:   AZ ← 2.12725 ▷ Assigning values to the Sellmier coefficients
  - 6:   BZ ← 1.18431
  - 7:   CZ ← 0.0514852
  - 8:   DZ ← 0.6603
  - 9:   EZ ← 100.00507
  - 10:   FZ ← 0.00968956
  - 11:   AY ← 2.19229
  - 12:   BY ← 0.83547
  - 13:   CY ← 0.0497
  - 14:   DY ← 0.01621
  - 15:   A<sub>0</sub> ← 9.9587×10<sup>-6</sup>
  - 16:   A<sub>1</sub> ← 9.9228×10<sup>-6</sup>
  - 17:   A<sub>2</sub> ← -8.9603×10<sup>-6</sup>
  - 18:   A<sub>3</sub> ← 4.1010×10<sup>-6</sup>
  - 19:   B<sub>0</sub> ← -1.1882×10<sup>-8</sup>
  - 20:   B<sub>1</sub> ← 10.459×10<sup>-8</sup>
  - 21:   B<sub>2</sub> ← -9.8136×10<sup>-8</sup>
  - 22:   B<sub>3</sub> ← 3.1481×10<sup>-8</sup>
  - 23:   C<sub>0</sub> ← 6.2897×10<sup>-6</sup>
  - 24:   C<sub>1</sub> ← 6.3061×10<sup>-6</sup>
  - 25:   C<sub>2</sub> ← -6.0629×10<sup>-6</sup>
  - 26:   C<sub>3</sub> ← 2.6486×10<sup>-6</sup>
  - 27:   D<sub>0</sub> ← -0.14445×10<sup>-8</sup>
  - 28:   D<sub>1</sub> ← 2.2244×10<sup>-8</sup>
  - 29:   D<sub>2</sub> ← -3.5770×10<sup>-8</sup>
  - 30:   D<sub>3</sub> ← 1.3470×10<sup>-8</sup>
  - 31:   LAMBDA\_P ← LAMBDA\_P × 10<sup>6</sup> ▷ Converting wavelength in meters to microns
  - 32:   LAMBDA\_S ← LAMBDA\_S × 10<sup>6</sup>
  - 33:   LAMBDA\_I ← LAMBDA\_I × 10<sup>6</sup>
  - 34:   NP ←  $\sqrt{AY + \frac{BY}{1 - CY \times LAMBDA_P^{-2}} - DY \times LAMBDA_P^2}$  ▷ Obtaining the values for the temperature-independent refractive indices
-

---

```

35:  NS ←  $\sqrt{AY + \frac{BY}{1-CY \times LAMBDAS^{-2}} - DY \times LAMBDAS^2}$ 
36:  NI ←  $\sqrt{AZ + \frac{BZ}{1-CZ \times LAMBDAL^{-2}} - \frac{DZ}{1-EZ \times LAMBDAL^{-2}} - FZ \times LAMBDAL^2}$ 
37:  REFP ← NP + (TEMP - 25) ×  $\left(C_0 + \frac{C_1}{LAMBDA P} + \frac{C_2}{LAMBDA P^2} + \frac{C_3}{LAMBDA P^3}\right)$  + (TEMP - 25)2 ×
     $\left(D_0 + \frac{D_1}{LAMBDA P} + \frac{D_2}{LAMBDA P^2} + \frac{D_3}{LAMBDA P^3}\right)$     ▷ Obtaining the final temperature-dependent refractive indices
38:  REFS ← NS + (TEMP - 25) ×  $\left(C_0 + \frac{C_1}{LAMBDA S} + \frac{C_2}{LAMBDA S^2} + \frac{C_3}{LAMBDA S^3}\right)$  + (TEMP - 25)2 ×
     $\left(D_0 + \frac{D_1}{LAMBDA S} + \frac{D_2}{LAMBDA S^2} + \frac{D_3}{LAMBDA S^3}\right)$ 
39:  REFI ← NI + (TEMP - 25) ×  $\left(A_0 + \frac{A_1}{LAMBDA I} + \frac{A_2}{LAMBDA I^2} + \frac{A_3}{LAMBDA I^3}\right)$  + (TEMP - 25)2 ×
     $\left(B_0 + \frac{B_1}{LAMBDA I} + \frac{B_2}{LAMBDA I^2} + \frac{B_3}{LAMBDA I^3}\right)$ 
40:  return REFP, REFS, REFI    ▷ Returning the specified outputs
41: end function

```

---

Algorithm 6 shows the numerical method for calculating the optimal temperature for obtaining the quasi-phase-matching condition for type-II SPDC process of single-photon generation. The function takes into account the poling period of the crystal and the signal and pump-beam wavelengths and returns the temperature for the condition that satisfies the QPM condition corresponding to the input wavelengths. This function also uses the “refractive index” function described in Algorithm 5 to obtain the refractive indices for any given temperature.

---

#### Algorithm 6 Obtaining the optimal temperature for the QPM condition

---

##### Require:

FLOAT LAMBDA P ▷ Pump-beam wavelength  
 FLOAT LAMBDA S ▷ Wavelength of signal photons  
 FLOAT POLPERIOD ▷ Poling period of the crystal

##### Ensure:

FLOAT TEMP ▷ Optimal temperature for the QPM condition

```

1: function QPMTEMPERATURE(LAMBDA P, LAMBDA S, POLPERIOD)
2:   FLOAT LAMBDAL    ▷ Wavelength of idler photons
3:   FLOAT KP         ▷ Wavevector of the pump beam
4:   FLOAT KS         ▷ Wavevector of the signal photons
5:   FLOAT KI         ▷ Wavevector of the idler photons
6:   FLOAT KPOL      ▷ Wavevector corresponding to the poling period
7:   FLOAT TPOL      ▷ Temperature-dependent poling period of the crystal
8:   FLOAT RES       ▷ Difference in the value of the wavevectors
9:   FLOAT [3] ARRREF ▷ Temperature-dependent refractive indices' array
10:  TEMPVAR ← 20    ▷ Start value for temperature scan
11:  MIN ← 10        ▷ Minimum difference value of the wavevectors
12:  A ← 6.7 × 10-6 ▷ Length expansion coefficients
13:  B ← 11 × 10-9
14:  while TEMPVAR ≤ 100 do    ▷ Iterating over temperature range
15:    REFARR ← REFRACTIVEINDICES(TEMPVAR, LAMBDA P, LAMBDA S, LAMBDAL)    ▷ Obtaining the
    temperature-dependent refractive indices using the function defined in Algorithm 5
16:    TPOL ← POLPERIOD × (1 + A × (T - 25) + B × (T - 25)2)    ▷ Obtaining the effective poling period at the
    temperature
17:    KP ←  $\frac{2 \times \pi \times \text{REFARR}[0]}{\text{LAMBDA P}}$     ▷ Obtaining the temperature-dependent wavevectors
18:    KS ←  $\frac{2 \times \pi \times \text{REFARR}[1]}{\text{LAMBDA S}}$ 

```

---

---

```

19:      KI ←  $\frac{2 \times \pi \times \text{REFARR}[2]}{\text{LAMBDAI}}$ 
20:      KPOL ←  $\frac{2 \times \pi}{\text{TPOL}}$ 
21:      RES ← KP − (KS + KI + KPOL)           ▷ Obtaining the wave vector mismatch
22:      if | RES | ≤ MIN then           ▷ Checking whether the wave vector mismatch is below the set threshold value
23:          MIN ← | RES |
24:          TEMP ← TEMPVAR                   ▷ Obtaining the optimal temperature for the QPM condition
25:      end if
26:      TEMPVAR ← TEMPVAR + 0.1             ▷ Incrementing the iterative temperature value for condition check
27:  end while
28:  return TEMP                           ▷ Returning the specified output
29: end function

```

---

Algorithm 7 shows the numerical approach for calculating the spatial mode overlap of the signal, idler, and the pump modes generated within the dimensions of the crystal.

---

### Algorithm 7 Spatial mode-overlap calculation

---

#### Require:

```

FLOAT TEMP           ▷ Input temperature value
FLOAT LAMBDA_P      ▷ Pump-beam wavelength
FLOAT LAMBDA_S      ▷ Wavelength of signal photons
FLOAT LAMBDA_I      ▷ Wavelength of idler photons
FLOAT SIZE_P        ▷ Spot size of the pump beam at the crystal center
FLOAT SIZE_S        ▷ Spot size of the signal photons at the crystal center
FLOAT SIZE_I        ▷ Spot size of the idler photons at the crystal center
FLOAT POLPERIOD     ▷ Poling period of the crystal
FLOAT LENCRYSTAL    ▷ Length of the crystal

```

#### Ensure:

```

FLOAT RES           ▷ Spatial mode-overlap value

```

```

1: function MODEOVERLAP(TEMP, LAMBDA_P, LAMBDA_S, LAMBDA_I, SIZE_P,
   SIZES, SIZE_I, POLPERIOD, LENCRYSTAL)
2:   FLOAT [3] ARRREF           ▷ Temperature-dependent refractive indices' array
3:   FLOAT RADP                 ▷ Beam radius of the pump beam at a given distance
4:   FLOAT RADS                 ▷ Beam radius of the signal photons at a given distance
5:   FLOAT RADI                 ▷ Beam radius of the idler photons at a given distance
6:   FLOAT KP                   ▷ Wavevector of the pump beam
7:   FLOAT KS                   ▷ Wavevector of the signal photons
8:   FLOAT KI                   ▷ Wavevector of the idler photons
9:   FLOAT QP                   ▷ Calculated quantity for the pump beam
10:  FLOAT QS                   ▷ Calculated quantity for the signal photons
11:  FLOAT QI                   ▷ Calculated quantity for the idler photons
12:  FLOAT Z                     ▷ Distance from the center of the crystal
13:  FLOAT J                     ▷ Relates the iteration step size to the poling period of the crystal
14:  SIGNED INTEGER F           ▷ Sign value for the integration
15:  ZSTEP ← 10-6             ▷ Step size for iteration
16:  I ← 0                       ▷ Initializing iterative variable
17:  REFARR ← REFRACTIVEINDICES (TEMP, LAMBDA_P, LAMBDA_S, LAMBDA_I) ▷ Obtaining the temperature-dependent
   refractive indices using the function defined in Algorithm 5
18:  J ←  $\frac{\text{POLPERIOD}}{2 \times \text{ZSTEP}}$ 

```

---



---

```

19:   RES ← 0                                     ▷ Initializing the mode-overlap value
20:   Z ←  $\frac{-\text{LENCRYSTAL}}{2}$                                ▷ Initializing iterative distance
21:   while Z ≤  $\frac{\text{LENCRYSTAL}}{2}$  do                               ▷ Iterating over the length of the crystal
22:     RADP ← SIZEP ×  $\sqrt{1 + \frac{Z \times \text{LAMBDA P}}{(\pi \times \text{REFARR}[0] \times \text{SIZEP}^2)^2}}$  ▷ Calculating the beam radius at the given distance from the
    center of the crystal
23:     RADS ← SIZES ×  $\sqrt{1 + \frac{Z \times \text{LAMBDA S}}{(\pi \times \text{REFARR}[1] \times \text{SIZES}^2)^2}}$ 
24:     RADI ← SIZEI ×  $\sqrt{1 + \frac{Z \times \text{LAMBDA I}}{(\pi \times \text{REFARR}[2] \times \text{SIZEI}^2)^2}}$ 
25:     KP ←  $\frac{2 \times \pi \times \text{REFARR}[0]}{\text{LAMBDA P}}$                                ▷ Calculating the wavevectors of the pump, signal, and idler beam
26:     KS ←  $\frac{2 \times \pi \times \text{REFARR}[1]}{\text{LAMBDA S}}$ 
27:     KI ←  $\frac{2 \times \pi \times \text{REFARR}[2]}{\text{LAMBDA I}}$ 
28:     QP ←  $\text{RADP}^2 + \frac{2i \times Z}{\text{KP}}$                                ▷ Calculating the intermediate variables
29:     QS ←  $\text{RADS}^2 + \frac{2i \times Z}{\text{KS}}$ 
30:     QI ←  $\text{RADI}^2 + \frac{2i \times Z}{\text{KI}}$ 
31:     I ← I + 1                                     ▷ Incrementing the iterative variable with each run of the loop
32:     if INT( $\frac{I}{J}$ ) mod 2 ≠ 0 then                               ▷ Conditioning over the ratio of the number of iteration and the derived
    quantity from the poling period of the crystal and step size of the iteration
33:       F ← 1                                       ▷ Assigning case-dependent value to the variable
34:     else
35:       F ← -1
36:     end if
37:     RES ← RES +  $\left( \frac{2^{\frac{3}{2}}}{\sqrt{\pi}} \times \frac{\text{RADP} \times \text{RADS} \times \text{RADI}}{\text{QP} \times \text{QS} + \text{QI} \times \text{QS} + \text{QP} \times \text{QI}} \times F \times \exp(i \times (\text{KP} - \text{KS} - \text{KI}) \times Z) \times \text{ZSTEP} \right)$  ▷ Obtaining the
    mode-overlap value
38:     Z ← Z + ZSTEP                               ▷ Incrementing the distance
39:   end while
40:   return RES                                     ▷ Returning the specified output
41: end function

```

---

Using the previous algorithms for the functions “refractive indices,” “QPM temperature,” and “mode overlap,” Algorithm 8 is employed to calculate the single-photon pair-generation rate. The module takes as input the intensity of the pump beam, the degeneracy condition, beam waist of the pump beam at the center of the crystal, the poling period and nonlinear coefficient of the crystal, and the total time for which the pump is switched on. In output, it returns the total number of single-photon pairs generated.

---

#### Algorithm 8 Obtaining the pair-generation rate from type-II SPDC process

---

##### Require:

```

FLOAT PUMPINT                                     ▷ Intensity of the pump beam
FLOAT LAMBDA P                                   ▷ Pump-beam wavelength
FLOAT LAMBDA S                                   ▷ Wavelength of signal photons
FLOAT SIZEP                                     ▷ Spot size of the pump beam at the crystal center
FLOAT POLPERIOD                                  ▷ Poling period of the crystal
FLOAT LENCRYSTAL                                ▷ Length of the crystal
FLOAT EFF                                         ▷ Effective nonlinear coefficient of the crystal for type-II SPDC process
FLOAT LAMBDA I                                   ▷ Initial value of the signal wavelength for the iteration

```

---

---

FLOAT LAMBDAFIN FLOAT TIME <b>Ensure:</b> FLOAT RATE	▷ Final value of the signal wavelength ▷ Runtime of the process ▷ Single-photon pair-generation rate
---	--

---

1: <b>procedure</b> RIOD, LENCRYSTAL, EFF, LAMBDAIN, LAMBDAP, LAMBDAS, SIZEP, PAIRRATECALCULATION(PUMPINT, LAMBDAP, LAMBDAS, SIZEP, POLPE-	RIOD, LENCRYSTAL, EFF, LAMBDAIN, LAMBDAP, LAMBDAS, SIZEP, POLPE-
2: FLOAT SIZES 3: FLOAT SIZEI 4: FLOAT TEMP 5: FLOAT LAMBDAI 6: FLOAT [3] ARRREF 7: FLOAT OVERLAP 8: FLOAT TPOL 9: FLOAT LAMBDAITER 10: FLOAT RES 11: FLOAT S 12: FLOAT ENERGY 13: UNSIGNED INTEGER NUMPUMP 14: $EPS \leftarrow 8.85 \times 10^{-12}$ 15: $HBAR \leftarrow \frac{6.626 \times 10^{-34}}{2\pi}$ 16: $VEL \leftarrow 3 \times 10^8$ 17: $STEP \leftarrow 10^{-11}$ 18: $A \leftarrow 6.7 \times 10^{-6}$ 19: $B \leftarrow 11 \times 10^{-9}$ 20: $S \leftarrow 0$ 21: $RES \leftarrow 0$ 22: $SIZES \leftarrow \sqrt{2} \times SIZEP$ 23: $SIZEI \leftarrow \sqrt{2} \times SIZEP$ 24: $TEMP \leftarrow QPMTEMPERATURE(LAMBDAP, LAMBDAS, POLPERIOD)$ 25: $TPOL \leftarrow POLPERIOD \times (1 + A \times (T - 25) + B \times (T - 25)^2)$ 26: LAMBDAITER $\leftarrow$ LAMBDAIN 27: <b>while</b> LAMBDAITER $\leq$ LAMBDAFIN <b>do</b> 28: LAMBDAI $\leftarrow \frac{LAMBDAITER \times LAMBDAP}{LAMBDAITER - LAMBDAP}$ 29: REFARR $\leftarrow$ REFRACTIVEINDICES(TEMP, LAMBDAP, LAMBDAITER, LAMBDAI) 30: OVERLAP $\leftarrow$ MODEOVERLAP(TEMP, LAMBDAP, LAMBDAITER, LAMBDAI, SIZEP, SIZES, SIZEI, TPOL, LENCRYSTAL ) 31: $S \leftarrow 2 \times EPS \times OVERLAP \times \sqrt{\frac{2\pi^2 \times HBAR}{EPS \times LAMBDAP \times LAMBDAITER \times LAMBDAI}}$ 32: $RES \leftarrow RES +  S ^2 \times STEP \times \frac{2\pi \times VEL}{LAMBDAP^2}$ 33: <b>end while</b> 34: $ENERGY \leftarrow \frac{2\pi \times HBAR \times VEL}{LAMBDAP}$ 35: NUMPUMP $\leftarrow$ INT $\left( \frac{PUMPINT}{ENERGY} \right)$ 36: RATE $\leftarrow$ RES $\times$ NUMPUMP 37: <b>return</b> RATE 38: <b>end procedure</b>	▷ Spot size of the signal photons at the crystal center ▷ Spot size of the idler photons at the crystal center ▷ Optimal temperature for QPM condition ▷ Wavelength of idler photons ▷ Temperature-dependent refractive indices' array ▷ Mode-overlap value ▷ Temperature-dependent poling period of the crystal ▷ Iterative wavelength of the signal photons ▷ Integration value for each iteration ▷ Stores intermediate values for integration at each iteration ▷ Energy of a single pump photon ▷ Number of pump photons ▷ Free-space permittivity ▷ Planck's constant ▷ Velocity of light in free space ▷ Step size of the iteration ▷ Length expansion coefficients ▷ Initializing the intermediate integration value ▷ Initializing the resultant value for each run of the integration ▷ Calculating spot size of signal photons ▷ Calculating spot size of idler photons ▷ Poling period of the crystal at the optimal temperature for QPM ▷ Initializing the iterative variable for wavelength ▷ Iterating over the range of signal wavelength ▷ Updating the value of the idler wavelength ▷ Obtaining the temperature-dependent refractive indices using the function defined in Algorithm 5 ▷ Obtaining the mode overlap using the function defined in Algorithm 7 ▷ Calculating the intermediate value ▷ Obtaining the resultant value for each iteration ▷ Calculating the total energy of the pump beam ▷ Obtaining the number of photons in the pump beam ▷ Obtaining the pair-generation rate at the crystal ▷ Returning the specified output

---

Refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

## APPENDIX F: SINGLE-PHOTON TIME STAMPING

### 1. Methodology

We use Algorithm 9 to generate time-stamping data for single photons emitted from an ideal single-photon source.

---

**Algorithm 9** Generates the time-stamping data of the single photons emitted from a single-photon source

---

#### Require:

UNIT INTERVAL PROBSINPHOT ▷ Probability of assigning a single-photon event to an empty bin  
 UNIT INTERVAL PROBMULPHOT ▷ Probability of assigning a photon event to a bin conditioned that another  
photon is present  
 UNSIGNED INTEGER TOTBINS ▷ Total number of bins  
 REAL SIGMA ▷ Standard deviation of the Gaussian pulse for each photon

#### Ensure:

UNSIGNED INTEGER LENTIMESTAMPS ▷ Length of the array to store time-stamping data  
 UNSIGNED INTEGER [LENTIMESTAMPS] TIMESTAMPS ▷ Array to store the time-stamping data

1: **procedure** TIMESTAMPS(PROBSINPHOT, PROBMULPHOT, TOTBINS, SIGMA)  
 2:   REAL FACTOR  $\leftarrow 1 - \frac{\text{PROBMULPHOT}}{\text{PROBSINPHOT}}$   
 3:   REAL [TOTBINS] PROBARRAY ▷ Array of TOTBINS size to store the assignment probabilities  
 4:   UNSIGNED INTEGER [] TIMESTAMPS ▷ Dynamic array for storing the time stamps  
 5:   UNSIGNED INTEGER LENTIMESTAMPS ▷ Counter for the number of stored time stamps  
 6:   **for** I = 1, TOTBINS **do** ▷ Iterates over the total number of bins  
 7:     PROBARRAY [I]  $\leftarrow$  PROBSINPHOT ▷ Initializes the array with assignment probability of single-photon events  
 8:   **end for**  
 9:   TIMESTAMPS  $\leftarrow$   $\emptyset$  ▷ Preinitializes the array as a NULL set  
 10:   **while** I  $\leq$  TOTBINS **do** ▷ Iterates over the total number of bins  
 11:     UNIT INTERVAL RANDNUM  $\leftarrow$  RANDOM (0, 1) ▷ Assigns a random real number between 0 & 1  
 12:     **if** RANDNUM  $\leq$  PROBARRAY [I] **then**  
 13:       TIMESTAMPS  $\leftarrow$  TIMESTAMPS  $\cup$  I ▷ Appends the corresponding time stamp to the output array  
 14:       LENTIMESTAMPS  $\leftarrow$  LENTIMESTAMPS + 1 ▷ Increments the counter to account for the appended time  
stamp  
 15:       **for** J = 0, TOTBINS - I **do** ▷ Iterates over a subset of the total number of bins  
 16:         PROBARRAY [I + J]  $\leftarrow$  PROBSINPHOT  $\times$   $\left(1 - \text{FACTOR} \times \exp\left(\frac{-J^2}{2 \text{SIGMA}^2}\right)\right)$  ▷ Updates the probability  
after assignment of the time stamp for each photon  
 17:       **end for**  
 18:       I = I - 1 ▷ Updates the iterator for estimating the chances of reassignment  
 19:     **end if**  
 20:     I = I + 1 ▷ Increments the iterator to search for the next assignment  
 21:   **end while**  
 22:   **return** LENTIMESTAMPS, TIMESTAMPS ▷ Returns the specified outputs  
 23: **end procedure**

---

Refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

### 2. Results

Thereafter, we plot the distribution for pairwise time interval, where the  $X$  axis is the time difference between any two consecutive photons ( $t_{n+1} - t_n$ ) and  $Y$  axis represents the number of such events per second. This distribution possesses an antibunching property at smaller time scales ( $\Delta t \rightarrow 0$ ) (see Fig. 30) and an exponential decay nature at larger time scales (far from zero time interval,  $\Delta t \gg 0$ ) as depicted in Fig. 31.

This behavior (exponential decay) is also noticed in our experimental data as shown in Fig. 32. We cannot observe the antibunching behavior at shorter time scale as the dead time (45 ns) of our detector is much larger than the coherence time.

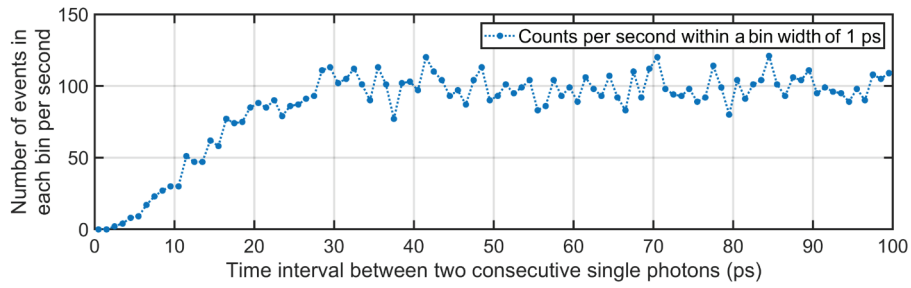


FIG. 30. Simulation: at high resolution (i.e., narrow bin width of 1 ps), we observe the antibunching behavior of a single-photon distribution. The exponential decay rate being low is invisible at very short (highly resolved) time periods. More particularly, at zero time difference no events occurred since the multiphoton probability is considered to be zero. Here, the standard deviation ( $\sigma$ ) is arbitrarily taken to be 10 ps and so the curve owing to the number of events saturates after  $3\sigma$  of pairwise time difference.

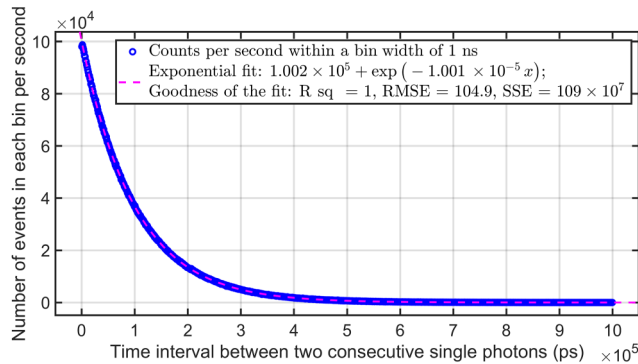


FIG. 31. Simulation: at low resolution (i.e., broader bin width of 1 ns), we observe the exponential decay nature, which is expected from any random distribution. Here, the abbreviations RMSE, SSE, and R sq. stand for root-mean-square error, sum of square error, and R squared, respectively. They determine the relative and absolute goodness of the fit.

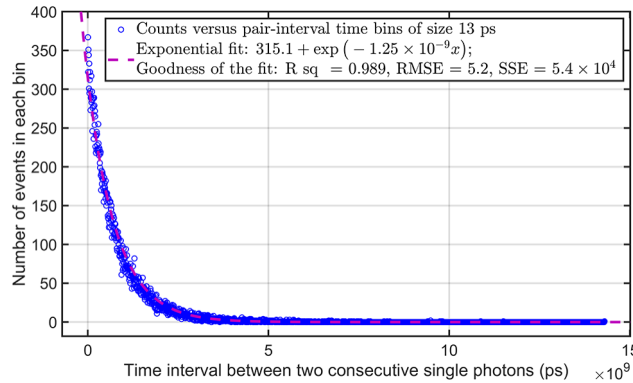


FIG. 32. Measurement: considering a binsize of 13 ps we observe the exponential decay nature for the distribution of frequency of single-photon events versus pairwise time interval between two consecutive photons emitted from the SPDC source along the signal (or idler) path. The measurements are collected for time window of 2 s and a pump power of 2 mW is used. The R-squared value of the exponential fit in pink is approximately 1 ensuring a nice fit to the measured data points.

### APPENDIX G: SIMULATION OF THE INCIDENT BACKGROUND PHOTON RATE AND GENERATION OF TIME STAMPS

In Sec. VI we explain in detail the technique that is used to infer the incident background rate from the noise level at the coincidence plots obtained experimentally. Algorithm 10 explains the numerical method.

---

**Algorithm 10** Generates the incident background rate at the detector

---

**Require:**

UNSIGNED INTEGER LENSIGNALARR ▷ Length of the array for storing arbitrary signal generation rate from source  
 UNSIGNED INTEGER [LENSIGNALARR]SIGNALARR ▷ Array for storing arbitrary signal generation rate from source  
 UNSIGNED INTEGER LENBGCOINARR1 ▷ Length of the array for storing simulated background coincidence rate for varied signal rate at a fixed background rate  
 UNSIGNED INTEGER [LENBGCOINARR1]BGCOINARR1 ▷ Array for storing simulated background coincidence rate for varied signal rate at a fixed background rate  
 UNSIGNED INTEGER LENBGRATEARR ▷ Length of the array for storing arbitrary incident background rate at the detector  
 UNSIGNED INTEGER [LENBGRATEARR]BGRATEARR ▷ Array for storing arbitrary incident background rate at the detector  
 UNSIGNED INTEGER LENBGCOINARR2 ▷ Length of the array for storing simulated background coincidence rate for varied incidence background rate for a fixed signal rate  
 UNSIGNED INTEGER [LENBGCOINARR2]BGCOINARR2 ▷ Length of the array for storing simulated background coincidence rate for varied incidence background rate for a fixed signal rate  
 UNSIGNED INTEGER DETBGCOINRATE ▷ Specific detected background coincidence rate at the TCSPCM given as input by the user  
 UNSIGNED INTEGER SIGRATE ▷ Specific signal generation rate obtained from the source  
 UNSIGNED INTEGER GENRATE ▷ Specific signal rate for which the interpolation dataset is generated

**Ensure:**

UNSIGNED INTEGER INBGRATE ▷ Incident background rate at the detector obtained by the two-step interpolation

- 1: **procedure** BACKGROUNDRATE(SIGNALARR, BGCOINARR1, BGRATEARR, BGCOINARR2, DETBGCOINRATE, SIGRATE, GENRATE)
- 2:    FLOAT ratio
- 3:     $F \leftarrow \text{INTERPOLATE}(\text{SIGNALARR}, \text{BGCOINARR1})$  ▷ Interpolated function generated from the arrays of arbitrary signal rate and the corresponding simulated background coincidence rate
- 4:     $\text{RATIO} \leftarrow \frac{F(\text{SIGRATE})}{F(\text{GENRATE})}$  ▷ Obtaining the ratio for arbitrary signal rate
- 5:     $F_1 \leftarrow \text{INTERPOLATE}(\text{BGRATEARR}, \text{BGCOINARR2})$  ▷ Interpolated function generated from the arrays of arbitrary incident background rate and the corresponding simulated background coincidence rate at a specific signal rate
- 6:     $\text{INBGRATE} \leftarrow \frac{F_1(\text{DETBGCOINRATE})}{\text{RATIO}}$  ▷ Incident background rate for an arbitrary signal rate calculated from the ratio and interpolated incident background rate for the specific signal rate
- 7:    **return** INBGRATE ▷ Returning the specified output
- 8: **end procedure**

---

Following the discussion in Sec. VI the obtained incident background rate is then used as an input to Algorithm 11 to obtain the time-stamp list of the background photons incident at the detector.

---

**Algorithm 11** Generates the time-stamping data of the background photons incident at the detection module

---

**Require:**

UNSIGNED INTEGER INBGRATE ▷ Incident background photons per second  
 UNSIGNED INTEGER TOTTIME ▷ Total time for the run of the protocol in seconds  
 UNSIGNED INTEGER STDEV ▷ Standard deviation of the Gaussian pulse for each photon in ps

**Ensure:**

UNSIGNED INTEGER LENBGTIMESTAMPS ▷ Length of the background photon time-stamp array  
 UNSIGNED INTEGER [LENBGTIMESTAMPS]BGTIMESTAMPS ▷ Background photon time-stamp array

---

---

```

1: procedure TIMESTAMPS( INBGRATE, TOTTIME, STDEV)
2:   UNIT INTERVAL  PROB1           ▷ Probability of assigning a single-photon event to an empty bin
3:   INTERVAL(0, PROB1) PROB2       ▷ Random variable defined in an interval
4:   UNIT INTERVAL  FACTOR           ▷ Variable calculated from PROB1 and PROB2
5:   UNSIGNED INTEGER BINS           ▷ Total number of bins
6:   DYNAMIC PROBARR                 ▷ Declaring the array, which will contain the different probability values for all its
  elements to be a dynamic array
7:   DYNAMIC BGTIMESTAMPS           ▷ Declaring the array, which will contain the output time stamps to be a dynamic
  array
8:   UNIT INTERVAL  RAND             ▷ Random variable, which can take values from 0 to 1
9:   UNSIGNED INTEGER I               ▷ Declaring an iterative variable
10:  UNSIGNED INTEGER J               ▷ Declaring an iterative variable
11:  BINS ← 1012 × TOTTIME           ▷ Obtaining the total number of time bins in ps resolution
12:  PROB1 ←  $\frac{\text{INBGRATE}}{10^{12}}$        ▷ Obtaining the probability of a single-photon event at each ps time interval
13:  PROB2 ← PROB12                 ▷ Obtaining the probability of assigning a photon event to a bin conditioned that another
  photon is already present at each ps time interval
14:  FACTOR ←  $1 - \frac{\text{PROB2}}{\text{PROB1}}$ 
15:  PROBARR ← ∅                       ▷ Probability array preinitialized to a null set
16:  I ← 1                             ▷ Initializing the iterative variable
17:  while I ≤ BINS do                 ▷ Looping over the total number of bins taken as input
18:    PROBARR ← PROBARR ∪ PROB1       ▷ Probability array is being appended with the probability PROB1 of
  assigning a single-photon event to each of the BINS
19:    I ← I + 1                       ▷ Updates the iterator to append the next element to the array
20:  end while
21:  BGTIMESTAMPS ← ∅                 ▷ Initializing the dynamic background time-stamp array to be null set
22:  I ← 1                             ▷ Reinitializing the iterator
23:  while I ≤ BINS do                 ▷ Looping over the total number of bins taken as input
24:    RAND ← RANDOM(0, 1)             ▷ Assigns an uniform random number between 0 & 1
25:    if RAND ≤ PROBARR [I] then     ▷ Randomly selects the instance at which a single photon is generated
26:      BGTIMESTAMPS ← BGTIMESTAMPS ∪ {I}   ▷ Update the output array with the time stamp
27:      for J = 0, BINS - I do       ▷ Iterating over the remaining bins after each single-photon time-stamp
  assignment at the Ith bin
28:        PROBARR [I + J] = PROB1  $\left(1 - \text{FACTOR} \times \exp\left(\frac{-J^2}{2 \times \text{STDEV}^2}\right)\right)$    ▷ Update the probability after
  assignment of the time stamp for each photon
29:      end for
30:      I ← I - 1                     ▷ Update the iterator for estimating the chance of reassignment
31:    end if
32:    I ← I + 1                       ▷ Increment the iterator to search for the next assignment
33:  end while
34:  LENBGTIMESTAMPS ← LENGTH(BGTIMESTAMPS)   ▷ Assigns the length of the dynamic array
35:  return BGTIMESTAMPS, LENBGTIMESTAMPS   ▷ Returning the specified output
36: end procedure

```

---

Refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

## APPENDIX H: SAMPLING OF TIME STAMPS

In order to save computational time, we employ a sampling technique to generate time-stamp lists quickly. This is used when the simulation is rerun multiple times as explained in Sec. VI. Algorithm 12 describes the sampling technique that we implement.

**Algorithm 12** Sampling of time-stamp data**Require:**

UNSIGNED INTEGER LENINPARR ▷ Length of the input time-stamp array  
 UNSIGNED INTEGER [LENINPARR]INPARR ▷ Input time-stamp array  
 UNSIGNED INTEGER TOTTIME ▷ Total time for which the input time stamps is generated in ps resolution

**Ensure:**

UNSIGNED INTEGER LENSAMPARR ▷ Length of the sampled time-stamp array  
 UNSIGNED INTEGER [LENSAMPARR]SAMPARR ▷ Sampled time-stamp array

```

1: procedure SAMPLINGTIMESTAMPS([LENINPARR]INPARR)
2:   DYNAMIC INTARR ▷ Declaring the time-stamp interval array to be dynamic
3:   UNSIGNED INTEGER LENINTARR ▷ Length of the time-stamp interval array
4:   DYNAMIC SAMPARR ▷ Declaring the sampled time-stamp array to be dynamic
5:   UNSIGNED INTEGER I ▷ Declaring an iterative variable
6:   UNSIGNED INTEGER SUM ▷ Variable for storing the iterative value of the time over which sampled time stamps are generated
7:   UNSIGNED INTEGER RAND ▷ Declaring a random variable
8:   INTARR ← ∅ ▷ Initializing the dynamic time-stamp interval array to be a null set
9:   SAMPARR ← ∅ ▷ Initializing the dynamic sampled time-stamp array to be a null set
10:  I ← 1 ▷ Initializing the iterative variable
11:  while I ≤ (LENINPARR - 1) do ▷ Iterating over the elements of the input time-stamp array
12:    INTARR ← INTARR ∪ (INPARR [I + 1] - INPARR [I]) ▷ Generating the time-stamp interval array
13:  end while
14:  SUM ← 0 ▷ Initializing the iterative variable
15:  while SUM ≤ TOTTIME do ▷ Iterating until the value of time reaches the set threshold
16:    RAND ← RANDOM (1, LENINTARR) ▷ Randomly choosing a location of the time-interval array
17:    SUM ← SUM + RAND ▷ Updating the iterative value for the time
18:    SAMPARR ← SAMPARR ∪ SUM ▷ Generating the sampled time-stamp array
19:  end while
20:  LENSAMPARR ← LENGTH (SAMPARR) ▷ Calculating the length of the sampled time-stamp array
21:  return LENSAMPARR, SAMPARR ▷ Returning the specified outputs
22: end procedure

```

Refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

**APPENDIX I: SINGLE-PHOTON DETECTION USING SINGLE-PHOTON DETECTORS AND TCSPCM**

Simulation of detection of single photons forms one of the most important aspects of the simulation of an implementation of a QKD protocol. We model the single-photon detectors and TCSPCM that forms the detection module of the setup. In Sec. VII, we explain in detail the simulation technique for the various parameters of these detection components. The following algorithms provide insight into the simulation of the single-photon detection.

Algorithm 13 provides a general algorithm for processing the time stamps of the input signal photons at a detection instrument (single-photon detector and TCSPCM).

**Algorithm 13** Generates the processed time stamps corresponding to the input single-photon time stamps at the instrument**Require:**

UNIT INTERVAL EFF ▷ Efficiency of the instrument  
 UNSIGNED INTEGER DEADTIME ▷ Dead time of the instrument in ps resolution  
 SIGNED INTEGER STDEV ▷ Standard deviation of the Gaussian distribution corresponding to the timing jitter  
 SIGNED INTEGER MEAN ▷ Mean of the Gaussian distribution corresponding to the timing jitter

---

```

UNUNSIGNED INTEGER LENIPTIMESTAMPS                                ▷ Length of the array of the input photon time stamps
UNUNSIGNED INTEGER [LENIPTIMESTAMPS]IPTIMESTAMPS                ▷ Input photon time-stamp array
Ensure:
UNUNSIGNED INTEGER LENOPTIMESTAMPS                               ▷ Length of the processed photon time-stamp array
UNUNSIGNED INTEGER [LENOPTIMESTAMPS]OPTIMESTAMPS                ▷ Processed photon time-stamp array

1: function GENERALTAGGER(EFF, DEADTIME, STDEV, MEAN, LENIPTIMESTAMPS, IPTIMESTAMPS)
2:   UNIT INTERVAL RAND                                           ▷ Random variable that can take values from 0 to 1
3:   UNSIGNED INTEGER DETTIMESTAMP                                ▷ Iterative variable to store time stamps of the detected photons
4:   DYNAMIC OPTIMESTAMPS                                         ▷ Declaring the processed time-stamp array as dynamic
5:   UNSIGNED INTEGER I                                           ▷ Declaring an iterative variable
6:   REAL NORMRAND                                               ▷ Declaring a random variable
7:   SIGNED INTEGER JITTER                                         ▷ Timing jitter associated to detected time stamps
8:   OPTIMESTAMPS ← ∅                                             ▷ Initializing the dynamic processed time-stamp array to be a null set
9:   DETTIMESTAMP ← 0                                             ▷ Initializing the iterative variable
10:  I ← 1                                                         ▷ Initializing the iterative variable
11:  while I ≤ LENIPTIMESTAMPS do                                ▷ Looping over the elements of the input time-stamp array
12:    if IPTIMESTAMPS [I] − DETTIMESTAMP ≥ DEADTIME then       ▷ Rejects events that arrives at the instrument
    within its dead time
13:      RAND ← RANDOM (0, 1)                                       ▷ Assigns an uniform random number between 0 & 1
14:      if RAND ≤ EFF then ▷ Randomly selects the input time stamps based on the efficiency of the instrument
15:        NORMRAND ← NORMALRANDOM (MEAN, STDEV)                    ▷ Obtaining the effective timing-jitter value
16:        JITTER ← ROUND (NORMRAND)                               ▷ Rounding off the timing-jitter value
17:        OPTIMESTAMPS ← OPTIMESTAMPS ∪ (IPTIMESTAMPS [I] + JITTER) ▷ Generating the output
    time-stamp array
18:        DETTIMESTAMP ← IPTIMESTAMPS [I] ▷ Value of the iterative variable updated with the processed time
    stamp
19:      end if
20:    end if
21:  end while
22:  LENOPTIMESTAMPS ← LENGTH(OPTIMESTAMPS)                       ▷ Assigns the length of the dynamic array
23:  return OPTIMESTAMPS, LENOPTIMESTAMPS                         ▷ Returning the specified output
24: end function

```

---

Algorithm 14 uses the function “general tagger” to simulate the detection of photons in single-photon detectors. The module takes as input the time stamps of the received signal photons at the detectors and the dead time, efficiency, and the timing jitter of the detector. Time stamps of the TTL pulses generated in correspondence to a detection event are returned as output.

---

**Algorithm 14** Generates the time stamps of the TTL pulses corresponding to the detected single photons at the detector

---

**Require:**

```

UNIT INTERVAL EFF                                               ▷ Quantum efficiency of the detector
UNUNSIGNED INTEGER DEADTIME                                     ▷ Dead time of the detector in ps resolution
UNUNSIGNED INTEGER STDEV   ▷ Standard deviation of the Gaussian distribution corresponding to the timing jitter
UNUNSIGNED INTEGER MEAN   ▷ Mean of the Gaussian distribution corresponding to the timing jitter
UNUNSIGNED INTEGER LENIPTIMESTAMPS                               ▷ Length of the input photon time-stamp array
UNUNSIGNED INTEGER [LENIPTIMESTAMPS]IPTIMESTAMPS                ▷ Array of the incoming photon time stamps

```

**Ensure:**

```

UNUNSIGNED INTEGER LENOPTIMESTAMPS                               ▷ Length of the output array for storing TTL pulse time stamps
UNUNSIGNED INTEGER [LENOPTIMESTAMPS]OPTIMESTAMPS                ▷ TTL pulse time-stamp array

```

---



- 
- 1: **procedure** DETECTOR(EFF, DEADTIME, STDEV, MEAN, LENIPTIMESTAMPS, IPTIMESTAMPS)
  - 2:     **return** GENERALTAGGER(EFF, DEADTIME, STDEV, MEAN, LENIPTIMESTAMPS, IPTIMESTAMPS) ▷ Returning the specified output through the function defined in Algorithm 13
  - 3: **end procedure**
- 

Algorithm 15 uses the function general tagger to simulate the time stamping of detected signal photons at the TCSPCM. The module takes as input the time stamps of the received TTL pulses at the TCSPCM, loss in the connecting SMA cables, the dead time, and timing jitter of the TCSPCM. Time stamps corresponding to the detected TTL pulses are returned as output. These time stamps are considered as the time stamps of the detected photons.

---

**Algorithm 15** Generates the time stamps of the TTL pulses corresponding to the detected single photons at the detector

---

**Require:**

- REAL LOSS ▷ Loss in the SMA connector cables
- UNSIGNED INTEGER DEADTIME ▷ Dead time of the TCSPCM in ps resolution
- UNSIGNED INTEGER STDEV ▷ Standard deviation of the Gaussian distribution corresponding to the timing jitter
- UNSIGNED INTEGER MEAN ▷ Mean of the Gaussian distribution corresponding to the timing jitter
- UNSIGNED INTEGER LENIPTIMESTAMPS ▷ Length of the input TTL pulse time-stamp array
- UNSIGNED INTEGER [LENIPTIMESTAMPS]IPTIMESTAMPS ▷ Input TTL pulse time-stamp array

**Ensure:**

- UNSIGNED INTEGER LENOPTIMESTAMPS ▷ Length of the output array of the time stamps for the TTL pulses corresponding to the detected photons
- UNSIGNED INTEGER [LENOPTIMESTAMPS]OPTIMESTAMPS ▷ Output array of the time stamps for the TTL pulses corresponding to the detected photons

- 1: **procedure** TCSPCM(LOSS, DEADTIME, STDEV, MEAN, LENIPTIMESTAMPS, IPTIMESTAMPS)
  - 2:     UNIT INTERVAL EFF ▷ Efficiency of the channel connecting the TCSPCM and the detector
  - 3:      $EFF \leftarrow 10^{-\frac{LOSS}{10}}$  ▷ Efficiency of the channel calculated
  - 4:     **return** GENERALTAGGER(EFF, DEADTIME, STDEV, MEAN, LENIPTIMESTAMPS, IPTIMESTAMPS) ▷ Returning the specified output through the function defined in Algorithm 13
  - 5: **end procedure**
- 

Refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

## APPENDIX J: GENERATION OF ELECTRIC FIELD DISTRIBUTION AND BEAM PROPAGATION

As explained in Sec. VI, the distribution of the electric field is considered as a Gaussian distribution whereas the beam waist of the beam acts as twice the standard deviation of the distribution. Algorithm 16 depicts the generation of the Gaussian distribution corresponding to the electric field of the beam. The algorithm takes as input the properties of the distribution such as the position of the center and the standard deviation. It also takes as input the extent in space over which the distribution will be generated. The algorithm returns the Gaussian distribution stored in an array.

Algorithm 17 describes the propagation of the Gaussian beams along the  $z$  axis following the assumptions considered in Secs. VB. The algorithm takes as input the electric field distribution at the initial point along with the extent of the beam at the initial and final point along the  $x$  axis. The wavelength of the beam and the distance along the  $z$  axis are also considered as inputs. The field amplitudes at the final point are calculated, for each point on the  $x$  axis over which the beam is spread, following Huygen's principle of beam propagation.

**Algorithm 16** Generation of electric field distribution following Gaussian distribution**Require:**

- FLOAT CENTER ▷ Position of the center of the Gaussian distribution  
 FLOAT STDDEV ▷ Standard deviation of the Gaussian distribution  
 FLOAT AMP ▷ Electric field amplitude at the center  
 FLOAT NUM ▷ Number of data points of the distribution  
 FLOAT LIMIT ▷ Extent in position over which the field is generated

**Ensure:**

- UNSIGNED INTEGER LENFIELDARR ▷ Length of the electric field array  
 FLOAT [LENFIELDARR]FIELDARR ▷ Electric field array

```

1: procedure FIELD DISTRIBUTION(CENTER, STDDEV, AMP, NUM, LIMIT)
2:   DYNAMIC POSARR ▷ Declaring the position array to be dynamic
3:   DYNAMIC FIELDARR ▷ Declaring the electric field array to be dynamic
4:   FLOAT POS ▷ Iterative position value
5:   FLOAT RES ▷ Resolution of the field distribution
6:   UNSIGNED INTEGER LENPOSARR ▷ Length of the position array
7:   FLOAT FIELDAMP ▷ Iterative amplitude of the electric field
8:   POS ← CENTER − LIMIT ▷ Initializing the iterative position value
9:   POSARR ← ∅ ▷ Initializing the dynamic position array to be a null set
10:  FIELDARR ← ∅ ▷ Initializing the dynamic electric field array to be a null set
11:  RES ←  $\frac{2 \times \text{LIMIT}}{\text{NUM}}$  ▷ Calculating the resolution for generating the position array
12:  while POS ≤ (CENTER + LIMIT) do ▷ Iterating over the limits of the position values
13:    POSARR ← POSARR ∪ POS ▷ Generating the position array
14:    POS ← POS + RES ▷ Incrementing the position value for each iteration
15:  end while
16:  LENPOSARR ← LENGTH (POSARR) ▷ Obtaining the length of the position array
17:  I ← 0 ▷ Initializing the iterator
18:  while I ≤ LENPOSARR do ▷ Iterating over the elements of the position array
19:    FIELDAMP ← AMP × exp  $\left( \frac{-(\text{POSARR}[I] - \text{CENTER})^2}{4 \times \text{STDDEV}^2} \right)$  ▷ Obtaining the electric field amplitude at each iteration
20:    FIELDARR ← FIELDARR ∪ FIELDAMP ▷ Updating the electric-field-distribution array with the amplitude value
    at each iteration
21:  end while
22:  LENFIELDARR ← LENGTH (FIELDARR) ▷ Obtaining the length of the electric field array
23:  return LENFIELDARR , FIELDARR ▷ Returning the specified outputs
24: end procedure

```

**Algorithm 17** Propagation of Gaussian beam**Require:**

- UNSIGNED INTEGER LENINFIELDARR ▷ Length of the input electric field array  
 FLOAT [LENINFIELDARR]INFIELDARR ▷ Input electric field array  
 FLOAT NUMIN ▷ Number of data points of the distribution of the input electric field  
 FLOAT NUMOUT ▷ Number of data points of the distribution of the output electric field  
 FLOAT LIMITIN ▷ Extent in position over which the input field is generated  
 FLOAT NUMOUT ▷ Number of data points of the distribution of the output electric field  
 FLOAT LIMITOUT ▷ Extent in position over which the output field will be generated  
 FLOAT CENTERIN ▷ Position of the center of the Gaussian distribution of the input field  
 FLOAT CENTEROUT ▷ Position of the center of the Gaussian distribution of the output field

---

FLOAT DIST                                   ▷ Distance between the two points along the  $z$  axis over which the beam is propagated

FLOAT LAMBDA                                   ▷ Wavelength of the beam

**Ensure:**

UNSIGNED INTEGER LENOUTFIELDARR                                   ▷ Length of the output electric field array

FLOAT [LENOUTFIELDARR]OUTFIELDARR                                   ▷ Output electric field array obtained after propagation

1: **procedure** FIELDISTRIBUTION([LENINFIELDARR]INFIELDARR, NUMIN, LIMITIN, NUMIUT, LIMITOUT, DIST, LAMBDA)

2:   DYNAMIC INPOSARR                                   ▷ Declaring the position array to be dynamic

3:   UNSIGNED INTEGER LENINPOSARR                                   ▷ Length of the position array

4:   DYNAMIC OUTPOSARR                                   ▷ Declaring the position array to be dynamic

5:   UNSIGNED INTEGER LENOUTPOSARR                                   ▷ Length of the position array

6:   DYNAMIC OUTFIELDARR                                   ▷ Declaring the electric field array to be dynamic

7:   FLOAT POS   ▷ Iterative position value

8:   FLOAT RES   ▷ Resolution of the field distribution

9:   UNSIGNED INTEGER I                                   ▷ Iterative variable as a pointer

10:   UNSIGNED INTEGER J                                   ▷ Iterative variable as a pointer

11:   FLOAT SUM   ▷ Integrated field amplitude value

12:   INPOSARR  $\leftarrow \emptyset$                                    ▷ Initializing the dynamic input position array to be a null set

13:   OUTPOSARR  $\leftarrow \emptyset$                                    ▷ Initializing the dynamic output position array to be a null set

14:   OUTFIELDARR  $\leftarrow \emptyset$                                    ▷ Initializing the dynamic output electric field array to be a null set

15:   RES  $\leftarrow \frac{2 \times \text{LIMITIN}}{\text{NUMIN}}$                                    ▷ Calculating the resolution for generating the position array corresponding to the input electric field

16:   POS  $\leftarrow \text{CENTERIN} - \text{LIMITIN}$                                    ▷ Initializing the iterative position value

17:   **while** POSIN  $\leq (\text{CENTERIN} + \text{LIMITIN})$  **do**                                   ▷ Iterating over the position range for the input electric field

18:     INPOSARR  $\leftarrow \text{INPOSARR} \cup \text{POS}$                                    ▷ Generating the position array

19:     POS  $\leftarrow \text{POS} + \text{RES}$                                    ▷ Incrementing the iterative position value for each iteration

20:   **end while**

21:   RES  $\leftarrow \frac{2 \times \text{LIMITOUT}}{\text{NUMOUT}}$                                    ▷ Calculating the resolution for generating the position array corresponding to the output electric field

22:   POS  $\leftarrow \text{CENTEROUT} - \text{LIMITOUT}$                                    ▷ Reinitializing the iterative position value

23:   **while** POS  $\leq (\text{CENTEROUT} + \text{LIMITOUT})$  **do**                                   ▷ Iterating over the position range for the output electric field

24:     OUTPOSARR  $\leftarrow \text{OUTPOSARR} \cup \text{POS}$                                    ▷ Generating the position array for the output electric field

25:     POS  $\leftarrow \text{POS} + \text{RES}$

26:   **end while**

27:   LENINPOSARR  $\leftarrow \text{LENGTH}(\text{INPOSARR})$                                    ▷ Obtaining the length of input electric-field-distribution array

28:   LENOUTPOSARR  $\leftarrow \text{LENGTH}(\text{OUTPOSARR})$                                    ▷ Obtaining the length of output electric-field-distribution array

29:   **while** I  $\leq \text{LENOUTPOSARR}$  **do**                                   ▷ Iterating over all the elements of the position array for the output field

30:     SUM  $\leftarrow 0$                                    ▷ Initializing the sum value, which corresponds to the field amplitude for iteration

31:     **while** doJ  $\leq \text{LENINPOSARR}$                                    ▷ Iterating over all the elements of the position array for the input field

32:       LEN  $\leftarrow \sqrt{(\text{OUTPOSARR}[I] - \text{INPOSARR}[J])^2 + \text{DIST}^2}$                                    ▷ Distance between two points of the input and output positions

33:       SUM  $\leftarrow \text{SUM} + \left( \frac{\exp\left(\frac{2\pi i \times \text{LEN}}{\text{LAMBDA}}\right)}{\text{LEN}} \times \text{INFIELDARR}[J] \right)$

34:       OUTFIELDARR  $\leftarrow \text{OUTFIELDARR} \cup \text{SUM}$                                    ▷ Generating the output electric-field-distribution array

35:     **end while**

36:   **end while**

37:   LENOUTFIELDARR  $\leftarrow \text{LENGTH}(\text{OUTFIELDARR})$                                    ▷ Calculating the length of the output field array

38:   **return** LENOUTFIELDARR, OUTFIELDARR                                   ▷ Returning the specified outputs

39: **end procedure**

---

Refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

**APPENDIX K: FIBER COUPLING EFFICIENCY**

The fiber coupling efficiency is obtained by calculating the overlap of the Gaussian distribution corresponding to the electric field intensity of the incident beam at the position of the fiber tip and the mode-field diameter of the fiber. Algorithm 18 described the numerical method followed to obtain the efficiency.

**Algorithm 18** Fiber coupling efficiency**Require:**

UNSIGNED INTEGER LENINFIELDARR ▷ Length of the input electric field array  
 FLOAT [LENINFIELDARR]INFIELDARR ▷ Input electric field array  
 FLOAT CENTER ▷ Position of the center of the Gaussian distribution  
 FLOAT LIMIT ▷ Extent in position over which the electric field is generated  
 FLOAT MODEMEDIA ▷ Mode-field diameter of the fiber

**Ensure:**

FLOAT COUPEFF ▷ Coupling efficiency

```

1: procedure COUPLING([LENINFIELDARR]INFIELDARR, MODEMEDIA, CENTER, LIMIT)
2:
3:   DYNAMIC INTARR ▷ Declaring the array for storing the electric field intensity to be dynamic
4:   FLOAT STDDEVFIELD ▷ Standard deviation of the Gaussian distribution corresponding to the electric field
   intensity distribution at the fiber position
5:   DYNAMIC POSARR ▷ Declaring the position array to be dynamic
6:   UNSIGNED INTEGER LENPOSARR ▷ Length of the position array
7:   FLOAT POS ▷ Iterative position value
8:   FLOAT RES ▷ Resolution of the field distribution
9:   FLOAT STDDEVFIBER ▷ Standard deviation of the Gaussian distribution corresponding to the mode diameter of
   the fiber
10:  UNSIGNED INTEGER I ▷ Declaring an iterative variable
11:  FLOAT INTPEAK ▷ Peak value of the electric field distribution
12:  FLOAT INT ▷ Iterative intensity value for the electric field
13:  INTARR ← ∅ ▷ Initializing the dynamic electric field intensity array to be a null set
14:  POS ← CENTER − LIMIT ▷ Initializing the iterative position value
15:  POSARR ← ∅ ▷ Initializing the dynamic position array to be a null set
16:  RES ←  $\frac{2 \times \text{LIMIT}}{\text{LENINFIELDARR}}$  ▷ Calculating the resolution for generating the position array
17:  while POS ≤ (CENTER + LIMIT) do ▷ Iterating over the elements of the input electric field array
18:    POSARR ← POSARR ∪ POS ▷ Generating the position array
19:    POS ← POS + RES
20:  end while
21:  I ← 1 ▷ Initializing the iterative variable
22:  while I ≤ LENINFIELDARR do
23:    INTARR ← INTARR ∪ ABSOLUTE (INFIELDARR [I]) ▷ Generating the electric field intensity for the output
   electric field
24:  end while
25:  INTPEAK ← MAXIMUM (INTARR) ▷ Obtaining the maximum value of the intensity array
26:  POS ← 1 ▷ Initializing the iterative position value
27:  INT ← INTARR [POS] ▷ Initializing the iterative intensity value
28:  while INT ≤  $\left(\frac{\text{INTPEAK}}{e^2}\right)$  do ▷ Conditioning on the value of the electric field intensity being lower than the set
   threshold value
29:    POS ← POS + 1 ▷ Incrementing the position value at each iteration
30:    INT ← INTARR [POS] ▷ Updating the intensity value at each iteration
31:  end while
32:  STDDEVFIELD ←  $\frac{\text{POSARR}[\text{POS}]}{2}$  ▷ Calculating the standard deviation for the input beam Gaussian distribution

```

---

```

33:   STDDEVFIBER ←  $\frac{\text{MODEDIA}}{4}$  ▷ Calculating the standard deviation for the Gaussian distribution corresponding to
    the fiber mode field diameter
34:   COUPEFF ← OVERLAP (STDDEVFIBER, STDDEVFIELD) ▷ Calculating the overlap of the two Gaussian
    distributions
35:   return COUPEFF ▷ Returning the specified outputs
36: end procedure

```

---

Refer to Appendix L for details on the data types and the libraries that are used at various instances in the above algorithm.

## APPENDIX L: GENERAL DATA TYPES AND LIBRARY FUNCTIONS

### 1. Data types

Each variables used in the algorithms are declared as a certain data type. Here we list the different data types that are used to declare the variables.

- (a) UNSIGNED INTEGER: declares the variable as a positive integer ( $\mathbb{Z}^+$ ) in the pseudocode.
- (b) SIGNED INTEGER: declares the variable as an integer ( $\mathbb{Z}$ ) that can take both positive and negative values in the pseudocode.
- (c) REAL: declares the variable as a real number ( $\mathbb{R}$ ) that can take both positive and negative values in the pseudocode.
- (d) UNIT INTERVAL: declares the variable to be in the open interval  $(0, 1)$  in the pseudocode, i.e., the variable can take any real value ( $\mathbb{R}$ ) within the interval.
- (e) INTERVAL( $a, b$ ): declares the variable to be in the open interval  $(a, b)$  in the pseudocode where  $a$  and  $b$  are real numbers( $\mathbb{R}$ ).
- (f) DYNAMIC: declares the variable as a dynamic array in the pseudocode.
- (g) CHAR: declares the variable that can store a single character.

### 2. Libraries

Here we list the various functions that are used in the algorithms.

- (a) RANDOM( $a, b$ ): returns a random number between  $a$  and  $b$  sampled from a uniform distribution
- (b) NORMALRANDOM( $\mu, \sigma$ ): returns a random number sampled from a Gaussian (normal) distribution of mean  $\mu$  and standard deviation  $\sigma$ .
- (c) INTERPOLATE( $X, Y$ ): This function takes as input two arrays  $X$  (domain) and  $Y$  (range) of same size and returns an interpolated function  $f : y = f(x)$ . An instance of this class is created by passing the two 1D vectors ( $X$  and  $Y$ ) comprising the data and a function is created out of it using linear interpolation. Behavior at the boundary can be specified at instantiation time, which by default is a linear spline if not specified otherwise.
- (d) LENGTH( $A$ ): returns the length of the sequence or list (1D array)  $A$ .
- (e) ROUND( $a$ ): returns the integer value closest to the real variable  $a$ .
- (f) RANDOMSEQ( $a, b, c$ ): returns a sequence of random integers between  $a$  and  $b$  sampled from an uniform distribution of length  $c$ .
- (g) SEC( $k$ ): returns the converted time in seconds for an input time interval  $k$  in ps.
- (h) GETTIMESTAMPS( $i, S, k, L$ ): returns a randomly selected chunk of contiguous time stamps of length  $k$ , starting from the  $S[i]$ th position, from the list  $L$ .
- (i) QKDRATE( $wl_1, wr_1, wl_2, wr_2, A, B, C$ ): returns the total key rate between coincidence window markers:  $wl_1$  &  $wr_1$  as well as  $wl_2$  &  $wr_2$  for cross-correlations between the detection at  $A$  &  $B$  as well as those at  $A$  &  $C$ , respectively.
- (j) MEAN( $L$ ): returns the mean of the sequence or list (1D array)  $L$ .
- (k) SD( $L$ ): returns the standard deviation between values in the sequence or list (1D array)  $L$ .
- (l) QKD( $wl, wr, lenA, A, lenBR, BR, lenBD, BD, t, c$ ): returns a  $1 \times 4$  dimensional list (or an array) containing values in the order: key rate, QBER, signal (corresponding to match of polarization basis), noise (corresponding to mismatch of polarization basis), and SNR ratio. The values are estimated between coincidence windows:  $wl$  &  $wr$ , owing to cross-correlations between the time stamps recorded in lists  $A$  [ $lenA$ ] with  $BR$  [ $lenBR$ ] for  $c = 'R'$ , i.e., along rectilinear basis

(or  $BD[lenBD]$  for  $c = 'D'$ , i.e., along diagonal basis). The key rate is estimated over a measurement time  $t$ . For detailed algorithmic description refer to Algorithm 4.

(m)  $CEIL(N)$ : return the smallest integer value that is bigger than or equal to the number  $N$ .

(n)  $FLOOR(N)$ : return the largest integer value that is smaller than or equal to the number  $N$ .

- 
- [1] Valerio Scarani, Helle Bechmann-Pasquinucci, Nicolas J. Cerf, Miloslav Dusek, Norbert Lütkenhaus, and Momtchil Peev, The security of practical quantum key distribution, *Rev. Mod. Phys.* **81**, 1301 (2009).
- [2] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, Cryptographic communications system and method, US Patent 4,405,829 (1983).
- [3] Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel, and Hugo Zbinden, Quantum cryptography, *Rev. Mod. Phys.* **74**, 145 (2002).
- [4] Osamu Hirota, A correct security evaluation of quantum key distribution, arXiv:1409.5991 (2014).
- [5] G. S. Vernam, Cipher printing telegraph systems for secret wire and radio telegraphic communications, *J. AIEE*, **45**, 109 (1926).
- [6] Imran Khan, Bettina Heim, Andreas Neuzner, and Christoph Marquardt, Satellite-based qkd, *Opt. Photonics News* **29**, 26 (2018).
- [7] Battelle: Quantum key distribution, <https://www.battelle.org/case-studies/case-study-detail/quantum-key-distribution>.
- [8] Yuan Cao, Yongli Zhao, Jianquan Wang, Xiaosong Yu, Zhangchao Ma, and Jie Zhang, Sdqaas: Software defined networking for quantum key distribution as a service, *Opt. Exp.* **27**, 6892 (2019).
- [9] Guan-Jie Fan-Yuan, Chao Wang, Shuang Wang, Zhen-Qiang Yin, He Liu, Wei Chen, De-Yong He, Zheng-Fu Han, and Guang-Can Guo, Afterpulse analysis for quantum key distribution, *Rev. Appl. Phys.* **10**, 064032 (2018).
- [10] Phuc V. Trinh, Thanh V. Pham, Ngoc T. Dang, Hung Viet Nguyen, Soon Xin Ng, and Anh T. Pham, Design and security analysis of quantum key distribution protocol over free-space optics using dual-threshold direct-detection receiver, *IEEE Access* **6**, 4159 (2018).
- [11] Phuc V. Trinh, Thanh V. Pham, Hung V. Nguyen, Soon Xin Ng, and Anh T. Pham, in *2016 IEEE Globecom Workshops (GC Wkshps)* (IEEE, Washington, DC, USA, 2016), p. 1.
- [12] L. O. Mailloux, R. D. Engle, M. R. Grimaila, D. D. Hodson, J. M. Colombi, and C. V. McLaughlin, Modeling decoy state quantum key distribution systems, *J. Defense Model. Simul.* **12**, 489 (2015).
- [13] Shuang Zhao and Hans De Raedt, Event-by-event simulation of quantum cryptography protocols, *J. Comput. Theor. Nanosci.* **5**, 490 (2008).
- [14] Marcin Niemiec, ukasz Romański, and Marcin Świety, in *International Conference on Multimedia Communications, Services and Security* (Springer, Krakow, Poland, 2011), p. 286.
- [15] Gabriela Mogos, in *Proc. 18th Conf. Quantum Inf. Process* (Sydney, Australia, 2015).
- [16] Abudhahir Buhari, Zuriati Ahmad Zukarnain, Shamla K. Subramaniam, Hishamuddin Zainuddin, and Suhairi Saharudin, in *2012 IEEE Symposium on Industrial Electronics and Applications* (IEEE, Bandung, Indonesia, 2012), p. 84.
- [17] N. Hafizah Mohamed Halip, M. Mokhtar, and A. Buhari, in *2014 IEEE 5th International Conference on Photonics (ICP)* (IEEE, Kuala Lumpur, Malaysia, 2014), p. 29.
- [18] Abudhahir Buhari, Zuriati Ahmad Zukarnai, Shamala K. Subramaniam, Hisham Zainuddin, and Suhairi Saharudin, in *Proc. Int. Conf. Comput. Intell. Syst., Int. Conf. Elect., Electron. (ICCIS)* (Bangkok, Thailand, 2012), p. 30.
- [19] Hui Qiao and Xiao yu Chen, in *Proceedings of 14th Youth Conference on Communication* (Academic Publisher, Dalian, China, 2009).
- [20] Omer K. Jasim, Safia Abbas, El-Sayed M. El-Horbaty, and Abdel-Badeeh M. Salem, Quantum key distribution: Simulation and characterizations, *Proc. Comput. Sci.* **65**, 701 (2015). *International Conference on Communications, management, and Information technology (ICCMIT'2015)*.
- [21] A. Atashpenda, Simulation and analysis of qkd (bb84), <http://www.qkdsimulator.com/>.
- [22] BB84 demo, <http://fredhenle.net/bb84/demo.php>.
- [23] NumericalQKD: Qkd security analysis software, <https://lutkenhausgroup.wordpress.com/qkd-software/>.
- [24] QKNetSim: Quantum key distribution network simulation module, <http://www.qkdnetstim.info/doc/models/build/html/qkd.html>.
- [25] W. Lardier, Q. Varo, and J. Yan, in *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)* (IEEE, Beijing, China, 2019), p. 1.
- [26] Open quantum safe, <https://openquantumsafe.org/>.
- [27] Open QKD network, <https://openqkdnetwork.ca/>.
- [28] Charles H. Bennett and Gilles Brassard, Quantum cryptography: Public key distribution and coin tossing, *Theor. Comput. Sci.* **560**, 7 (2014). *Theoretical Aspects of Quantum Cryptography – celebrating 30 years of BB84*.
- [29] Charles H. Bennett and Gilles Brassard, Experimental quantum cryptography: The dawn of a new era for quantum cryptography: The experimental prototype is working, *ACM Sigact News* **20**, 78 (1989).
- [30] Hoi-Kwong Lo and Hoi Fung Chau, Unconditional security of quantum key distribution over arbitrarily long distances, *Science* **283**, 2050 (1999).
- [31] Daniel Gottesman, Hoi-Kwong Lo, Norbert Lütkenhaus, and John Preskill, Security of quantum key distribution with imperfect devices, *Quantum Info. Comput.* **4**, 325 (2004).
- [32] Charles H. Bennett, Quantum Cryptography Using any two Nonorthogonal States, *Phys. Rev. Lett.* **68**, 3121 (1992).
- [33] W. T. Buttler, R. J. Hughes, P. G. Kwiat, G. G. Luther, G. L. Morgan, J. E. Nordholt, C. G. Peterson, and

- C. M. Simmons, Free-space quantum-key distribution, *Phys. Rev. A* **57**, 2379 (1998).
- [34] W. T. Buttler, R. J. Hughes, Paul G. Kwiat, S. K. Lamoreaux, G. G. Luther, G. L. Morgan, J. E. Nordholt, C. G. Peterson, and C. M. Simmons, Practical free-space quantum key distribution over 1 km, *Rev. Lett. Phys.* **81**, 3283 (1998).
- [35] Artur K. Ekert, Quantum Cryptography Based on Bell's Theorem, *Phys. Rev. Lett.* **67**, 661 (1991).
- [36] Charles H. Bennett, Gilles Brassard, and N. David Mermin, Quantum Cryptography Without Bell's Theorem, *Phys. Rev. Lett.* **68**, 557 (1992).
- [37] C. Erven, C. Couteau, R. Laflamme, and G. Weihs, Entangled quantum key distribution over two free-space optical links, *Opt. Exp.* **16**, 16840 (2008).
- [38] Alexander Ling, Matt Peloso, Ivan Marcikic, Antia Lamas-Linares, and Christian Kurtsiefer, Experimental e91 quantum key distribution, *Proc. SPIE*, **6903**, 69030-U (2008).
- [39] Dominic Mayers and Andrew Yao, in *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)* (IEEE, New York, USA, 1998), p. 503.
- [40] Umesh Vazirani and Thomas Vidick, Fully Device-Independent Quantum key Distribution, *Phys. Rev. Lett.* **113**, 140501 (2014).
- [41] Carl A. Miller and Yaoyun Shi, Robust protocols for securely expanding randomness and distributing keys using untrusted quantum devices, *J. ACM* **63**, 33 (2016).
- [42] Antonio Acín, Nicolas Brunner, Nicolas Gisin, Serge Massar, Stefano Pironio, and Valerio Scarani, Device-Independent Security of Quantum Cryptography against Collective Attacks, *Phys. Rev. Lett.* **98**, 230501 (2007).
- [43] Stefano Pironio, Antonio Acín, Nicolas Brunner, Nicolas Gisin, Serge Massar, and Valerio Scarani, Device-independent quantum key distribution secure against collective attacks, *New J. Phys.* **11**, 045021 (2009).
- [44] Rotem Arnon-Friedman, Frederic Dupuis, Omar Fawzi, Renato Renner, and Thomas Vidick, Practical device-independent quantum cryptography via entropy accumulation, *Nat. Commun.* **9**, 459 (2018).
- [45] Simon J. D. Phoenix, Stephen M. Barnett, and Anthony Chefles, Three-state quantum cryptography, *J. Mod. Opt.* **47**, 507 (2000).
- [46] J.-C. Boileau, K. Tamaki, J. Batuwantudawe, R. Laflamme, and J. M. Renes, Unconditional Security of a Three State Quantum key Distribution Protocol, *Phys. Rev. Lett.* **94**, 040503 (2005).
- [47] Charles H. Bennett, Francois Bessette, Gilles Brassard, Louis Salvail, and John Smolin, Experimental quantum cryptography, *J. Cryptol.* **5**, 3 (1992).
- [48] Michael Hatcher, Cryptography beats 100 km barrier, <http://optics.org/articles/news/9/6/3/1>.
- [49] Sheng-Kai Liao, Wen-Qi Cai, Wei-Yue Liu, Liang Zhang, Yang Li, Ji-Gang Ren, Juan Yin, Qi Shen, Yuan Cao, Zheng-Ping Li, *et al.*, Satellite-to-ground quantum key distribution, *Nature* **549**, 43 (2017).
- [50] Bernd Fröhlich, Marco Lucamarini, James F. Dynes, Lucian C. Comandar, Winci W.-S. Tam, Alan Plews, Andrew W. Sharpe, Zhiliang Yuan, and Andrew J. Shields, Long-distance quantum key distribution secure against coherent attacks, *Optica* **4**, 163 (2017).
- [51] Yang Liu, Teng-Yun Chen, Jian Wang, Wen-Qi Cai, Xu Wan, Luo-Kan Chen, Jin-Hong Wang, Shu-Bin Liu, Hao Liang, Lin Yang, *et al.*, Decoy-state quantum key distribution with polarized photons over 200 km, *Opt. Exp.* **18**, 8587 (2010).
- [52] Shuang Wang, Wei Chen, Zhen-Qiang Yin, De-Yong He, Cong Hui, Peng-Lei Hao, Guan-Jie Fan-Yuan, Chao Wang, Li-Jun Zhang, Jie Kuang, *et al.*, Practical gigahertz quantum key distribution robust against channel disturbance, *Opt. Lett.* **43**, 2030 (2018).
- [53] Richard J. Hughes, Jane E. Nordholt, Derek Derkacs, and Charles G. Peterson, Practical free-space quantum key distribution over 10 km in daylight and at night, *New J. Phys.* **4**, 43 (2002).
- [54] John G. Rarity, P. R. Tapster, P. M. Gorman, and Peter Knight, Ground to satellite secure key exchange using quantum cryptography, *New J. Phys.* **4**, 82 (2002).
- [55] Kevin Günthner, Imran Khan, Dominique Elser, Birgit Stiller, Ömer Bayraktar, Christian R. Müller, Karen Saucke, Daniel Tröndle, Frank Heine, Stefan Seel, *et al.*, Quantum-limited measurements of optical signals from a geostationary satellite, *Optica* **4**, 611 (2017).
- [56] Juan Yin, Yuan Cao, Yu-Huai Li, Ji-Gang Ren, Sheng-Kai Liao, Liang Zhang, Wen-Qi Cai, Wei-Yue Liu, Bo Li, Hui Dai, *et al.*, Satellite-To-Ground Entanglement-Based Quantum key Distribution, *Phys. Rev. Lett.* **119**, 200501 (2017).
- [57] Jian-Yu Wang, Bin Yang, Sheng-Kai Liao, Liang Zhang, Qi Shen, Xiao-Fang Hu, Jin-Cai Wu, Shi-Ji Yang, Hao Jiang, Yan-Lin Tang, *et al.*, Direct and full-scale experimental verifications towards ground-satellite quantum key distribution, *Nat. Photonics* **7**, 387 (2013).
- [58] Sheng-Kai Liao, Hai-Lin Yong, Chang Liu, Guo-Liang Shentu, Dong-Dong Li, Jin Lin, Hui Dai, Shuang-Qiang Zhao, Bo Li, Jian-Yu Guan, *et al.*, Long-distance free-space quantum key distribution in daylight towards inter-satellite communication, *Nat. Photonics* **11**, 509 (2017).
- [59] Id quantique, <https://www.idquantique.com>.
- [60] Magiq technologies, <http://www.magiqtech.com>.
- [61] Qasky, <http://www.qasky.com/en/>.
- [62] Quintessencelabs, <https://www.quintessencelabs.com>.
- [63] Robert Bedington, Juan Miguel Arrazola, and Alexander Ling, Progress in satellite quantum key distribution, *npj Quantum Inf.* **3**, 1 (2017).
- [64] Philip Sibson, Chris Erven, Mark Godfrey, Shigehito Miki, Taro Yamashita, Mikio Fujiwara, Masahide Sasaki, Hiro-taka Terai, Michael G. Tanner, Chandra M. Natarajan, *et al.*, Chip-based quantum key distribution, *Nat. Commun.* **8**, 13984 (2017).
- [65] Davide Bacco, Ilaria Vagniluca, Beatrice Da Lio, Nicola Biagi, Adriano Della Frera, Davide Calonico, Costanza Toninelli, Francesco S. Cataliotti, Marco Bellini, Leif K. Oxenlwe, *et al.*, Field trial of a three-state quantum key distribution scheme in the florence metropolitan area, *EPJ Quantum Technol.* **6**, 5 (2019).
- [66] Davide Rusca, Alberto Boaron, Fadri Grünenfelder, Anthony Martin, and Hugo Zbinden, Finite-key analysis for the 1-decoy state qkd protocol, *Appl. Phys. Lett.* **112**, 171104 (2018).

- [67] Renato Renner, Nicolas Gisin, and Barbara Kraus, Information-theoretic security proof for quantum-key-distribution protocols, *Phys. Rev. A* **72**, 012332 (2005).
- [68] Ian Sommerville, *Software Engineering* (Addison-Wesley Publishing Company, USA, 2010), 9th ed.
- [69] Roger S. Pressman, *Software Engineering: A Practitioner's Approach* (McGraw-Hill Higher Education, New York, NY, 1996), 4th ed.
- [70] N. Munassar, A. Govardhan, Karimnagar Dt Jagityal, and Andhra Pradesh, in *Proceedings of the 11th International Arab Conference on Information Technology* (IEEE, Giza, Egypt, 2010).
- [71] Ulf Eriksson, How to make agile and waterfall methodologies work together, <https://reqtest.com/agile-blog/agile-waterfall-hybrid-methodology-2>.
- [72] M. Bourennane, F. Gibson, A. Karlsson, A. Hening, P. Jonsson, T. Tsegaye, D. Ljunggren, and E. Sundberg, Experiments on long wavelength (1550 nm) “plug and play” quantum cryptography systems, *Opt. Express* **4**, 383 (1999).
- [73] Richard J. Hughes, George L. Morgan, and C. Glen Peterson, Quantum key distribution over a 48 km optical fibre network, *J. Mod. Opt.* **47**, 533 (2000).
- [74] W. T. Buttler, R. J. Hughes, P. G. Kwiat, S. K. Lamoreaux, G. G. Luther, G. L. Morgan, J. E. Nordholt, C. G. Peterson, and C. M. Simmons, Practical Free-Space Quantum key Distribution Over 1 km, *Phys. Rev. Lett.* **81**, 3283 (1998).
- [75] R. J. Hughes, W. T. Buttler, P. G. Kwiat, S. K. Lamoreaux, G. L. Morgan, J. E. Nordholt, and C. G. Peterson, in *2000 IEEE Aerospace Conference. Proceedings* (Cat. No. 00TH8484) Vol. 1 (IEEE, Big Sky, MT, USA, 2000), p. 191.
- [76] L. Ma, T. Chang, A. Mink, O. Slattery, B. Hershman, and X. Tang, Experimental demonstration of a detection-time-bin-shift polarization encoding quantum key distribution system, *IEEE Commun. Lett.* **12**, 459 (2008).
- [77] Jeffrey Wilson, Dalton Chaffee, Nathaniel Wilson, John Lekki, Roger Tokars, John Pouch, Tony Roberts, Philip Battle, Bertram Floyd, Alexander Lind, John Cavin, and Spencer Helmick, Free-space quantum key distribution with a high generation rate potassium titanyl phosphate waveguide photon-pair source, *Quantum Commun. Quantum Imaging XIV*, **9980**, 99800U (2016).
- [78] W. T. Buttler, R. J. Hughes, S. K. Lamoreaux, G. L. Morgan, J. E. Nordholt, and C. G. Peterson, Daylight Quantum key Distribution Over 1.6 km, *Phys. Rev. Lett.* **84**, 5652 (2000).
- [79] J. C. Bienfang, A. J. Gross, A. Mink, B. J. Hershman, A. Nakassis, X. Tang, R. Lu, D. H. Su, Charles W. Clark, Carl J. Williams, E. W. Hagley, and Jesse Wen, Quantum key distribution with 1.25 gbps clock synchronization, *Opt. Express* **12**, 2011 (2004).
- [80] Ronald Meyers and Keith Deacon, Entangled and non-line-of-sight (nlos) free-space photon quantum communication, *J. Opt. Netw.* **4**, 573 (2005).
- [81] K. J. Gordon, V. Fernandez, P. D. Townsend, and G. S. Buller, A short wavelength gigahertz clocked fiber-optic quantum key distribution system, *IEEE J. Quantum Electron.* **40**, 900 (2004).
- [82] Xiao Tang, Lijun Ma, Alan Mink, Anastase Nakassis, Hai Xu, Barry Hershman, Joshua C. Bienfang, David Su, Ronald F. Boisvert, Charles W. Clark, and Carl J. Williams, Experimental study of high speed polarization-coding quantum key distribution with sifted-key rates over mbit/s, *Opt. Express* **14**, 2062 (2006).
- [83] M. J. García-Martínez, N. Denisenko, D. Soto, D. Arroyo, A. B. Orue, and V. Fernandez, High-speed free-space quantum key distribution system for urban daylight applications, *Appl. Opt.* **52**, 3311 (2013).
- [84] Giuseppe Vallone, Davide G. Marangon, Matteo Canale, Ilaria Savorgnan, Davide Bacco, Mauro Barbieri, Simon Calimani, Cesare Barbieri, Nicola Laurenti, and Paolo Villoresi, Adaptive real time selection for quantum key distribution in lossy and turbulent free-space channels, *Phys. Rev. A* **91**, 042320 (2015).
- [85] Chi-hang Fred Fung, Kiyoshi Tamaki, Bing Qi, Hoi-Kwong Lo, and Xiongfeng Ma, Security proof of quantum key distribution with detection efficiency mismatch, *Quantum Inf. Comput.* **9**, 131 (2009).
- [86] Xiongfeng Ma, Bing Qi, Yi Zhao, and Hoi-Kwong Lo, Practical decoy state for quantum key distribution, *Phys. Rev. A* **72**, 012326 (2005).
- [87] Robert W. Boyd, *Nonlinear Optics* (Elsevier, Burlington, MA, USA, 2003).
- [88] Leonard Mandel and Emil Wolf, *Optical Coherence and Quantum Optics* (Cambridge University Press, New York, USA, 1995).
- [89] Kiyoshi Kato and Eiko Takaoka, Sellmeier and thermo-optic dispersion formulas for ktp, *Appl. Opt.* **41**, 5040 (2002).
- [90] John D. Bierlein and Herman Vanherzeele, Potassium titanyl phosphate: Properties and new applications, *JOSA B* **6**, 622 (1989).
- [91] K. Fradkin, A. Arie, A. Skliar, and G. Rosenman, Tunable midinfrared source by difference frequency generation in bulk periodically poled ktiopo 4, *Appl. Phys. Lett.* **74**, 914 (1999).
- [92] Andreas Ahlrichs, Ph.D. thesis, School Humboldt-Universität zu Berlin, 2019.
- [93] Shai Emanuel and Ady Arie, Temperature-dependent dispersion equations for ktiopo 4 and ktiopso 4, *Appl. Optics* **42**, 6661 (2003).
- [94] W. Wiechmann, Shigeo Kubota, T. Fukui, and Hisashi Masuda, Refractive-index temperature derivatives of potassium titanyl phosphate, *Opt. Lett.* **18**, 1208 (1993).
- [95] Rodney Loudon, *The Quantum Theory of Light* (Clarendon Press, Oxford, 1973).
- [96] John David Jackson, *Classical electrodynamics* (John Wiley & Sons, Inc., USA, 1962).
- [97] Plamen Petrov, Ph.D. thesis, School University of Copenhagen, 2006.
- [98] Ryan S. Bennink, Optimal collinear gaussian beams for spontaneous parametric down-conversion, *Phys. Rev. A* **81**, 053805 (2010).
- [99] Matthias Scholz, Lars Koch, and Oliver Benson, Analytical treatment of spectral properties and signal-idler intensity correlations for a double-resonant optical parametric oscillator far below threshold, *Opt. Commun.* **282**, 3518 (2009).