IEEE Transactions on Quantum Engineering

Received 11 May 2023; revised 22 September 2023; accepted 9 October 2023; date of publication 19 October 2023; date of current version 28 November 2023.

Digital Object Identifier 10.1109/TQE.2023.3326093

Scalable QKD Postprocessing System With Reconfigurable Hardware Accelerator

NATARAJAN VENKATACHALAM[1] (Member, IEEE), FORAM P. SHINGALA[1], SELVAGANGAI C[1], HEMA PRIYA S[1], DILLIBABU S[1], POOJA CHANDRAVANSHI[2], AND RAVINDRA P. SINGH[2]

[1]Society for Electronic Transactions and Security, Chennai 600113, India
[2]Physical Research Laboratory, Ahmedabad 380009, India

Corresponding author: Natarajan Venkatachalam (e-mail: natarajan.v.in@ieee.org).

This work was supported by the Department of Science and Technology, within the Ministry of Science and Technology, Government of India, through the "Quantum enabled Science and Technology" program and Ministry of Electronics and Information Technology.

ABSTRACT Key distillation is an essential component of every quantum key distribution (QKD) system because it compensates for the inherent transmission errors of a quantum channel. However, the interoperability and throughput aspects of the postprocessing components are often neglected. In this article, we propose a high-throughput key distillation framework that supports multiple QKD protocols, implemented in a field-programmable gate array (FPGA). The proposed design adapts a MapReduce programming model to efficiently process large chunks of raw data across the limited computing resources of an FPGA. We present a novel hardware-efficient integrated postprocessing architecture that offers dynamic error correction, mutual authentication with a physically unclonable function, and an inbuilt high-speed encryption application that utilizes the key for secure communication. In addition, we have developed a semiautomated high-level synthesis framework that is compatible with any discrete variable QKD system, showing promising speedup. Overall, the experimental results demonstrate a noteworthy enhancement in scalability achieved through the utilization of a single FPGA platform.

INDEX TERMS High-level synthesis (HLS), key distillation engine (KDE), MapReduce framework, physical unclonable function (PUF), quantum key distribution (QKD).

I. INTRODUCTION

Secure data communication is a vital challenge in today's high-speed networks. The basic and most critical element of a cryptographic solution is the encryption key, as defined by Auguste Kerckhoffs in 1883 [15]. Traditional cryptographic techniques that rely on the complexity of mathematical problems may become vulnerable in the face of the emergence of quantum computers. Nevertheless, the same principles of quantum mechanics that could potentially provide an adversary with significant computational advantage can also be harnessed to achieve unconditional security when establishing a secret key between two communicating parties using quantum key distribution (QKD). Different implementations of QKD protocols exist, and they vary in how information is encoded and decoded using quantum states [4], [12], [14], [29]. In general, it comprises two channels: a quantum communication channel for transmitting quantum information and an authenticated classical channel. A quantum communication channel transmits a quantum state of a photon in a specific degree of freedom, such as polarization, time bin, phase, frequency, and so on, which serves as a means for encoding and decoding key information. The authenticated classical channel is used for secret key reconciliation and is also required to synchronize the transmitter and the receiver, often separated by large distances. Owing to the inherently noisy nature of the quantum channel, measurement device imperfections, and noisy encoding/decoding, the raw key vector extracted from the quantum channel may contain bit-flip errors. Consequently, postprocessing techniques are required to construct the final secret key at both the ends. A robust implementation of QKD that provides sufficient throughput in

terms of the key rate in a real-time environment is challenging. Achieving the speed of quantum communication necessitates a fast key distillation layer with efficient control hardware.

The QKD postprocessing can be broken down into submodules, namely: 1) synchronization; 2) sifting for erasure; 3) sifting for basis reconciliation; 4) random sampling; 5) parameter estimation (PE); 6) information reconciliation (IR) and verification; 7) privacy amplification (PA); and 8) key management. These classical components require massive computing and memory resources, which is why they were earlier implemented on server systems. However, owing to complex infrastructure and the security assumption of device isolation (trusted node architecture) [30], there is a need for a solution that is stand-alone, compact, and reconfigurable at low power consumption. Hence, the research has now progressed toward field-programmable gate array (FPGA) accelerators. A hardware description language (HDL) is a language used to program FPGAs [13]. Any HDL design is directly correlated with resource consumption in the FPGA. As the design becomes more complex, the need to streamline the design-flow process has led FPGA developers to explore software-based productivity tools that automate the register-transfer level (RTL) design flow; high-level synthesis (HLS) is one such tool, and its adaptation to QKD key distillation engine (KDE) is further described in Section III.

One of the major engineering challenges in real-time implementations of high-performance QKD networks is the continuous storage and processing of large amount of data, resulting in memory and computational overhead on the targeted systems. Given that quantum encoding takes place at frequencies in the hundreds of gigahertz range, it is essential that classical postprocessing techniques operate in real time to facilitate the extraction of a secure secret key at a level consistent with the quantum process. The IR phase in postprocessing involves excessive dynamic computations and memory accesses. Therefore, the overall processing time is dominated by complex computations and unstructured memory management.

Recently, there has been an increased interest in accelerating postprocessing using FPGAs [8], [9], [16], [25], [31], [33] with limited memory storage and management capabilities. However, extended memory units, such as static random-access memory (SRAMs) and dynamic random-access memory (DRAMs), can be used along with the FPGA to overcome this drawback. A technological gap still exists in the practical implementation of a high-throughput and efficient hardware–software codesign for large-scale quantum key distillation. In this article, we propose an FPGA design for large-volume data processing based on the MapReduce programming model. This design aims to improve processing throughput while ensuring security. We report a comprehensive experimental study to evaluate the performance and efficiency of different QKD protocols.

We summarize the main contributions of our work as follows.

1) A hardware-based KDE is designed with the capability to support multiprotocol discrete variable (DV) QKD systems.
2) A hardware-based MapReduce accelerator is designed to achieve a significant speedup of the computationally intensive tasks related to IR and PA.
3) A reconfigurable architecture attributed to a framework is developed through HLS technology.
4) Mutual authentication for QKD systems using a physically unclonable function (PUF) is implemented.
5) Rate-adaptive error reconciliation codes are utilized to optimize classical channel throughput, with enhanced error correction capacity.
6) An on-device high-speed encryptor with a throughput of up to 10 Gb/s is included.
7) A detailed experimental field trial for the following three different protocols is presented:
   a) Coherent one way (COW) [26];
   b) BB84, developed by Charles H. Bennett and Gilles Brassard in 1984. It is named after the duo's surnames (Bennett and Brassard, BB84) [37];
   c) BBM92, developed by Charles H. Bennett, Gilles Brassard, and N. David Mermin in 1992. It is named after the trio's surnames (Bennett, Brassard, and Mermin, BBM92) [36].

The rest of this article is organized as follows. Section II covers related work and literature review. Section III covers the proposed system design, architecture, and implementation of KDE in hardware. Section IV defines the experimental setup of the QKD protocols, and Section V provides the implementation results and the performance analysis of the design. Section VI describes the future work and open challenges. Finally, Section VII concludes this article.

II. RELATED WORK

One of the early attempts in 2012 to design a complete compact QKD system that integrated optics and control hardware into a single chassis was made by Zhang et al. [33]. They implemented the decoy-state BB84 protocol, but the key distillation software stack faced challenges due to inefficient computing devices, resulting in lower key rates. Around the same time, Tanaka et al. [27] achieved a high-speed phase-encoded BB84 QKD system that covered a distance of 50 km. They transmitted with a repetition frequency of tens of gigahertz using parallel transmission of photons and wavelength division multiplexing. However, their work underscored due to the requirement of massive computing and memory resources, making the system unsuitable for portable applications.

IEEE Transactions on Quantum Engineering

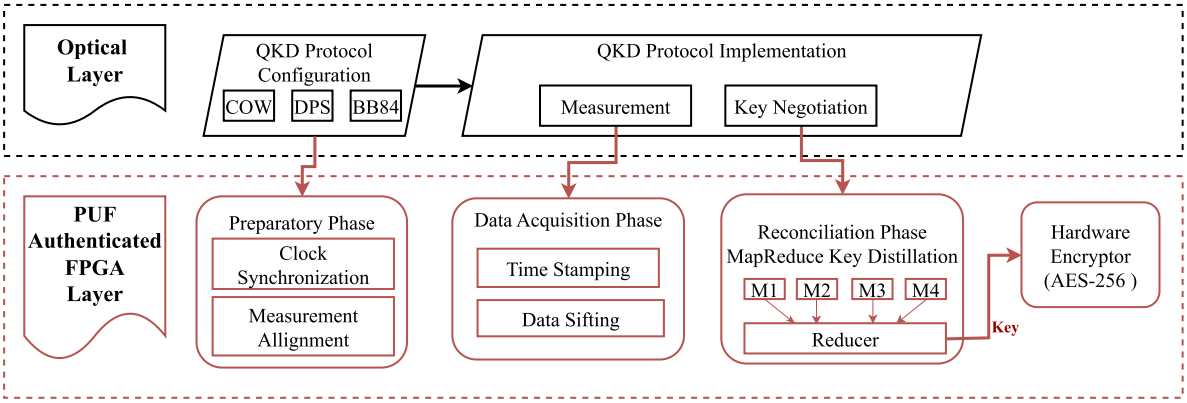

FIGURE 1. Overview of the FPGA-based support system for multi-QKD protocols with corresponding postprocessing stages; M {1, 2, 3, 4} represents the number of parallel mapper instances.

Efforts to accelerate data processing of measured qubits have led to research on individual modules within the QKD postprocessing. Cui et al. [9] focused on an efficient implementation of the error reconciliation module by exploiting FPGA parallelism and pipelining the execution between read, write, and compute operations. Walenta et al. [28] made significant strides toward a field-deployable QKD system by integrating control hardware and data processing units. Constantin et al. [8] provided a detailed description of key distillation modules, all implemented on a single Xilinx Virtex 6 FPGA. However, the system size remained substantial. An alternative to FPGA-accelerated postprocessing is a pipelined software implementation. Zhou et al. [34] proposed a multithreaded pipelined approach that utilized multiple CPU cores to optimize performance parameters. Yuan et al. [32] configured a host server system with FPGA-based accelerators for various QKD processing tasks. Yang et al. [31] focused on parallelizing the IR phase for continuous-variable QKD.

A recent review, by Li and Pang [16], has emphasized the almost mandatory choice of the FPGA for QKD applications due to its advantages in power consumption [23] and design productivity, which are key features for critical applications, such as for satellite quantum communication (CubeSat missions) [21]. It also highlighted the benefits of using HLS to design and configure accelerators. Moving from the FPGA to a system-on-chip (SoC) architecture, a recent work [25] presented a hardware- and software-integrated architecture suitable for practical QKD and quantum random number generation schemes. This architecture optimally distributes time-related and management tasks between the FPGA and the CPU.

While previous designs focused on classical postprocessing engines for specific QKD protocols, our proposal aims to create a complete suite of algorithms compatible with multiple DV QKD protocols, implemented as a stand-alone and isolated system with a parallel data storage and processing framework. In the following sections, we provide a detailed design implementation of this proposed architecture.

III. SYSTEM ARCHITECTURE AND DESIGN METHODOLOGY

We propose an FPGA-based flexible high-throughput key distillation framework that supports multiple QKD protocols. This generalized key distillation framework is designed to adapt to various QKD protocols. The KDE performs all the postprocessing tasks and provides the final key to the encryption application. To ensure effective implementation, the entire postprocessing flow, as illustrated in Fig. 1, can be executed in three distinct phases: the preparatory phase, the data acquisition phase, and the reconciliation phase.

The preparatory phase, as described in more detail in Section III-C, involves alignment and measurement processes to establish a low-loss and low-error quantum channel. The software components required for the preparatory phase are specific to the particular QKD protocol being used. In this work, we have adopted the HLS framework to perform tasks that require modifications based on the QKD protocol and to create a unified integration platform for QKD postprocessing. Specific methods, such as clock synchronization and measurement alignment, are closely tied to the protocol implementation specifications. We provide further details about this platform in Section III-B.

The data acquisition phase (see Section III-D) involves the gathering and transformation of the detected raw key vector before proceeding to the reconciliation phase. The gathering and time-stamping techniques used are generic to any QKD protocol. On the other hand, data-sifting methods depend on the specific QKD protocol and its implementation.

The reconciliation phase, as described in Section III-E, encompasses error estimation, correction, verification, and PA modules, all of which are independent of the chosen QKD protocol. It is worth noting that error correction and PA techniques are computationally intensive. To address this, our hardware design efficiently employs the MapReduce programming model to further optimize throughput. This approach strikes a balance between hardware utilization and

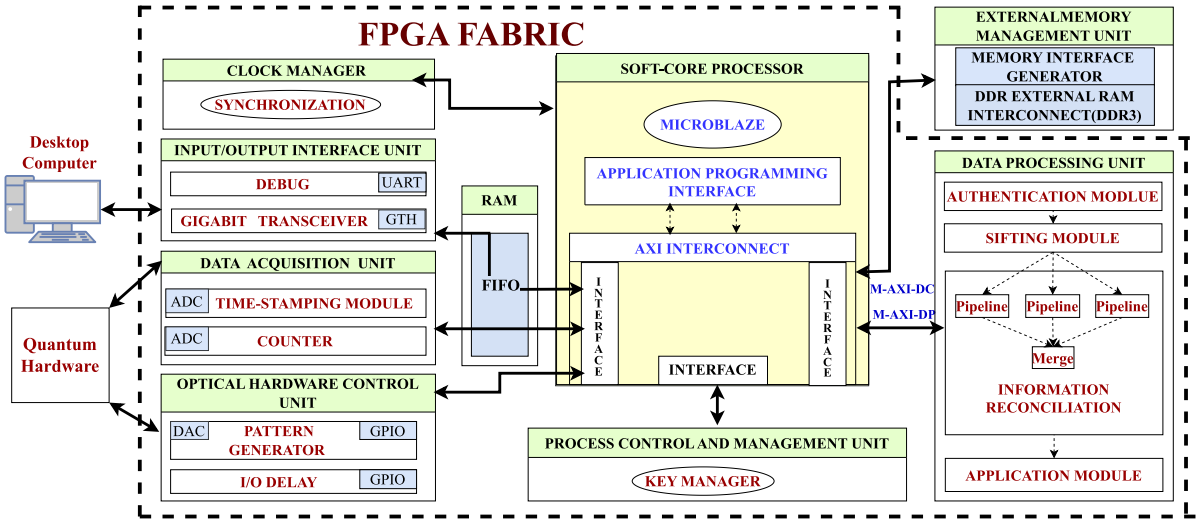

FIGURE 2. FPGA block design for the classical postprocessing engine describing all the modules and their interconnects.

bulky data processing. The MapReduce data storage and processing framework also assist in distributing complex data processing tasks across the limited computing resources of the FPGA.

A. FPGA-BASED PROCESSOR DESIGN

In this article, a single FPGA is used to perform all the control and data processing for QKD. This FPGA fabric is designed with a soft-core-processor-based SoC architecture. Process control, task scheduling, and interface for data processing modules are defined as a software development toolkit (SDK) application program interface (API)-based software application. Makni et al. [17] previously highlighted the advantages of an FPGA-based SoC architecture. Fig. 2 illustrates the high-level architecture, consisting of the control unit, data processing unit, and memory management unit. The soft-core processor, along with the data processing and control modules, is implemented on the FPGA's programmable logic. The board is equipped with 2 GB of DDR3 SODIMM memory, which is used for storing data required during processing. MicroBlaze, a soft processor core designed for Xilinx FPGAs, is used in this architecture. It is integrated with an AXI interconnect peripheral bus for system-memory-mapped transactions, offering server–client capability. Each data processing unit module interfaces with the AXI data port of MicroBlaze for communication, while DDR3 is interfaced with MicroBlaze over AXI instruction cache and data cache ports. Further details regarding the implementation of individual data processing, synchronization, and authentication modules are provided in Sections III-C–III-F. The control and data processing algorithms are implemented as custom-developed intellectual property (IP) cores, each serving a specific function. The software application defines a flow of execution of each custom hardware accelerator.

B. MULTIPROTOCOL QKD SUPPORT

Our design approach aims to create an efficient and configurable control and processing framework that can be seamlessly integrated into the design solution for any QKD protocol implementation, regardless of the underlying technology. This generic framework, tailored for programmable hardware, leverages HLS technology. HLS facilitates reconfigurability and the transformation of high-level algorithmic descriptions into RTL models, making it accessible to individuals without extensive HDL development experience [13].

In our framework, we embrace modularity by incorporating independent IP cores for each control and data processing task. Control modules can be developed by integrating software drivers for optoelectronic components commonly used in QKD experiments. These software drivers are often available as open-source libraries in high-level programming languages such as C/C++, making them directly compatible with our design through HLS. These drivers play a vital role in configuring parameters like variable attenuation, interference visibility, modulator bias, state of polarization, and more, all of which contribute to the establishment of a secure quantum channel. In addition, our design accommodates libraries for mathematical and computational utility functions [24].

The Phase I modules, or preparatory modules, as described in Section III-C, are designed and tested using a QKD simulator. Subsequently, they are refined to comply with the synthesizable subset, and a test bench is developed to ensure that their functionality remains intact. Optimization directives and data types are added to enhance performance. During synthesis, C functions translate into RTL blocks, function arguments become RTL I/O, and arrays translate into memory elements (e.g., RAM, ROM, or FIFO). To optimize hardware resource utilization, HLS offers user directives. Pipelining directives are integrated to meet timing constraints [35]. In

addition, we employ arbitrary precision data types in our design [1].

C. PREPARATORY PHASE

The operation of the control unit primarily involves configuring the QKD protocol and relies heavily on software drivers of optoelectronic components. The integration of this functionality has been given in detail in [24]. Furthermore, the remaining components of the current KDE can be reconfigured to accommodate various QKD protocols with minimal effort. Each submodule within the system is designed as a separate proprietary library (IP core), making it readily reusable for different QKD protocol configurations. Consequently, the switching time between different protocols is reduced, simplifying the process of building interoperable systems. These design considerations enhance the flexibility and adaptability of the KDE to support a range of QKD protocols.

1) SYNCHRONIZATION AND CLASSICAL CHANNEL COMMUNICATION

Precise timing information recording the launch of a quantum state from Alice and its arrival at Bob is crucial for effective key sifting. To achieve time synchronization, we have adopted the commercial White Rabbit Lite Embedded node (WR-LEN), as part of our system architecture. WR-LEN is a versatile synchronization solution capable of accommodating various classical communication protocols [19]. WR-LEN leverages two essential technologies: Synchronous Ethernet (SyncE) and the enhanced precision time protocol (PTP). SyncE ensures frequency matching, while PTP enables precise offset adjustments of the clock. This synchronization is facilitated over a single channel, which serves a dual purpose—providing both synchronization and a means for exchanging information required for classical postprocessing.

For high-speed communication, which occurs at gigabit-per-second rates, we have utilized the Aurora 8B/10B protocol. Developed by Xilinx, this protocol is designed for point-to-point serial links and serves as a robust link layer communication protocol. An important feature of the Aurora 8B/10B protocol is that it is an open standard, making it available for implementation by anyone. In our implementation, we have employed this protocol for data transmission over the classical channel. However, it is worth noting that our proposed system is flexible and supports integration with any standard communication protocol.

D. QKD DATA ACQUISITION

In a QKD system, data are collected by Alice and Bob using dedicated hardware components. The key information is derived from the measurement device by recording the time stamps. This time-stamping capability is essential for each side to implement a full-stack QKD postprocessing system. In a real-world scenario, a fully integrated hardware KDE with an embedded operating system would handle these tasks. However, in our implementation, two desktop computers are used on each side to record the experimental data and also serve as command-and-control systems for the FPGAs. The hardware and firmware used for time stamping and measurement are based on open-source components that have been developed and utilized in other QKD experiments. To record the time-of-flight measurement of the single photons detected by a single photon detector (SPD), time-to-digital converters (TDCs) are employed. The SPD outputs a detected photon as a nuclear instrumentation module pulse, often referred to as a "click," which is then read by the FPGA. A multichannel tapped delay line TDC was developed based on a reference design provided by Adamic and Trost [38]. This TDC logic is integrated into an IP core for the FPGA interface. Furthermore, the design framework includes interfaces to and from the optical setup using digital-to-analog converters (DACs) and analog-to-digital converters (ADCs). ADCs and DACs are crucial for controlling the optical components, e.g., ADC is used to drive the coherent laser, and the DAC is used for amplitude and phase modulation. The specific details of the ADC and DAC implementations are beyond the scope of this article.

1) SIFTING

After the quantum transmission and measurement, Alice and Bob utilize the classical channel to derive a secret key. The alignment step involves procedures to synchronize the detection times (time stamps) in Bob's reference frame with Alice's key bits in her reference frame. This process can be extensive in terms of effort and transmitted information. The sliding window protocol is used for alignment, and the autocorrelation coefficient is used to confirm the alignment.

The basis sifting procedure filters out inconclusive or incompatible measurement results at Bob's end compared to Alice's preparation. Depending on the implemented QKD protocol, the required information exchange (e.g., measurement basis) might need to be bidirectional, as in standard BB84, or unidirectional, as in the distributed-phase-reference (DPR) protocol. As a result of the sifting procedure, both Alice and Bob have a set of n_{sift} elements.

In the BB84 protocol, Alice stores two bits to describe the prepared state, the key bit value, and the basis choice. Bob also stores two-bit information about the measurement choice and measurement outcome, along with the time stamp of the detection. After the alignment step, Bob announces one bit describing the measurement choice for each detection, and Alice responds with one bit containing the XOR value between Bob's and her measurement choice. This method is chosen to reduce communication overhead in the classical channel. Therefore, the communication rate for basis sifting in the BB84 protocol is calculated as $m_{BS}^{BB84} = 2 \times n_Q$ bits, where n_Q is the number of measurement outcomes Bob has recorded.

In the COW protocol, Alice encodes the key information in two consecutive time bins. Alice's n_{sift} elements contain two bits each, describing the prepared state and indicating

whether the state is a signal or decoy. Bob's n_{sift} elements contain two-bit information: one holds the decoded key bit and the other indicates which detector clicked (data line or monitoring detector). Bob also records the time stamp of the detection. For each detection, Bob announces the $\lfloor \text{time stamp}/2 \rfloor$ information during the alignment phase and one bit describing the detector click during the basis sifting phase. In this case, no response is required from Alice to sift out incompatible detection. Therefore, the communication rate for basis sifting in the COW protocol is $m_{\text{BS}}^{\text{COW}} = n_Q$ bits, where n_Q is the number of measurement outcomes Bob has recorded.

E. RECONCILIATION PHASE

IR is a critical phase in every QKD system, aiming to generate an identical key between the sender and the receiver through error correction codes (ECCs). Unless the error in the sifted key is below a certain threshold, it cannot be used to derive a secure secret key. The process begins with key sifting of the raw key obtained through quantum transmission. The error introduced during quantum transmission is then estimated. This error information guides the selection of ECC parameters, as explained in Section III-E3. In this section, we introduce a novel approach to effectively implement a computationally intensive part of the QKD postprocessing stack. We integrate error estimation, correction, verification, and PA into a MapReduce framework, demonstrating the computational advantages of this approach.

1) MAPREDUCE-BASED HARDWARE ACCELERATOR FOR RECONCILIATION

Hadoop is an open-source ecosystem that enables the storage, processing, and analysis of large volumes of data. MapReduce is a crucial processing component of the Hadoop framework. The proposed design framework leverages the MapReduce programming model to efficiently store and process large chunks of sifted key vectors obtained from quantum measurement hardware. This is accomplished at a rate faster than the FPGA's clock. The resulting bottlenecks are overcome by employing new optimization strategies to buffer data and pipeline the processing of multiple blocks of sifted key vectors simultaneously. The FPGA's onboard block RAM (BRAM) is organized into distinct memory banks. Each bank buffers data to only one computational node (a map instance) via a unique address and an independent data bus. This allows multiple replicated pipelines to execute in parallel and optimally utilize the FPGA's hardware resources. The implementation is done taking into consideration the hardware resource limitation, the operating frequency, and the throughput of the pipeline, based on which the best parameter configuration is recommended to improve pipeline efficiency.

The scheduler, based on MicroBlaze, invokes the MapReduce framework. Running at a clock frequency of 100 MHz, it operates a C++ SDK-based application that controls and interconnects all data processing and control modules of the KDE. The execution of the MapReduce hardware accelerator consists of two stages: map and reduce. In the map function (first step), subtasks, namely, splitting and mapping, are performed. First, a splitting function divides the input stored in onboard synchronous DRAM (SDRAM) (DDR3) into smaller blocks based on a user-defined block size. The map function then takes these smaller blocks as input and performs error correction and error verification (EV) on each subblock.

The chosen block size for PA is 10^6 bits or more to mitigate the finite-key-security effects [3] of the QKD protocol. Owing to the resource-intensive nature of ECC for larger block sizes, the design approach chosen involves multiple iterations of ECC with smaller block sizes. The output of the mapper is temporarily stored in block RAM and combined and sent to the PA module, which acts as the reducer. After the PA step, the final key is written back to DDR3 memory. The data flow across these stages is depicted in Fig. 3, and Algorithm 1 outlines the implementation flow.

Algorithm 1: Combined Error Correction and PA Process Using the MapReduce Framework.

```
Input: Sifted_Key_Vector, QBER, Security Parameter
  ρ
Output: Final key
  Initialisation: binary m ∗ n parity-check matrix H_{m∗n}
  /* Stage 1: the map stage*/
  class Mapper
  Method Map (Sifted_Key_Vector, LDPC Instance)
  for each block of Sifted_Key_Vector do
    Error_correction (Sifted_Key_Vector, Blocksize)
    if BER = 0 then
      Emit (Error_Corrected_Key)
    end if
  end for
  for All blocks do
    Error_verification
    if (Result=success) then
      Emit (Error_Corrected_Key, Result)
    end if
  end for
  class Combiner
    Method Combine (Error_Corrected_Key, LDPC
     Instance)
     Emit (Result)
     return Combined_Corrected_Key
    /* Stage 2: the reduce stage*/
  class Reducer
    Method Reduce (Combined_Corrected_Key,
     Security Parameter)
     Privacyamplification()
     Emit (Result)
     return FinalKey
```

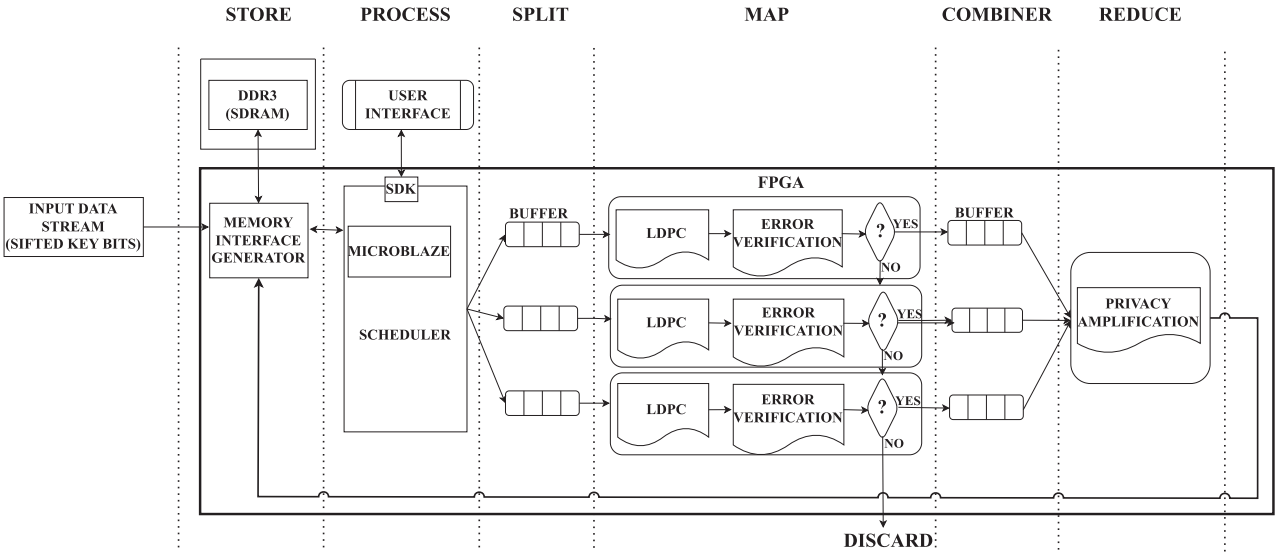

FIGURE 3. MapReduce programming-model-based framework for IR.

The number of mapper instances declared in the design depends on the input block size and the required output block size. In our case, the design is tested with two, three, and four instances of the mapper. The ECC runs a single iteration over a block size of 8192 bits. The input size for the PA reducer module is 1 007 616 bits, so for three instances of the mapper, 41 iterations ($1\,007\,616 = 8192 \times 3 \times 41$) of each instance are required to generate the desired output size.

Increasing the number of mapper instances can reduce the number of iterations, thereby decreasing the time complexity of the implementation, up to a point. This scalability potential is demonstrated in Fig. 7. By incorporating a MapReduce framework into the KDE, a high throughput of 400 kb/s is achieved for a block size of 10^6 compared to 40 kb/s without MapReduce. Further optimization and more powerful hardware could potentially scale the design to seven instances of the mapper, enabling PA on larger block sizes with a substantial increase in throughput. Our design model draws inspiration from the implementation of Neshatpour et al. [20].

2) PARAMETER ESTIMATION

After the sifting process, Alice needs to estimate the approximate lower limit of the error introduced during the transmission of quantum states. This estimation is determined by comparing a randomly sampled subset of the sifted key vector, which is shared by Bob. The inequality described in (1) is proved using Chernoff–Hoeffding-type bounds and is dependent on the sample size. Based on the deduced error, the key distillation process is either aborted or continued

$$\text{Error rate of sampled subset} \approx \text{Error rate of remaining bits} + \Delta. \quad (1)$$

The approximate error rate is estimated as $\sum_{n=1}^{r} \frac{1}{N}$, where r represents the number of errored bits received, and N is the total number of exposed bits. This value is captured as the quantum bit error rate (QBER) of the quantum channel.

In the proposed design, the random sampling of exposed bits and the QBER calculation is implemented on the MicroBlaze processing system. This implementation involves modulo 2 additions and a single 32-bit division operation. The QBER estimate plays a crucial role in determining whether a secure secret key can be extracted through further processing. If the QBER exceeds a predefined threshold (defined for each protocol, considering errors due to device and measurement imperfections as well as the amount of information that can potentially be leaked to an all-powerful quantum adversary through the devices or the channel), the iteration is aborted, and the derived key is discarded.

3) LOW-DENSITY PARITY-CHECK (LDPC) ERROR CORRECTION

LDPC codes have been extensively researched as forward ECCs for QKD systems [10], [11], [18]. In the proposed system, LDPC codes are implemented as an IP core using the HLS design flow described in [24]. The technique used to construct the LDPC parity check matrix is known as protograph code construction. Protograph codes are created by expanding a base protograph. The resulting LDPC parity check matrix is a combination of submatrices. In the proposed system, an irregular parity check matrix is constructed and populated with values from the Galois field of two elements [GF(2)]. Index positions of the elements of the matrix containing a value of one are stored in the local memory of MicroBlaze. The row and column indexes are used to construct a Tanner graph at the decoder. A soft-decision

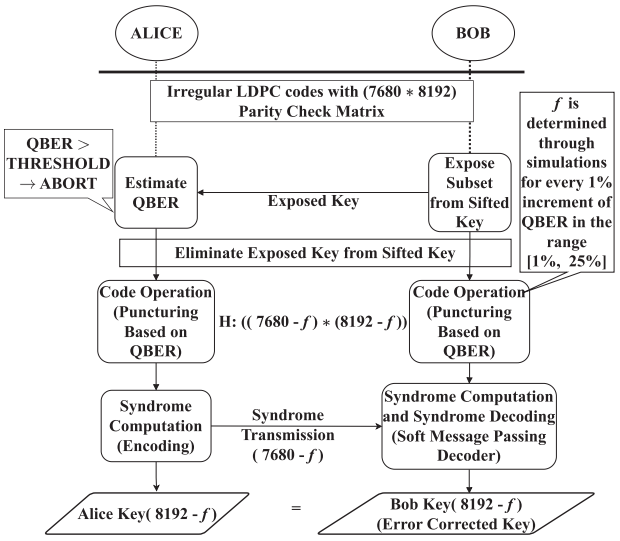

FIGURE 4. Flow diagram of the LDPC codes.

message-passing decoder propagates the belief deduced for each syndrome bit iteratively and decodes the sifted key bit value. The encoding and decoding technique used in the proposed design is a standard technique described in the literature [10], [11]. The QBER plays a crucial role in selecting an efficient LDPC code for the quantum communication system. The goal is to choose an LDPC code dimension such that it maximizes the achievable rate while keeping the error rate below a certain threshold. This process involves determining the LDPC code threshold using the measured QBER.

The approach described in the work by Andrew et al. [22] is followed for deriving the LDPC codes in the proposed design. This reference likely provides detailed insights into constructing LDPC codes tailored for quantum communication systems. Fig. 4 illustrates the algorithmic flow of the LDPC code implementation, which involves steps such as code construction, threshold determination, and LDPC code adaptation based on the measured QBER. The proposed design incorporates rate-adaptive LDPC codes, which are designed to adapt to variations in the QBER. These rate-adaptive LDPC codes are constructed based on the principles outlined in the work by Elkouss et al. [11]. Here is how the rate-adaptive LDPC codes work in the proposed design: The dimension of the parity check matrix is taken as 7680×8192. As described in the PE module, a subset of the key (the exposed key) is shared to estimate the QBER. Based on the estimated QBER, the size of the syndrome vector (denoted as $(7680 - f)$, where f is the reduction factor) is determined for error reconciliation. This adaptation is achieved using a technique called "puncturing," which is further elaborated in the work by Elkouss et al. [11]. The rate-adaptive LDPC codes are designed to optimize the code rate based on the QBER. This optimization involves shortening message bits and puncturing parity bits to maximize the channel capacity. The reduction factor f is inversely proportional to the QBER, and it helps in adjusting the code dimension accordingly.

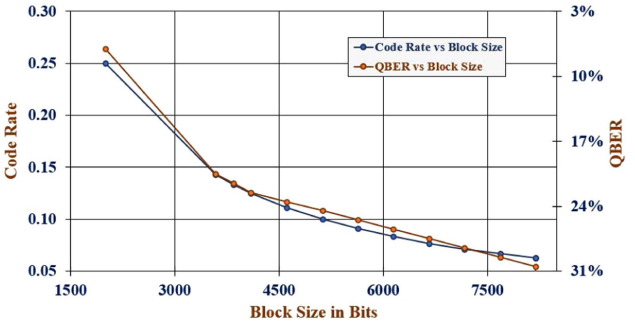

FIGURE 5. Plot recording the outcome of the Monte Carlo simulations for QBER from the range of {7–30%} resulting in reduced block sizes [$(8192 - f)$, in bits] with corresponding code rate as the primary axis.

Through Monte Carlo simulations, the implemented LDPC codes have been evaluated to be capable of correcting up to 25% block errors, with a maximum tested input size of 10^9 bits (1 GB) and a maximum of 50 decoder iterations. The primary objective of rate-adaptive LDPC codes in this design is to take a full-capacity LDPC code and dynamically adjust the code rate based on the QBER. This is done to optimize the code's performance. Fig. 5 provides an illustration of this rate adaptation process.

4) ERROR VERIFICATION

EV is an essential step followed by error correction to ensure the integrity of the decoded key vector. There still remains a finite probability that the sifted key vector shared between Alice and Bob may contain errors necessitating an additional step of EV. To implement this integrity check, we employ the universal hash function technique referred to in [7]. The error-corrected key, along with a preshared key, serves as input to the hash function. This preshared key can be generated either from the FPGA-based true random number generator (TRNG) module or as part of the QKD-generated key from the previous iteration.

In our implementation, the block size of the input is large. To ensure information-theoretic security, a preprocessing procedure, as shown in Algorithm 2, is employed to extract 128 bits from a large input block, as required by the architecture of the Poly1305 algorithm [5]. This output, associated with each chunk $[C_1, C_2, \ldots, C_t]$, along with the 256-bit preshared key, is used as input to Poly1305, which generates a 128-bit tag. The tags generated by both Alice and Bob are compared at Bob's end. Bob then communicates the EV flag to Alice. If any of the tags are erroneous, the specific chunk will be discarded.

5) PRIVACY AMPLIFICATION

PA is one of the most vital postprocessing steps. Within postprocessing, the transmission of the key vector from Alice to Bob requires mandatory classical communication for sifting, error detection, correction, and verification, which can lead to information leakage. Information leakage can also occur due to eavesdropping attacks on the quantum channel.

Algorithm 2: EV Algorithm.

Input: Error-corrected key of size N.
Output: t Tags each 128 bits.

$t \leftarrow$ Number of Tags
$k \leftarrow 128$
$Eck \leftarrow Error\_Corrected\_Key[b_1, \ldots, b_N]$
$[c_1, c_2, \ldots, c_t] \leftarrow Split(Eck)$
$Size \leftarrow \frac{N}{t}$
while $Size > k$ bits **do**
  for $i \leftarrow 1, t \triangleright$ Iterate over each chunk **do**
    $D_{2i-1}, D_{2i} \leftarrow Split(c_i) \triangleright D_i, size = size/2$
    $C_i \leftarrow XOR(D_{2i-1}, D_{2i})$
  end for
  $Size \leftarrow \frac{Size}{2}$
end while
$Tag_{Alice,i} \leftarrow$ **Poly1305**$(C_{i,Alice}\ random\ seed))$
$Tag_{Bob,i} \leftarrow$ **Poly1305**$(C_{i,Bob}\ random\ seed))$
Send $Tag_{Alice,i}$ to Bob
Verify $Tag_{Alice,i} == Tag_{Bob,i}$

Therefore, to ensure a strict level of security, it is necessary to eliminate this amount of leaked information through a PA step. In PA, the partially secure key is transformed into a completely secure key through public discussion. This is achieved by computing a publicly chosen two-universal hash function of the key vector. We use Toeplitz hashing to alter the error-corrected key vector by an adjustable compression ratio, guaranteeing the security of the remaining secret key bits. The compression ratio is derived from the estimate of the information leakage [39].

A very simple implementation idea for a large-scaled PA scheme is to directly perform multiplication operations between the weak key W and the Toeplitz matrix $T(A)$, resulting in a computational complexity of $O(n^2)$. We reduce the complexity to $O(n \log n)$ by using fast Fourier transform instead of normal multiplication [8]. W is taken as the error-corrected key, which is of length n. Both Alice and Bob decide on a final secure key length, denoted as r, with rigorous statistical analysis. Furthermore, Alice and Bob publicly discuss a random seed of length $(n-1)$ to construct the universal hash function. The random seed is generated using an FPGA-based TRNG at Alice. Our PA scheme for KDE mainly consists of three steps, i.e., splitting with shuffling, sub-PA, and secure key merging, as described in detail in [8].

F. AUTHENTICATION OF QKD SYSTEMS WITH PUF

The QKD system requires authentication to ensure that the received data are from the intended device and to verify that there have been no modifications in the data received through a public channel. Numerous techniques have been introduced to authenticate the public channel; however, the state-of-the-art authentication mechanisms are based on classical cryptographic primitives and are only loosely coupled with QKD hardware electronics.

In our article, we have implemented an arbiter PUF-enabled mutual device authentication protocol [2]. PUFs represent a promising security primitive that guarantees unclonability, uniqueness, and tamper-evident properties. Our goal is to employ PUFs for QKD device identification and data integrity maintenance, effectively addressing the security requirements of QKD systems and providing a means to verify the integrity of both the device and the transmitted message.

Our approach involves implementing an Arbiter PUF-enabled mutual device authentication protocol to establish authenticity and integrity among the devices participating in the quantum key negotiation process. The execution of PUF-based authentication occurs at two levels. In the first level, during device enrollment, the device's unique identity is presented as a challenge to the PUF, and the response is recorded and extracted. This information exchange takes place exclusively through a secure channel and is a one-time enrollment process. During authentication, the QKD system necessitates a public channel for communication. Therefore, each QKD device, in conjunction with KDE, is equipped with a PUF, where the response depends on its unique physical characteristics. If an attacker attempts to tamper with the hardware or software associated with the QKD device, any deviation from the device's expected behavior is detected, indicating a potential breach of privacy. Consequently, under reasonable assumptions, PUFs are more secure and robust against device tampering and man-in-the-middle attacks over the classical channel for QKD, given their tight integration with the hardware governing the QKD system.

G. ADVANCED ENCRYPTION STANDARD (AES) ENCRYPTION

We have implemented an efficient hardware architecture for the AES-256. The AES algorithm, as defined by the National Institute of Standards and Technology of the United States, has gained widespread acceptance. Our implementation achieves a throughput exceeding 10 Gb/s for both encryption and decryption processes when utilizing the Xilinx Virtex Family device XC7VX485T. The hardware design approach relies on precalculated lookup tables (LUTs) and parallelly executable instances, resulting in a less complex architecture that offers high throughput and low latency. The speedup is achieved by running the key expansion module independently of the AES rounds. AES with a larger key size is considered resistant to attacks by powerful quantum adversaries, making it an ideal choice for applications utilizing QKD-generated keys. Consequently, AES is employed for secure image, video, or message encryption.

IV. EXPERIMENTAL SETUP

To validate and analyze the performance of the KDE designed and implemented, data were collected from quantum experiments conducted at the Physics Research Laboratory

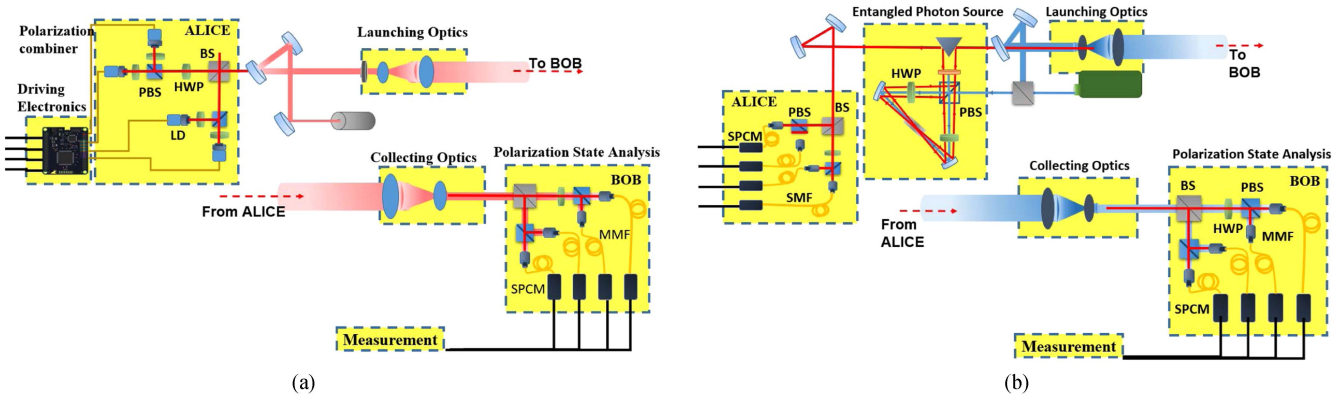

FIGURE 6. Experimental scheme of (a) BB84 protocol and (b) BBM92 protocol, which includes both optical and electronic arrangements. EPS: entangled photon source, FM: flip mirror, PM: prism mirror, M: mirror, F: filter, FC: fiber coupler, BS: balanced beam splitter, PBS: polarization beam splitter, DPBS: dual-wavelength PBS, HWP: half-wave plate, SMF: single-mode fiber, MMF: multimode fiber, SPCM: single-photon counting modules, PPKTP: Periodically poled potassium titanyl phosphate. LD: laser driver, BD: beam dumper.

in Ahmedabad, India, where the polarization-based BB84 QKD protocol [6] and the BBM92 QKD protocol were implemented. Details of the experimental setups are depicted in Fig. 6. In the BB84 setup shown in Fig. 6(a), weak coherent pulses are generated by using a variable optical attenuator at the output of a pulsed laser with a repetition rate of 80 MHz. The encoded state is then propagated in a free-space lossy medium with channel transmissivity estimated at 70%. At Bob, there is a polarization-based detection setup consisting of a balanced beam splitter (BS; a passive random basis selector) with a polarizing beam splitter (PBS) on the reflected arm (measurement in H, V), and a combination of the half-wave plate with PBS (measurement in D, A) on the transmitted arm. At the output ports of the PBS, the photons are detected by fiber-coupled avalanche photodiodes (Excelitas *SPCM AQRH-14-FC*). The BBM92 protocol is just the entangled version of the BB84 protocol. The BBM92 protocol involves pairs of entangled photons. In this protocol, a common sender prepares the entangled photon source and sends it to Alice and Bob through the quantum channel. In Fig. 6(b), the polarization Sagnac interferometer is used to prepare entangled photons. In this interferometry, a diagonally polarized 405-nm continuous-wave laser with an output power of $\sim 5\ mW$ is used to pump a 30-mm-long Type-0 periodically poled potassium titanyl phosphate (PPKTP) crystal of period 3.425 μm. A lens L1 of focal length 400 mm is used to focus the pump beam on the crystal to generate entangled photons. The horizontally polarized pump beam is transmitted through the dual-wavelength PBS (DPBS) in a clockwise direction, and vertically polarized light is reflected through the DPBS in a counterclockwise direction. Since both the clockwise and counterclockwise pump beams follow the same path but in opposite directions inside the Sagnac interferometer and the *Type-0* PPKTP crystal is placed symmetric to the DPBS, the implemented scheme is robust against any optical path changes to produce spontaneous parametric downconversion photons in orthogonal polarizations with ultrastable phase. At the output of the Sagnac interferometer, a filter is used to block the pump beam while transmitting the entangled photons. A prism mirror is used to separate the entangled photon pairs. One photon is sent to Alice, and another photon is sent to Bob (each has a detection setup) through launching optics. The detection setup is the same as BB84. The output from the SPD is fed into electronics for recording the counts per integration time, and these data are then used to derive the sifted key vector. The sifted key vector is the input to the KDE.

TABLE 1 Performance Metrics of Each KDE Module Implemented on Hardware Including Both Alice's and Bob's Design

Module	Input Block size (bits)	Key Rate	Time (ns)
Authentication	128	1 Gb/s	140
Parameter Estimation	10^6	70 Mb/s	18×10^6
LDPC Encoder	10^6	484 kb/s	2.1×10^9
LDPC Decoder	10^6	80 kb/s	12.2×10^9
Error verification	10^6	1 Gb/s	140
Privacy Amplification	10^6	510 kb/s	1.89×10^9
AES Encryption	1024	10 Gb/s	100
AES Decryption	1024	9 Gb/s	100

TABLE 2 Utilization Report of the Alice (Transmitter) and Bob (Receiver) KDE Designs Without MapReduce Framework (WOMF), and With MapReduce Framework (WMF; Three Parallel Instances)

ENTITY	DESIGN	LUT%	FF (Flip-flop%)	BRAM (%)	Total Power (W)
Alice	WOMF	9.14	4.50	23.6	3.286
	WMF	9.71	4.91	38.7	3.525
Bob	WOMF	11.99	5.56	32.2	3.837
	WMF	18.46	8.26	61.2	5.131

V. IMPLEMENTATION AND ANALYSIS

The developed KDE hardware design is tested and verified using quantum bits (raw key vector) collected from the

TABLE 3 MapReduce-Based IR; Implementation Validation on Intel CPU Core for {1,3,4} Number of Parallel Instances Using Python's Inbuilt Functional Programming Feature, Running the Same Algorithms

	CPU			FPGA		
Input Size (bits)	10,07,616	10,07,616	9,83,040	10,07,616	10,07,616	9,83,040
MapReduce parameters (No of Instances)	1	3	4	1	3	4
Implementation status	Python simulation	Python simulation	Python simulation	Hardware implemented	Hardware implemented	Hardware implemented
Average no. of iterations per instance	123	41	30	123	41	30
Processor and Clock speed	Intel(R) Core i7 CPU @ 2.60 GHz (max eight cores)	Intel(R) Core i7 CPU @ 2.60 GHz (max eight cores)	Intel(R) Core i7 CPU @ 2.60 GHz (max eight cores)	MicroBlaze @ 100 Mhz	MicroBlaze @ 100 Mhz	MicroBlaze @ 100 Mhz
Total Latency	41.24 s	14.47 s	10.97 s	10.37 s	3.52 s	2.66 s

quantum experiments implementing the BB84 protocol [see Fig. 6(a)], the BBM92 protocol [see Fig. 6(b)], and the COW protocol. The QKD control and postprocessing hardware designs are implemented on a Virtex-7 VX485T Xilinx FPGA. Table 2 provides a record of the area and utilization parameters for the hardware implementation of the KDE design, both with MapReduce framework (WMF) and without MapReduce framework (WOMF) in the parallel architecture. Utilization summaries of the transmitter (Alice) and receiver (Bob) designs with three parallel instances are captured in WMF.

Table 1 records the implementation efficiency and performance of each module in KDE design in terms of latency and key rate. The execution time is determined by considering both the number of clock cycles required to process the module and the clock period, determined by the board's clock frequency. The key rate is derived from the total input block size divided by the time and the maximum delay path.

In terms of resource utilization, an increase in the utilization of logic cells and block RAM units is observed in Table 2, only on the receiver side with the MapReduce framework. This is primarily due to the computationally intensive LDPC decoder. This can be reduced by adopting optimized decoder implementations. The experiment is conducted for each protocol, collecting and processing 10 MB of raw key bits. The variations in QBER recorded in the experiments are used to validate the effectiveness of rate-adaptive LDPC codes with a threshold error correction capacity of 25% at 90% efficiency. Table 3 summarizes the average performance of both the FPGA implementation and CPU core implementation. Table 4 presents the performance of our implemented design across various QKD experimental settings and compares performance at different QBER values. It is worth noting that while increasing the number of parallel instances, there is a significant increase in key rate and subsequently a decrease in utilization time due to the parallel postprocessing system implementation. Performancewise, Fig. 7 illustrates that as QBER increases, execution time or latency also increases, inversely affecting the secret key extraction rate. However, leveraging more parallel instances in the design can reduce latency, thereby increasing the key rate.

TABLE 4 Implementation Results for Multiple Parallel Instances of the Mapper

Protocol	Input Block size (bits)	QBER	No of parallel Instances	Key Rate (Kbps)	Time (sec)
COW	10,07,616	25%	1	39	25.7
	10,07,616		3	116.7	8.6
	9,83,040		4	157.9	6.2
BB84	10,07,616	2.63%	1	96.8	10.37
	10,07,616		3	285.1	3.52
	9,83,040		4	368.1	2.66
COW	10,07,616	21.40%	1	45.4	22.13
	10,07,616		3	128.7	7.8
	9,83,040		4	175.8	5.57
BBM92	10,07,616	9.03%	1	70.6	14.22
	10,07,616		3	207.4	4.84
	9,83,040		4	273.5	3.58

VI. FUTURE WORK AND OPEN CHALLENGES

Our experimental studies provide valuable insights into complexity estimation, computing resource requirements, and bandwidth needs for establishing a fully integrated QKD system. These findings lay the groundwork for the future development of large-scale commercial QKD systems. The proposed work utilizes a generic MapReduce design framework, which can be extended to build interconnected FPGA-based systems for large-scale applications. Future work may explore novel approaches, such as viewing FPGAs as individual coprocessors within computing clusters with server–client architecture, offering scalability for QKD systems. Nonetheless, several challenges remain. First, the complexity of the FPGA-based framework for quantum key reconciliation on large datasets requires further optimization for enhanced efficiency and performance. Second, the integration of QKD systems with reconfigurable FPGA

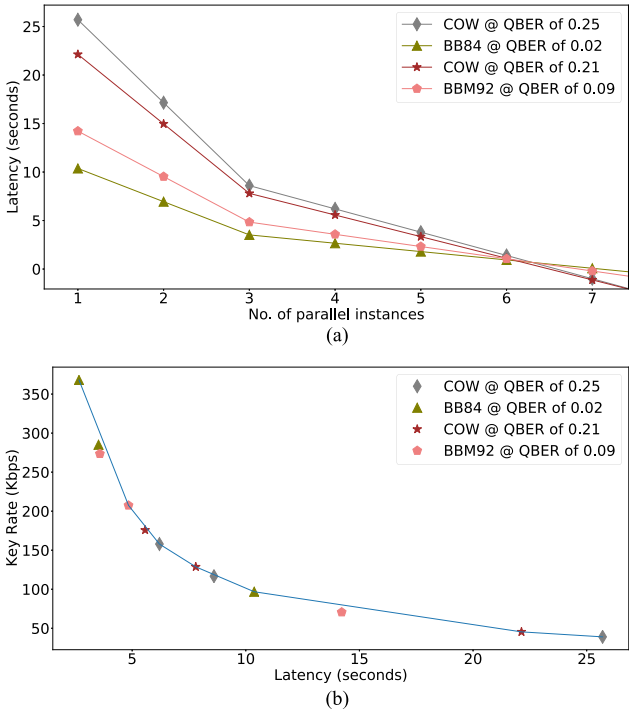

FIGURE 7. Plot of (a) number of parallel instances of mapper in the Hadoop framework versus execution time in seconds and (b) execution time, in seconds, versus key rate in kb/s, for COW, BB84, and BBM92 QKD protocols at varying QBER (measured during the quantum experiment).

accelerators presents challenges in terms of interoperability and reconfigurability. Third, QKD systems are sensitive to clock synchronization and memory management, requiring solutions and optimizations to achieve reasonable performance.

VII. CONCLUSION

In this article, we presented a design that enhances scalability and enables faster key reconciliation for QKD systems by leveraging the FPGA-based MapReduce architecture. We introduced a novel reconfigurable architecture that optimizes resources through the use of reusable blocks. In addition, our research introduced a PUF-based authentication protocol to facilitate mutual device authentication and secure message exchange for QKD devices. Our experimental results demonstrated that the hardware design strikes a balance between resource utilization and throughput. Consequently, implementing MapReduce-based QKD postprocessing functionality directly in hardware emerges as a preferred technique to meet the computational and critical security requirements of commercial QKD systems. An added advantage of FPGA-based key distillation is its capacity to enhance performance and compatibility with various QKD experimental setups. Typically, a QKD KDE utilizes a combination of individual IP core libraries to construct complete postprocessing protocols, including a data encryption module. This approach allows for the creation of a secure multi-QKD protocol-based key distillation hardware with FPGA systems. For those interested, the source codes supporting the findings of this study are available upon reasonable request from the corresponding author. Access to the code can also be requested via the GitHub repository.[1]

ACKNOWLEDGMENT

The authors would like to acknowledge and thank Dr. Jothi Ramalingam and Sarika K Menon for their invaluable assistance and insightful discussions. The authors would also like to acknowledge M. Swathi Mithran for his support and assistance.

REFERENCES

[1] A. Alhamali et al., "FPGA-accelerated Hadoop cluster for deep learning computations," in *Proc. IEEE Int. Conf. Data Mining Workshop*, 2015, pp. 565–574, doi: 10.1109/ICDMW.2015.148.

[2] N. N. Anandakumar, M. S. Hashmi, and M. A. Chaudhary, "Implementation of efficient XOR arbiter PUF on FPGA with enhanced uniqueness and security," *IEEE Access*, vol. 10, pp. 129832–129842, 2022, doi: 10.1109/ACCESS.2022.3228635.

[3] D. Bacco, M. Canale, N. Laurenti, G. Vallone, and P. Villoresi, "Experimental quantum key distribution with finite-key security analysis for noisy channels," *Nature Commun.*, vol. 4, no. 1, pp. 1–8, 2013, doi: 10.1038/ncomms3363.

[4] C. H. Bennett, F. Bessette, G. Brassard, L. Salvail, and J. Smolin, "Experimental quantum cryptography," *J. Cryptol.*, vol. 5, no. 1, pp. 3–28, 1992, doi: 10.1007/BF00191318.

[5] D. J. Bernstein, "The poly1305-AES message-authentication code," in *Proc. Int. Workshop Fast Softw. Encryption*, 2005, pp. 32–49, doi: 10.1007/11502760_3.

[6] A. Biswas, A. Banerji, N. Lal, P. Chandravanshi, R. Kumar, and R. P. Singh, "Quantum key distribution with multiphoton pulses: An advantage," *Opt. Continuum*, vol. 1, no. 1, pp. 68–79, 2022, doi: 10.1364/OPTCON.445727.

[7] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *J. Comput. Syst. Sci.*, vol. 18, no. 2, pp. 143–154, 1979, doi: 10.1016/0022-0000(79)90044-8.

[8] J. Constantin et al., "An FPGA-based 4 Mbps secret key distillation engine for quantum key distribution systems," *J. Signal Process. Syst.*, vol. 86, no. 1, pp. 1–15, 2017, doi: 10.1007/s11265-015-1086-1.

[9] K. Cui, J. Wang, H.-F. Zhang, C.-L. Luo, G. Jin, and T.-Y. Chen, "A real-time design based on FPGA for expeditious error reconciliation in QKD system," *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 1, pp. 184–190, Jan. 2013, doi: 10.1109/TIFS.2012.2228855.

[10] AR Dixon and H. Sato, "High speed and adaptable error correction for megabit/s rate quantum key distribution," *Sci. Rep.*, vol. 4, no. 1, pp. 1–6, 2014, doi: 10.1038/srep07275.

[11] D. Elkouss, J. Martinez-Mateo, and V. Martin, "Information reconciliation for quantum key distribution," 2010, *arXiv:1007.1616*, doi: 10.48550/arXiv.1007.1616.

[12] B. Frhlich et al., "Long-distance quantum key distribution secure against coherent attacks," *Optica*, vol. 4, no. 1, pp. 163–167, 2017, doi: 10.1364/OPTICA.4.000163.

[13] M. Gurel, *A Comparative Study Between RTL and HLS for Image Processing Applications With FPGAs*. San Diego, CA, USA: Univ. California, 2016.

[14] K. Inoue, E. Waks, and Y. Yamamoto, "Differential-phase-shift quantum key distribution using coherent light," *Phys. Rev. A*, vol. 68, no. 2, 2003, Art. no. 022317, doi: 10.1103/PhysRevA.68.022317.

[15] A. Kerckhoffs, "La cryptographie militaire," *J. des Sci. Mil.*, vol. IX, pp. 5–38, Jan. 1883. [Online]. Available: https://petitcolas.net/kerckhoffs/crypto_militaire_2.pdf

[16] H. Li and Y. Pang, "FPGA-accelerated quantum computing emulation and quantum key distillation," *IEEE Micro*, vol. 41, no. 4, pp. 49–57, Jul./Aug. 2021, doi: 10.1109/MM.2021.3085431.

[1] https://github.com/SETSQKD

[17] M. Makni, M. Baklouti, S. Niar, and M. Abid, "Hardware resource estimation for heterogeneous FPGA-based SoCs," in *Proc. Symp. Appl. Comput.*, 2017, pp. 1481–1487, doi: 10.1145/3019612.3019683.

[18] J. Martinez-Mateo, C. Pacher, M. Peev, A. Ciurana, and V. Martin, "Demystifying the information reconciliation protocol cascade," 2014, *arXiv:1407.3257*, doi: 10.48550/arXiv.1407.3257.

[19] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer, "White rabbit: Sub-nanosecond timing distribution over ethernet," in *Proc. Int. Symp. Precis. Clock Synchronization Meas., Control Commun.*, 2009, pp. 1–5, doi: 10.1109/ISPCS.2009.5340196.

[20] K. Neshatpour et al., "Energy-efficient acceleration of MapReduce applications using FPGAs," *J. Parallel Distrib. Comput.*, vol. 119, pp. 1–17, 2018, doi: 10.1016/j.jpdc.2018.02.004.

[21] D. K. L. Oi et al., "CubeSat quantum communications mission," *EPJ Quantum Technol.*, vol. 4, pp. 1–20, 2017, doi: 10.1140/epjqt/s40507-017-0060-1.

[22] A. K. Pradhan, A. Thangaraj, and A. Subramanian, "Construction of near-capacity protograph LDPC code sequences with block-error thresholds," *IEEE Trans. Commun.*, vol. 64, no. 1, pp. 27–37, Jan. 2016, doi: 10.1109/TCOMM.2015.2500234.

[23] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels," in *Proc. IEEE Int. Conf. Embedded Softw. Syst.*, 2019, pp. 1–8, doi: 10.1109/ICESS.2019.8782524.

[24] U. Sisodia, "Using high-level synthesis to migrate open source software algorithms to semiconductor chip designs," in *System Level Flows for SoC Architecture Analysis and Design*. Noida, India: CircuitSutra Technologies, 2020.

[25] A. Stanco et al., "Versatile and concurrent FPGA-based architecture for practical quantum communication systems," *IEEE Trans. Quantum Eng.*, vol. 3, 2022, Art. no. 6000108, doi: 10.1109/TQE.2022.3143997.

[26] D. Stucki, N. Brunner, N. Gisin, V. Scarani, and H. Zbinden, "Fast and simple one-way quantum key distribution," *Appl. Phys. Lett.*, vol. 87, no. 19, 2005, Art. no. 194108, doi: 10.1063/1.2126792.

[27] A. Tanaka et al., "High-speed quantum key distribution system for 1-Mbps real-time key generation," *IEEE J. Quantum Electron.*, vol. 48, no. 4, pp. 542–550, Apr. 2012, doi: 10.1109/JQE.2012.2187327.

[28] N. Walenta et al., "A fast and versatile quantum key distribution system with hardware key distillation and wavelength multiplexing," *New J. Phys.*, vol. 16, no. 1, 2014, Art. no. 013047, doi: 10.1088/1367-2630/16/1/013047.

[29] W. Wang, K. Tamaki, and M. Curty, "Measurement-device-independent quantum key distribution with leaky sources," *Sci. Rep.*, vol. 11, no. 1, pp. 1–11, 2021, doi: 10.1038/s41598-021-81003-2.

[30] F. Xu, X. Ma, Q. Zhang, H.-K. Lo, and J.-W. Pan, "Secure quantum key distribution with realistic devices," *Rev. Modern Phys.*, vol. 92, no. 2, 2020, Art. no. 025002, doi: 10.1103/RevModPhys.92.025002.

[31] S.-S. Yang, Z.-G. Lu, and Y.-M. Li, "High-speed post-processing in continuous-variable quantum key distribution based on FPGA implementation," *J. Lightw. Technol.*, vol. 38, no. 15, pp. 3935–3941, Aug. 2020, doi: 10.1109/JLT.2020.2985408.

[32] Z. Yuan et al., "10-Mb/s quantum key distribution," *J. Lightw. Technol.*, vol. 36, no. 16, pp. 3427–3433, Aug. 2018, doi: 10.1109/JLT.2018.2843136.

[33] H.-F. Zhang et al., "Real-time QKD system based on FPGA," *J. Lightw. Technol.*, vol. 30, no. 20, pp. 3226–3234, Oct. 2012, doi: 10.1109/JLT.2012.2217394.

[34] J. Zhou, B. Liu, and B. Zhao, "A pipeline optimization model for QKD post-processing system," in *Proc. Inf. Commun. Technol.-EurAsia Conf.*, 2014, pp. 472–481, doi: 10.1007/978-3-642-55032-4_48.

[35] M. D. Zwagerman, "High level synthesis, a use case comparison with hardware description language," M.S. thesis, School of Eng., Grand Valley State Univ., Allendale Charter Township, MI, USA, 2015.

[36] C. H. Bennett, G. Brassard, and N. D. Mermin, "Quantum cryptography without Bell's theorem," *Phys. Rev. Lett.*, vol. 68, no. 5, pp. 557–559, 1992, doi: 10.1103/PhysRevLett.68.557.

[37] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in *Proc. IEEE Int. Conf. Comput., Syst., Signal Process.*, 1984, pp. 175–179, doi: 10.1016/j.tcs.2014.05.025.

[38] M. Adamič and A. Trost, "A fast high-resolution time-to-digital converter implemented in a Zynq 7010 SoC," in *Proc. Austrochip Workshop Microelectronics*, 2019, pp. 29–34, doi: 10.1109/Austrochip.2019.00017.

[39] C. H. Bennett, G. Brassard, C. Crépeau, and U. M. Maurer, "Generalized privacy amplification," *IEEE Trans. Inf. Theory*, vol. 41, no. 6, pp. 1915–1923, Nov. 1995, doi: 10.1109/18.476316.

Natarajan Venkatachalam (Member, IEEE) received the M.Sc. degree in applied mathematics and Ph.D. degree in computational sciences from Anna University, Chennai, India, in 2010 and 2015, respectively.

He is currently a Scientist with the Society for Electronic Transactions and Security (SETS), Chennai, and has worked in the cybersecurity area for nearly 15 years. Prior to joining the SETS, he was a Postdoctoral Research Associate with Quantum Engineering Technology, University of Bristol, Bristol, U.K. His research interests include secure quantum communications and experimental quantum key distribution systems, in particular fundamental, and applied postquantum cryptography.

Dr. Venkatachalam is an Active Member of the Cryptology Research Society of India and the Computer Society of India.

Foram P. Shingala received the M.E. degree in computer science and engineering from Madras Institute of Technology, Chennai, India, in 2017.

She is currently a Scientist with the Society for Electronic Transactions and Security, Chennai. She has worked on the proof-of-concept demonstration of coherent one-way quantum key distribution (QKD) system with field-programmable-gate-array-based classical reconciliation algorithms for QKD. Her current research interests include quantum cryptography and computing.

Selvagangai C received the M.E. degree in very large scale integration design from the PSG College of Technology, Coimbatore, India, in 2018.

In 2019, she joined the Society for Electronic Transactions and Security, Chennai, India, as a Research Fellow. Her current research interests include classical reconciliation and key distillation algorithms for quantum key distribution and field-programmable gate array programming.

Hema Priya S received the B.Tech. degree in electronics and communication and the M.Tech. degree in very large scale integration design from Anna University, Chennai, India, in 2015 and 2017, respectively.

She is currently a Project Scientist with the Society for Electronic Transactions and Security, Chennai. Her research interests include field-programmable gate array design and verification, quantum cryptography, and hardware security.

Dillibabu S received the B.E. degree in electronics and communication engineering from Anna University, Chennai, India, in 2005.

He is currently a Scientist with the Society for Electronic Transactions and Security, Chennai. He is also a Ph.D. Research Scholar with the Worcester Polytechnic Institute, Worcester, MA, USA. He has experience in the field of cryptography and hardware security for 16 years. His research interests include differential power analysis from cryptography, side-channel analysis of hardware cryptographic modules, and postquantum cryptography.

Pooja Chandravanshi received the B.Tech. degree in electronics and telecommunication from the Bhilai Institute of Technology (BIT), Durg Chhattisgarh, India, in 2016.

She is currently a Scientist with the Physical Research Laboratory, Ahmedabad, India. Her research interests include electronic circuits and field implementation of quantum key distribution protocols.

Ravindra P. Singh is currently a Senior Professor and the Chair of Atomic, Molecular and Optical Physics Division, Physical Research Laboratory, Ahmedabad, India. His research interests include light scattering, phase singularities of light, nonlinear optics, quantum optics, and quantum information. He is also engaged in experiments on high dimensional entangled states, free space quantum communication, satellite-based quantum key distribution, and quantum radar.