J Sign Process Syst
DOI 10.1007/s11265-015-1086-1

An FPGA-Based 4 Mbps Secret Key Distillation Engine for Quantum Key Distribution Systems

Jeremy Constantin[1] · Raphael Houlmann[2] · Nicholas Preyss[1] · Nino Walenta[3] · Hugo Zbinden[2] · Pascal Junod[4] · Andreas Burg[1]

Received: 25 February 2014 / Revised: 6 August 2015 / Accepted: 6 November 2015

Abstract Quantum key distribution (QKD) enables provably secure communication between two parties over an optical fiber that arguably withstands any form of attack. Besides the need for a suitable physical signalling scheme and the corresponding devices, QKD also requires a secret key distillation protocol. This protocol and the involved signal processing handle the reliable key agreement process over the fragile quantum channel, as well as the necessary post-processing of key bits to avoid leakage of secret key information to an eavesdropper. In this paper we present in detail an implementation of a key distillation engine for a QKD system based on the coherent one-way (COW) protocol. The processing of key bits by the key distillation engine includes agreement on quantum bit detections (sifting), information reconciliation with forward error correction coding, parameter estimation, and privacy amplification over an authenticated channel. We detail the system architecture combining all these processing steps, and discuss the design trade-offs for each individual system module. We also assess the performance and efficiency of our key distillation implementation in terms of throughput, error correction capabilities, and resource utilization. On a single-FPGA (Xilinx Virtex-6 LX240T) platform, the system supports distilled key rates of up to 4 Mbps.

Keywords Quantum cryptography · Quantum key distribution · Secret key distillation · Communication system design

1 Introduction

Modern information societies rely heavily on transferring data in digital form. Symmetric cryptography provides the necessary means to securely exchange data, relying on two fundamental aspects: first, the distribution of shared secret keys; and second, the usage of these secret keys to authenticate or encrypt data. To date, the only information-theoretic provable secure solution for secret key distribution is provided by quantum key distribution (QKD) [2, 3, 7]. In contrast to classical key distribution schemes whose security relies on assumptions on the computational power of a potential eavesdropper, the security of QKD is based on fundamental laws of quantum mechanics to detect and forbid eavesdropping without any further restrictions or assumptions.

Besides the necessity to exchange optical signals on the quantum levels, all QKD protocols require synchronization and alignment procedures, as well as several post-processing steps to distil the final secret key. Namely, these steps comprise sifting of inconclusive measurement results, correction of quantum bit errors, verification, parameter estimation, privacy amplification to remove information leakage, and authentication of all communication

✉ Jeremy Constantin
jeremy.constantin@epfl.ch

[1] Telecommunications Circuits Laboratory, École Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland

[2] GAP-Optique, Université de Genève, 1211 Genève, Switzerland

[3] Battelle, Columbus, OH 43201, USA

[4] University of Applied Sciences Western Switzerland, 1401 Yverdon-les-Bains, Switzerland

Published online: 18 November 2015

produced by post-processing steps to prevent man-in-the-middle attacks. The choice of appropriate protocols and their implementations depends mainly on the composability of the algorithms to maintain information-theoretic provable security, their efficiency, and the trade-off between desired throughput, complexity and the allocated resources. Moreover, as the last step of authentication consumes a fraction of secret key bits distributed in a previous QKD round, all post-processing must be carefully designed with respect to minimizing their required amount of communication. On the other hand, the analysis of finite size effects for QKD puts severe requirements on the minimum block size of raw key bits that have to be processed as a single entity [24].

Traditionally, all post-processing procedures have been implemented in software and run on processor-based systems, from powerful workstations [16] to specialized systems-on-chip for security applications, using embedded processors [14]. However, increasingly, implementations of post-processing based on custom hardware become more and more attractive and necessary to support the higher throughput of modern QKD physical layers. For example, in [9] error correction for QKD based on low-density-parity-check codes (LDPC) has been implemented on a graphics processing unit (GPU), yielding decoding rates of 7.3 Mbps, before privacy amplification, i.e., not final secret key rates ($5-10\times$ lower, depending on channel conditions). In [27], all post-processing except authentication was implemented on two field programmable gate arrays (FPGA) to support a QKD system at a rate of 20 MQubps, with a final secret key rate of up to 17 kbps (theoretical post-processing rate of 70 kbps). Cui et al. [6] illustrates considerable performance benefits of a hardware implementation of a real-time error reconciliation module in an FPGA, over similar error correction algorithms running on a workstation. The fastest complete post-processing engine so far has been demonstrated in [23], where six FPGAs were used, each dedicated to one of the post-processing steps, to support up to eight 1.25 GHz clocked QKD channels in parallel. There, the block size per distillation round is 10^6 bits, and the theoretical post-processing output key rate is 1 Mbps, with measured sustained secure key rates of up to 200 kbps.

Contributions In this paper we present and analyze a fast and flexible distillation engine for QKD which has been developed in the scope of the Nano-Tera QCrypt project [25]. Although our system requires only one single medium-sized industry standard FPGA per user, our distillation engine performs the complete QKD operation including synchronization, continuous real-time distillation and authentication for secret key rates up to 4 Mbps. The employed post-processing block size after error correction is 10^6 bits, and all classical communication channels are time-multiplexed in one optical channel in each direction next to the quantum channel to run the complete system continuously. All implemented algorithms (except for the expansion of the quantum random number generation) have been chosen to maintain information-theoretic provable security, and are optimized to minimize authentication costs, which results in an overall higher secret key rate.

2 Quantum Key Distribution Optics

Our hardware distillation engine has been developed with the aim of providing a flexible, compact and fast building block for QKD. It is independent of the respective QKD protocol or optical implementation, and it supports in particular popular QKD protocols such as BB84 [3] or differential phase-shift (DPS) [22].

Here, we implemented and tested the engine with a high-speed coherent one-way (COW) [20] platform. A schematic of the COW protocol is presented in Fig. 1. The sender (Alice) has a continuous-wave laser and an intensity modulator, which generates a random series of optical pulses. The information is encoded in time, i.e., the presence of the pulse either in an early or a late time bin, encodes a logical one or zero, respectively. The pulses are attenuated such that

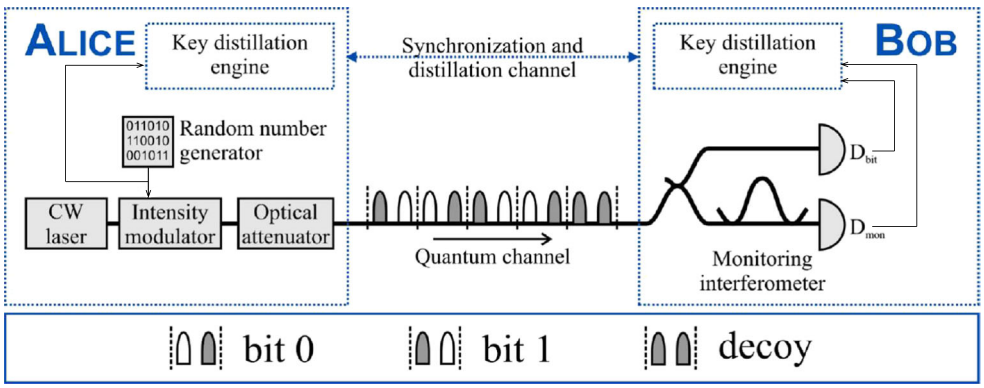

Figure 1 Schematic of the coherent one-way (COW) QKD protocol.

they contain on average less than one photon. The receiver (Bob) measures the arrival time of photons with a single photon detector D_{bit} in order to retrieve the information. This encoding scheme based on time bins hence requires a very accurate clock synchronization between both parties. In order to guarantee the secrecy of the transmission, an interferometer is added at Bob. Any attempt of an eavesdropper (Eve) to intercept the photons, will cause a decrease of the interference visibility, which can be detected with the monitoring detector D_{mon}. For security reasons, decoy sequences (optical pulses in both time bins which do not encode a bit value) have to be introduced randomly, too. For a detailed description of the QKD protocol see [20].

3 Secret Key Distillation Process

The secret key distillation process enables reliable key agreement between Alice and Bob on top of the optical signalling scheme (e.g., the introduced COW) that is employed by the QKD system. The distillation process is detailed in Fig. 2 in the form of a sequence diagram. While the individual processing units involved in the distillation and their interactions including the corresponding data flows are presented in greater detail in Section 4, we summarize the process as follows:

Alice sends random qubits over the quantum channel to Bob, and the secret key distillation then begins after single photon detection on Bob's side. For this detection to work, both Alice and Bob need to be synchronized, which is achieved by a high precision clock recovery scheme attached to the classical communication channel which is a point to point connection between both sides in addition to the quantum channel. During the sifting step Bob sends its successful detection timings[1] (the bit index, not the detected bit values) back to Alice, who in turn reveals the decoys that were introduced by her, such that both sides agree after sifting on the same raw bits that will form part of the secret key.

Following the sifting phase, the core signal processing on the raw key bits is performed. In a first step the errors in the detected bits are corrected with a forward error correction code. The resulting key bits are then verified and filtered accordingly using a randomized universal hash function process. Note that most key distillation steps so far are leaking some amount of information to Eve over the classical communication channel, e.g., in form of redundancy added by the syndromes used for error correction. This quantity of leaked information and the information leaked to Eve through the quantum channel (in case of an eavesdropping attack) needs to be removed from the key bits to produce the final secret key, in a step called privacy amplification. We employ Toeplitz hashing with a randomized matrix to amplify the key bits privacy. The secret key bits are then forwarded to a key manager, which uses some of the bits for authentication of the classical channel and emits the rest as the secret key which is shared between Alice and Bob.

[1] Note that only a small fraction of the pulses can be detected, strongly depending on the optical fiber length between Alice and Bob. However, these missed detections do not result in bit errors, since these bits are omitted by Alice during the sifting process.

4 Secret Key Distillation System Architecture

The hardware architecture of the key distillation system is presented in Fig. 3. The diagram show all blocks necessary to operate either as Alice or as Bob. The front-end that is fed by the quantum channel consists of the sifting modules, which rely on the synchronization and alignment blocks that handle the clock recovery and exact synchronization of the transmitter and receiver systems (cf. Section 4.1). On Alice's side the front end also includes a quantum state selection module, which consumes random bits from an external quantum random number generator, and generates the random qubits that are sent over the optical quantum channel to Bob. In fact, the required random numbers are provided on both Alice and Bob by an intermediate AES randomness expansion block, which expands the available throughput of random bits to the range of hundreds of Mbps

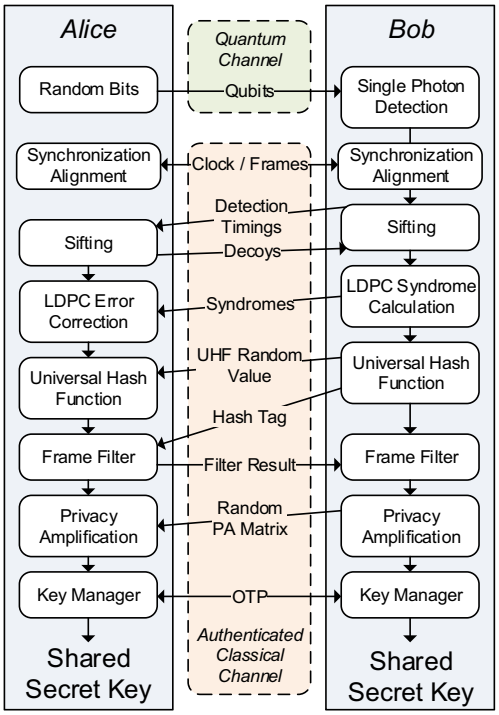

Figure 2 Sequence diagram of secret key distillation process.

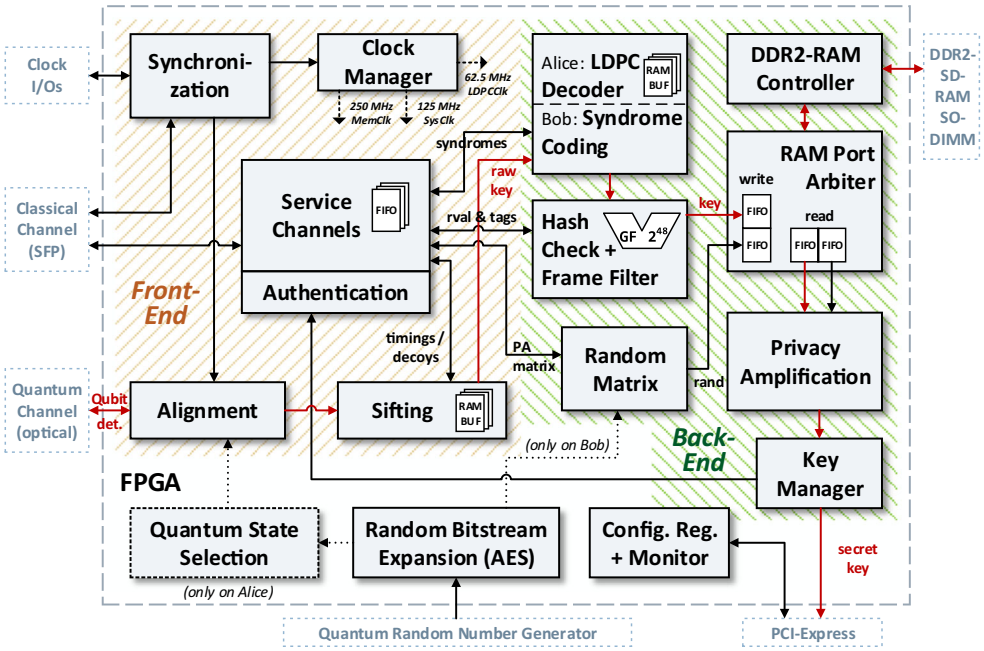

Figure 3 System architecture of the secret key distillation engine.

up from the limited output rate of the truly random quantum source [8].

The back-end or key post-processing chain employed to distil the secret key consists of two core modules, the forward error correction (FEC) and privacy amplification (PA).

The FEC module is based on a quasi-cyclic low density parity check (LDPC) code, with syndrome encoding performed at the quantum bit receiver side (Bob). The transmitter side (Alice) performs decoding of its own transmitted codeword bits to adapt (correct) them according to the syndrome information received from Bob. Please refer to Section 4.4 for an in-depth description of the FEC and its embedded frame filter mechanism based on randomized universal hashing.

After error correction, the key bits are hashed in a privacy amplification module, which effectively compresses the raw key by an adjustable compression ratio (secret fraction) to obtain the final secret key (cf. Section 4.5). The random bits required for the PA step are generated on the receiver side (Bob) and transmitted over the service channel to Alice. Since the PA involves large block sizes of the order of MBit to reduce the impact of finite size effects, the key and random bits have to be stored in off-chip memory. During processing, the raw key bits are then repeatedly read by the PA module from external memory, while new raw key bits for the next block are simultaneously produced by the FEC and stored in a back buffer, also located in external memory. Consequently, the core architecture also comprises a module dedicated to the arbitration of the single external DDR memory port, which is detailed in Section 4.6. This module allows to keep FPGA-internal buffer sizes low by enforcing a memory access schedule with high memory interface utilization.

The key distillation system furthermore employs a PCI-Express link to a PC, used for configuration, monitoring and internal debugging purposes. The link also provides a means to extract the final shared secret key and redistribute it to appropriate consumers, e.g., OTP encryption systems, or high speed AES encryptors [15].

The full system architecture is targeted to be implemented on a single-FPGA (Xilinx Virtex-6 LX240T) board that supports Small Form-factor Pluggable (SFP) transceivers, a PCI-Express interface, and external DDR-RAM.

In the following we elaborate on the specific implementation details of signal processing modules that are unique to the context of quantum key distillation.

4.1 Synchronization and Clocking

The QKD system requires a synchronized and latency compensated clocking scheme between Alice and Bob, to correctly detect the qubits (i.e., to identify the correct time bins) at Bob. Figure 4 shows this clocking scheme. The clocks of Bob and Alice are synchronized through the classical channel, which is using a SFP link at a frequency of 2.5 GHz. Bob recovers the data and also the clock that is sent over this serial link from Alice with the 8B/10B protocol using the clock-data recovery (CDR) function provided by the FPGA built-in GTX transceivers. However, the recovered clock on Bob's side is not sufficiently clean in terms of jitter, for the requirements of the photon detection circuitry on the quantum channel receiver side. Hence, the recovered

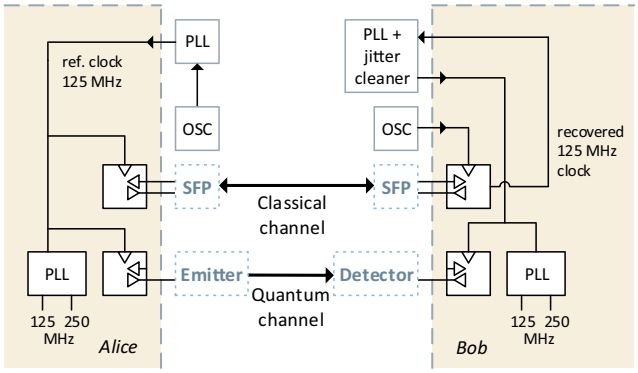

Figure 4 System clocking scheme with clock recovery.

clock is fed outside the FPGA to an external PLL with high quality jitter cleaner. The cleaned clock is then used inside the FPGA to drive the alignment block for quantum channel detections, which correctly phase-aligns the triggering of the photon detector with the arrival of the incoming photons on Bob. Furthermore, from this cleaned clock all other system clocks are derived. The clock manager outputs a memory clock (250 MHz) for the DDR2 controller interface and a general system clock (125 MHz) for the signal processing modules. Additionally, on Alice a separate LDPC clock (62.5 MHz) is generated, since the LDPC decoder can be implemented more efficiently at a reduced clock speed.

4.2 Service Channel

The service channel provides a FIFO-based multi-channel abstraction interface for the classical channel serial link which connects Alice and Bob. All signal processing modules in the key distillation engine use their own private channel (accessible through a FIFO interface) to exchange data between the two sides. The multiple data streams are interleaved by the service channel module, which also authenticates the communication between both parties as required by the QKD protocol to exclude any tampering by a third party with the key distillation process data. Information-theoretically secure authentication is provided by families of strongly-universal hash functions constructed based on polynomial hashing as proposed in [5, 26]. There, a $m/2^n$-almost universal family of hash functions is composed with a $1/2^{n-1}$-almost strongly universal family of hash functions to give a $(m+2)/2^n$-almost strongly universal family of hash functions, where $n \cdot (m+1)$ is the length in bits of the input message. The advantages of this method are that only a total of $3n-1$ secret key bits are consumed per authentication to select the hash functions, which is constant in terms of the initial size of the message. A total of $m+2$ multiplications over $GF(2n)$ are performed, as well as the same number of additions (XORs) over n-bit values. The authenticator processes blocks of 995328 bits and outputs an authentication tag of $n-1 = 127$ bits. To decrease the key bits consumption from 383 bits per block down to almost only 128 bits per block, we exploit the fact that the system has to authenticate several messages by using the variant proposed in [26]: instead of generating a new strongly-universal hash function for each message and hence consume $3n-1$ bits, one can generate a single strongly-universal hash function and keep it secret. Then, every authentication tag is encrypted with a one-time-pad. With this re-use scheme, authenticating t messages requires $3n-1+t \cdot (n-1)$ bits instead of $t \cdot (3n-1)$ bits when no key is reused. It was recently shown in [17] that, in Canetti's universal composability (UC) framework [4], this authentication protocol with key re-use is ϵ-UC-secure, which means that the overall authentication error is upper-bounded by $t \cdot \epsilon$.

To simplify the service channel implementation at high data rates, we avoid an acknowledge- or retransmission-based communication protocol. However, since the transmission bit error rate of 10^{-12} for the serial link at 2.5 Gbps is too high for our purposes (one bit error approximately every 7 min), we perform error correction on the classical channel data with a Hamming(15,11) code to bring the error rate down to 10^{-18}. The encoding and decoding of the Hamming code is implemented using lookup tables.

4.3 Sifting

The sifting module on Bob's side discloses the timings and types (base) of the photon detections to Alice over the service channel. The encodings for this sifting information can be chosen with a precision of 6 or 14 bit for the differential timestamp data which is always relative to the previous detection. This flexibility in the number of bits allows the system to be better tuned to the detection rate which drops with increasing fiber-length. With the long encoding up to 250 km are feasible, while the service channel bandwidth usage is better optimized for short distances (up to 20 km) with shorter time stamps. Alice prunes its bit stream according to the timing information from Bob, and additionally discloses to Bob the decoy positions and monitor detection outcomes, which Bob uses to also sift its own stream of raw key bits and to estimate the visibility.

Note that the internal buffer of the sifting module on Alice's side needs to be of considerable size. The longer the distance between Alice and Bob, the larger is the required sifting buffer, since Alice needs to hold all transmitted bits in this buffer, until the sifting information from Bob arrives. For a fiber length of 50 km the round-trip time is about 500 μs, which results in a typical buffer size of 625 kbit to store both the bit value and the base at a link rate of 625 Mbps. At 250 km of fiber length the required buffer increases to almost 3 Mbit to accommodate for the round

trip time increase to 2380 μs. Hence, at larger link distances an external RAM buffer is potentially needed to store the transmitted bits of Alice efficiently before they can be sifted.

4.4 Forward Error Correction

4.4.1 LDPC Codes

LDPC codes have been chosen for the error correction because of their excellent Shannon-capacity approaching error correction capabilities and the existence of low-complexity decoding solutions [21]. A valid code word x of a classical LDPC code is characterized by satisfying the condition that

$$H \cdot x = s$$

where H is the sparsely populated parity check matrix, which defines the specific instance of the code, and s is a known vector called syndrome. The aspect ratio of H defines the coding rate (R). LDPC codewords can be decoded by believe propagation (BP) algorithms.

For our implementation we have chosen a quasi-cyclic prototype matrix H as it used in various modern communication standards. The set of parameters and the H are derived from the LDPC codes specified for the IEEE 802.11n standard [1].

The quasi-cyclic nature of H allows the use of a partial-parallel decoder architecture based on a layered decoding schedule [19]. Such an architecture offers a good trade-off between efficient implementation and high reconfigurability. Hence, we can support a wide set of different code rates without additional complexity [21]. Adapting the coding rate to the channel conditions allows to utilize the quantum channel with high efficiency over many different scenarios. Results are provided for 4 different parity check matrices corresponding to $R = \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}$. A fixed codeword size of 1944 is sufficient as our application assumes a constant data stream.

In conventional wireless applications the syndrome of the LDPC code is usually fixed to $s = 0$. The parity information is encoded in the codeword x. For the quantum channel application we adopt the concept of syndrome coding as described in [13]. In our system the syndrome s is not a fixed value but dynamically calculated by Bob for each codeword based on his sifted data.

The forward error correction provided by the LDPC decoder in the QKD system achieves a residual frame error rate (FER) of less then 0.5 % at a quantum bit error rate (QBER) of 6 % on the quantum channel, for a code rate of $R = 1/2$, as can be observed in Fig. 5. One frame corresponds to a codeword of size 1944 bits, i.e., for a QBER of 6 % each frame has an independent probability of 0.5 % of being filtered out by the following check mechanism of the FEC, due to an unrecoverable error.

The efficieny of our system is defined as the achieved percentage of the ideal Shannon capacity of the channel based on the QBER. The capacity of the binary symmetric channel is defined as

$$C(\epsilon) = 1 - h(\epsilon)$$

with $h(\epsilon)$ being the entropy of a bit with error probability ϵ which corresponds to our QBER. The achievable error free throughput of information is calculated as

$$G = (1 - \mathrm{FER}) \cdot R$$

FER is the frame error rate after decoding and also a function of QBER. We can see from Fig. 6 that by switching between rates we can achieve a high efficiency for different QBER regimes. Especially at high rate codes there is a significant gain compared to the results published in [27].

The LDPC decoder operates at a clock frequency of 62.5 MHz (halved system clock), with 10 iterations per decoded code word, providing a maximum throughput of 235 Mbps. Hence, the processing speed of the LDPC decoder is not the limiting factor for the overall maximum secret key rate of the key distillation engine.

4.4.2 Frame Filtering

After error correction of the sifted key bits, it is possible that a codeword on Alice side still has a mismatch compared to the bits on Bob (FEC failure). Hence, the QKD system requires an integrity check on the decoded key bits such that no mismatch in the final agreed secret key stream is possible (with an upper-bounded probability $\epsilon < 2^{-32}$). Verification of the parity check equations in the LDPC decoder provides a first error detection mechanism. Unfortunately, this mechanism leaves many block errors undetected, since the

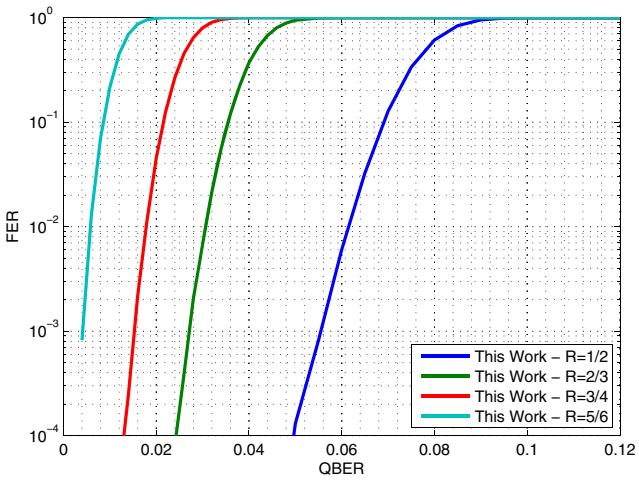

Figure 5 Error correction performance of forward error correction (LDPC decoder).

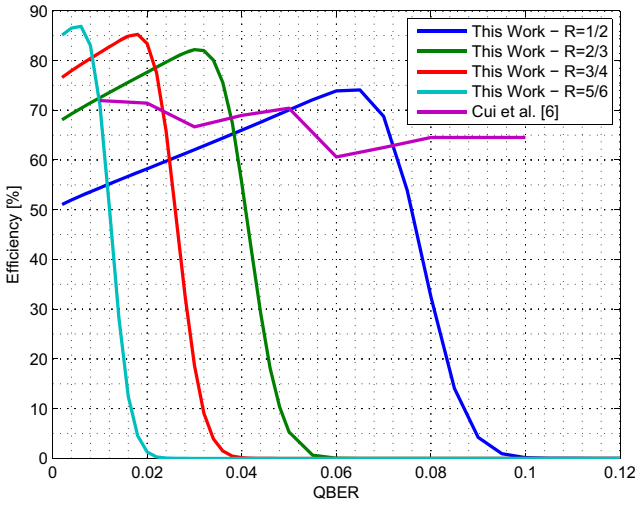

Figure 6 Channel usage efficiency for different code rates.

decoder tries to produce a valid codeword and converges, despite being in error. Hence, we implement an additional integrity check, i.e., error detection after correction, by performing a hash tag calculation and comparison for each LDPC code word of size 1944 bits after zero-padding it to 1968 bits. The padding is necessary, since the hashed word needs to have a length which is a multiple of 48 bit.

After calculation of the syndrome for the sifted key bits on Bob's side, a 48-bit hash tag is calculated for each code word and sent to Alice, together with a random value of the same size, which is needed to randomize the hashing process (required for security reasons). Alice then calculates its own hash tag for its decoded key bits, using the same random value, and then performs a comparison of the tags. If a tag mismatch occurs, the corresponding frame (codeword) is discarded by Alice, before the key bits are written to the external memory for further processing. All hash tag comparison results are furthermore forwarded to Bob, which also performs the corresponding filtering of frames on his key bits stream. Note that the internal FIFO buffer required at Bob to perform the frame filtering can grow to considerable size, if the round-trip delay time on the service channel for the hash check is large.

As mentioned above, the process of hash tag calculation requires randomization, since it is necessary to prevent an attacker from forging an attack vector on the service- or quantum-channel that could force a collision of the employed hash function, resulting in a key agreement between Alice and Bob on differing key bits.

To prevent this, we employ an error detection method named polynomial hashing, which is built on top of a universal hash function (UHF) [5, 26]. The UHF is based on finite-field arithmetic over the Galois Field $GF(2^{48})$ with the polynomial $x^{48}+x^{47}+x^{21}+x^{20}+1$. The method uses a 48-bit uniformly distributed random value to avoid any targeted collision attacks, and produces a 48-bit hash tag as output. A new random value is used for each invocation of the UHF, i.e., for each calculated hash tag. The maximum collision probability for a single tag is $2^{-42.6}$, while for a sequence of 512 tags (one PA block) it increases to $2^{-33.5}$.

The block diagram of the UHF unit is shown in Fig. 7. The accumulator of the UHF module is loaded during initialization with the first 48 bit of the key bits (code word) to be hashed. In the following 40 rounds the hash tag is sequentially computed by performing a $GF(2^{48})$ multiplication of the accumulator with the defined random value for the current tag. After the multiplication the result is XOR'ed with the next 48 bit of key bits, before being stored back in the accumulator. Note that the last part of the code word (k_{40}) is only 24 bit long and needs to be padded with zeros to fit the 48-bit format.

The complexity of the UHF module lies in the required $GF(2^{48})$ finite-field multiplier. Our design space exploration showed satisfactory results for a fully combinatorial approach [12]. The multiplier generates all possible multiples (GF powers) of the first operand, using chained exponentiation blocks. Note that these blocks only contain a fixed barrel shifter (rotate by one bit), and three parallel XOR gates at fixed bit positions (defined by the $GF(2^{48})$ polynomial), which reduces the average logic depth through the exponentiation chain down to only three XOR gates. All the multiples are then each combined with the second input operand and the result vectors are finally individually summed to produce the 48-bit result.

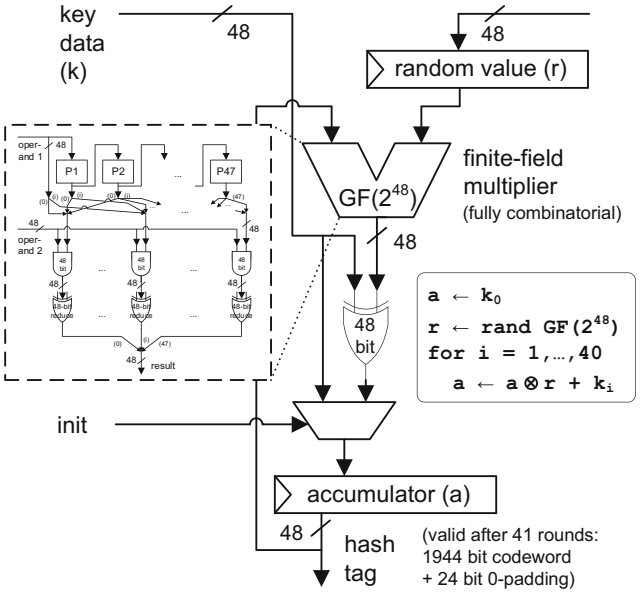

Figure 7 Block diagram of the universal hash function module using randomized polynomial hashing over $GF(2^{48})$.

4.5 Privacy Amplification

The transmission of key bits from Alice to Bob is accompanied by leakage of information due to the necessary communication on the service channel for sifting, error correction and detection, or due to eavesdropping attacks. Hence, to guarantee a certain level of security, it is necessary to remove this quantity of leaked information through a privacy amplification step. During privacy amplification a partially secure string is to be transformed into a highly secret key by public discussion. It has been shown that this can be realized by computing the output of a randomly but publicly chosen two-universal hash function applied to the input string, which results in a secret key that is secure according to a universally composable security definition [18]. To this end, we employ Toeplitz hashing [11] to reduce the error corrected bit stream by an adjustable compression ratio, ensuring the security of the remaining secret key bits.

Toeplitz hashing is performed by a single matrix-vector multiplication over GF(2), according to

$$\begin{pmatrix} r_1 & r_2 & \cdots & r_n \\ r_{n+1} & r_1 & \cdots & r_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n+m-1} & r_{n+m-2} & \cdots & r_{n-m-1} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

where the $m \times n$ matrix consists of random bits r_i with $i = 1 \ldots n + m - 1$, and the corresponding vector is constructed from the n raw key bits x_j with $j = 1 \ldots n$ that need to be privacy amplified, i.e., hashed, resulting in m secret key bits y_l with $l = 1 \ldots m$. The employed matrix is a diagonal-constant (Toeplitz) matrix, which means that it has a regular diagonal structure. This specific structure reduces the amount of random bits required for the compression of a vector of length n with a secret fraction of m/n (using an $m \times n$ matrix) from mn to $m + n - 1$, compared to a fully random matrix. This allows to reduce the number of random bits that have to be generated, processed and sent over the service channel to a reasonable amount.

Note that the random matrix is not kept secret, however it is not to be revealed over the public channel before all qubits associated with the hashing matrix have been transmitted and the raw key bits have been error corrected and filtered on both sides.

For correct handling of unwanted finite size effects, the block size n for the PA vector should be as large as possible. We choose a block size of $n \approx 10^6$ (specifically $n = 512 \times 1944$ for our reference implementation), which for a typical secret fraction of 10 % leads to a multiplication with a Toeplitz matrix of dimensions $10^5 \times 10^6$.

Since the total amount of bits required to construct the random matrix and the raw key bit vector itself are typically too large to fit into FPGA on-chip memory (BRAMs), the data is kept in off-chip memory (e.g., DDR2-RAM), and is processed in slices. The complete matrix multiplication is performed in multiple steps, by slicing the result vector into m/k slices of length k bits each. Every result slice $(y_{(s-1)k+1}, y_{(s-1)k+2}, \ldots, y_{sk})$ for $s = 1 \ldots m/k$ is calculated completely, before continuing with the next part of the result vector.

The architecture of the PA performing the matrix multiplication is shown in Fig. 8. The calculation of one slice is performed in multiple cycles, processing w raw key bits per cycle. The hashing architecture consists of k parallel multiply-accumulator (MAC) units, which each perform per cycle in parallel w single-bit multiplications over GF(2), combining random matrix bits with raw key bits. The w results are then summed (XOR / parity bit) together with the current value of the accumulator register to produce the new sum bit. The input raw key bits for the MAC units, i.e., the currently needed part of x of size w, are the same for each MAC unit (in a cycle), while the corresponding row of random bits is taken from a large shift register of size $k + w$. The first MAC unit takes the w random bits starting from the second flip-flop, while the second MAC unit takes the w random bits starting from the third flip-flop, and so on, as illustrated in Fig. 8. Note that the first flip-flop is only used for loading purposes (of full random words of size w), but not directly for calculation.

As the example in the bottom right corner of Fig. 8 shows, every cycle the shift register is advanced by a step size of w, shifting in a new word of random bits at the top (e.g., random bits r_5, r_6, r_7 in cycle 2). By only storing each cycle the boundary of the currently required matrix area of dimension $k \times w$, we achieve a very area-efficient implementation of the Toeplitz hashing algorithm. We exploit the regular diagonal structure of the matrix, by a shifting mechanism that allows the boundary required for cycle i to be easily transformed into the boundary that is required for cycle $i + 1$.

Before the calculation of a slice can begin, initial random matrix bits (the k bits that correspond to the slice of bits of the first column of the full random matrix) are preloaded into the lower part of the shift register, loading w bits per cycle. At the same time the result bits y of the previous slice calculation are shifted out of the MAC units, and are forwarded to the key manager as secret key bits.

During the calculation of one slice, the architecture consumes w random bits as well as w key bits per cycle. The calculation for one slice is processed in n/w cycles, while the loading phase takes k/w cycles. Since one full PA block consists of m/k slices, generation of m secret key bits uses $((k/w) + (n/w))(m/k)$ cycles (including all load overhead between slices).

The parameter w is chosen based on the width and throughput of the external memory interface, i.e., how much

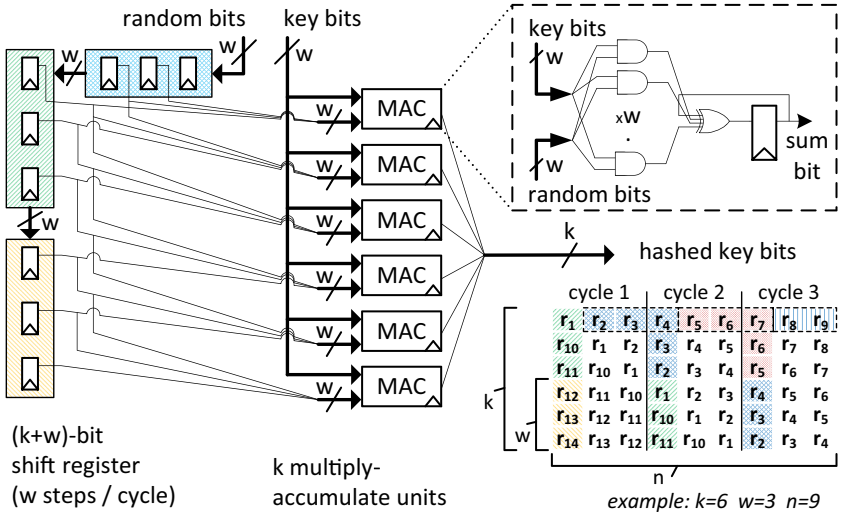

Figure 8 Toeplitz hashing architecture employed by privacy amplification.

data it can provide per clock cycle. The main parameter for adjusting the circuit area (FPGA resource) utilization as well as the degree of parallelism, is the number of parallel MAC units, k.

The PA architecture supports flexible compression ratios, providing support for the optimal secret fraction corresponding to the physical link properties of the underlying QKD system installation and the desired finite block length security parameter. The secret fraction can be set in the full range of 0 to 100 % in increments of k/n.

The design space exploration of the privacy amplification unit is presented in Table 1. The values relate to a QKD system using a Virtex-6 LX240T FPGA, where the PA is clocked at 125 MHz. The external memory used by the PA is a DDR2 SDRAM providing a peak throughput of 25.6 Gbps. Processing of one full PA block is associated with a certain number of bits that need to be transferred (written and read) from the external memory, depending on the number of available MAC units (and the compression ratio). The number of read bits mainly depends on the number of slices m/k that need to be processed for one PA block. Since the secret key output size (per PA block / matrix multiplication) scales directly with the compression ratio, the maximum key rate is practically independent of the compression ratio and only depends on the number of parallel MAC units k. This maximum key rate is always bounded by the available external memory throughput.

The actual achievable key rate for a given number of MAC units k then depends on how much hardware in form of LUTs is expended by choosing an appropriate word width w for the MAC units. In general, doubling of that word width provides a two-fold increase in output key rate, while also requiring twice the amount of FPGA LUTs. The number of required flip-flops is about $2k$, since each MAC unit contains one flip-flop, and a shift-register of size k is also needed. Hence, to optimize the resource utilization it is beneficial to choose the largest w such that the maximum key rate (bounded by the memory throughput limit) is not exceeded. In our specific system a word width w of 128 would not be beneficial (for any k), since the supported key rate would always exceed the maximum that the external RAM allows.

Regarding the available resources in the V6LX240T FPGA (150720 LUTs), a configuration of $k = 512$ and $w = 64$ provides a good compromise between supported maximum key rate (4.1 Mbps) and the required device utilization (11 %). With a PA block size of $n \approx 10^6$ this configuration enables a very fine step size of 0.05 % for the secret fraction PA compression parameter.

4.6 External Memory Arbitration

The QKD system architecture as shown in Fig. 3 comprises an external memory for storage of data blocks that are too large for on-chip storage in FPGA BRAMs. Since the raw key block size (n) used as input for the privacy amplification step is in the order of MBit, and the PA algorithm needs access to the full block multiple times for a single compression (i.e., simple streaming is not possible), the external RAM has to be used for storage of these input blocks and the random matrix bits that are additionally consumed by the PA. It is hence necessary to arbitrate four different producers and consumers of data that need access to the external RAM:

- writing of (corrected and filtered) raw key bits by the LDCP decoder (Alice) / the syndrome encoder (Bob)
- writing of random bits by the Toeplitz matrix generator
- reading of raw key bits by the privacy amplification
- reading of random bits by the privacy amplification

Table 1 Design space exploration of resource utilization for the privacy amplification unit on a Virtex-6 LX240T, with varying numbers of parallel MAC units (k) and word widths (w).

MACs	Transf.	Max. KR	Flip-	$w = 16$		$w = 32$		$w = 64$	
(k)	[Mbit][a]	[Mbps][b]	flops	KR[c]	LUTs	KR[c]	LUTs	KR[c]	LUTs
128	1562	1.64	272	0.26	975	0.51	1950	1.03	3901
256	783	3.28	528	0.51	1950	1.03	3901	2.06	7802
512	392	6.55	1040	1.03	3901	2.06	7802	4.11	15604
1024	197	13.02	2064	2.06	7802	4.11	15604	8.21	31208
2048	100	25.76	4112	4.11	15604	8.21	31208	16.43	62415

[a]Per block, for a secret fraction of $196/1944 \approx 10$ %, i.e. a compression from $\approx 10^6$ to 10^5

[b]Theoretical maximum keyrate, when limited by external memory throughput (25.6 Gbps), i.e. external memory is exclusively used for PA

[c]Achievable keyrate in Mbps, when PA is running at 125 MHz and input data buffers are never starved

The external memory arbitration unit provides the necessary memory access coordination such that all four producers and consumers can operate at the same time, without any of them suffering internal buffer overflow or starvation, respectively. The goal is furthermore to maximize utilization of the single external memory DDR2 SDRAM port, which provides 400 MT/s (mega transfers per second). With 64-bit transfers (read or write accesses) this results in a maximum theoretical external memory throughput of 25.6 Gbps. This limited throughput has to be carefully partitioned in such a way that all modules operate at their highest possible throughput, without being throttled due to inefficient memory access patterns, to allow for an overall high PA output rate, i.e., high final secret key rate.

The arbitration operates on a round-robin schedule based on the state diagram depicted in Fig. 9. In general, the system operates on three buffers, which are used in a ring configuration, each buffer holding a full set of raw key and random bits for one PA block. During normal operation in the PA block loop, the oldest buffer is being processed by the PA, while the two younger buffers are being written with new key and random bits.

At start-up the system first stores one full block of raw key bits into external buffer 0, and then proceeds by filling a second back-buffer 1 with raw key bits, while at the same time storing the random bits associated with the first block of key bits in its corresponding buffer 0. This delay of the random bits by one full back-buffer is necessary for security reasons, since the random bits generated on Bob's side must not be revealed over the service channel, before the entire raw key block has already been processed by the system. After this initial phase of filling two buffers (buffer 1 only with key bits), the RAM arbiter enters its main PA block processing loop, which consists of five states. In the first state the PA internal shift register of size k is preloaded with random bits from memory. This is followed by PA processing of one slice, where key and random bits are read in an alternating fashion, always keeping the two corresponding PA input buffers / FIFOs filled. After one slice has been processed the arbiter writes new key bits followed by new random bits into the corresponding back buffers, before starting to process the next PA slice.

The required total amount of bits that need to be transferred (written to and read from the external RAM) per PA block of size n is $2n(\frac{m}{k} + 1) + 2m - 1$ bits (where m/n is the secret fraction, and k is the slice size, i.e., the number of parallel MAC units).

The round-robin schedule employs varying access window sizes, which are freely configurable for different compression ratios. The sizes are defined by the required access bandwidth ratios, i.e., the amount of bits each port needs to

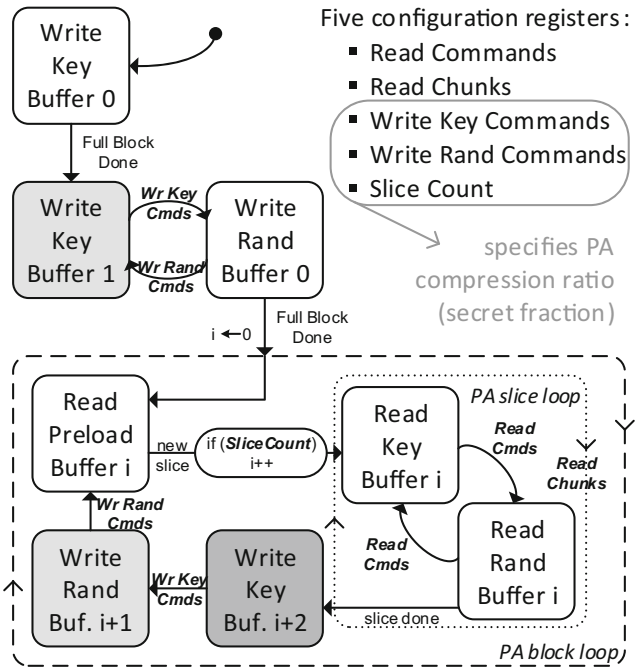

Figure 9 State diagram of configurable external memory arbitration unit for privacy amplification using back-buffers.

write or read, over the total amount of transferred bits per PA block. Free configurability of window sizes also allows for a trade-off of RAM access efficiency (burst lengths), i.e. total throughput, against the size of the FPGA-internal PA input buffers.

5 Implementation Results

The presented key distillation system architecture has been implemented as a prototype on two custom FPGA boards, each hosting a single Virtex-6 LX240T FPGA. The fully operational QKD system furthermore comprises the optical devices for realization of the quantum channel, as shown in Fig. 10. Additionally the setup includes a workstation computer on either side, which manages the configuration of the QKD system and provides monitoring capabilities via the PCI-Express interface. Results for the overall QKD system performance, focusing on the quantum physics and optical systems (which are currently the limiting element regarding the measured secret key rate in the QKD system) can be found in [25] and a demonstration of the QKD engine with the BB84 protocol is presented in [10].

The Virtex-6 LX240T FPGA has a total of 150720 LUTs and flip-flops, as well as 834 BRAMs (of size 18 kbit) available. On Alice the percentage of used LUTs is 38, and 98 % of BRAMs, while on Bob 42 % of LUTs, and 43 % of BRAMs are used. FPGA device utilization regarding BRAMs on Alice is higher than on Bob, due to the required larger sifting buffers. The LUT utilization is similar on both sides, although the forward error correction part on Alice's side is significantly larger due to the higher complexity of the LDPC decoder compared to the syndrome encoder on Bob's side. Bob's system however has additional LUT requirements due to an additional delay module (partly grouped under front-end other), which is required for its sifting functionality.

Table 2 shows a detailed breakdown of the device utilization (relative to the full FPGA), regarding the different modules of the system.

The key distillation engine is furthermore designed that all signal processing modules meet timing at a 125 MHz system clock speed (with halved frequency for the LDPC decoder).

We provide an overview of FPGA-based hardware implementations of different QKD systems (including their distillation engines) in Table 3. A direct comparison regarding metrics such as clock speed and device utilization, in relation to the achieved processing rates, proves difficult due to the wide range of different devices and overall system architectures employed in the respective designs. Furthermore, the absence of any official standardisation regarding the QKD process and the underlying quantum channel link makes comparison challenging, because of the varying requirements on highest system level.

However, in direct comparison to [27] and [23], we show in this work for the first time that a QKD distillation engine with multi-Mbps secret key rate can be implement in a single FPGA-device per user system architecture, with moderate resource requirements. Our distillation engine outperforms the current state-of-the-art implementations [23, 27] in terms of their maximum secret key rate throughput capabilities. It has to be noted that [23] does not provide any details on the underlying FPGA device specifics, and the (possibly) available processing headroom in terms of distillation engine throughput can hence not be fully assessed.

5.1 Processing Rates

This subsection provides an overview of the capabilities of our distillation engine regarding its internal processing rates. We complement this engine rate analysis with measured numbers from our running QKD link [25] including the quantum channel devices, to illustrate the available processing rate headroom of our developed engine.

As shown in Fig. 2, random bits are generated on Alice and sent to Bob in form of qubits, where they are detected. In our setup the transmission rate of random bits on Alice is 625 Mbps, while the detection rate on Bob strongly depends on the fiber length and other aspects of the quantum channel setup. The detection probabilities range from the order of 10^{-1} to 10^{-4}. Concerning our distillation engine, the maximum output key rates after the sifting process are assumed to be 20 Mbps of raw key bits that both parties agree upon. Note however that even for short distances of 1 km the

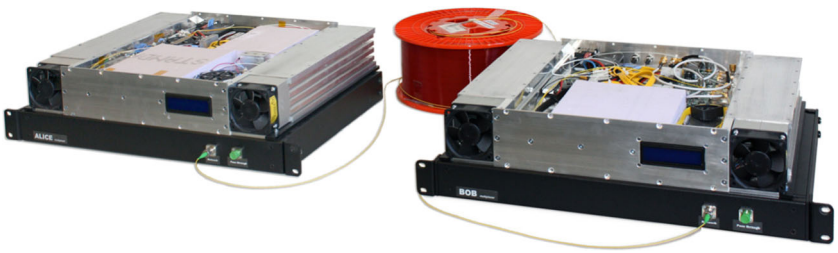

Figure 10 Photo of the opened QKD devices, housing the hardware key distillation engine and quantum optics [25].

measured sifted key rate is with current device technology significantly lower at about 1.3 Mbps (cf. [25] Table 1), while for larger distances the rate typically does not exceed 1 Mbps.

After sifting, the raw key bits are treated further in the distillation process by the error correction and detection modules, specifically the syndrome encoder and LDPC decoder, as well as the UHF frame filter, which all have to support a rate of 20 Mbps. Since the syndrome calculation is computationally rather inexpensive (small matrix vector multiplication), the main focus here lies on the LDPC decoder on Alice, which in our parallel implementation actually supports a decoding throughput of up to 235 Mbps (at 62.5 MHz). As demonstrated in Section 4.4.2 the UHF frame filter can process 48 bit per cycle, which provides at 125 MHz a throughput of 6 Gbps. The reduction of the key bit rate after error correction and detection is only very minor, depending on the QBER of the link. The rate reduction is typically limited by about 1 %, also depending on the chosen code rate of the LDPC (cf. Fig. 5). The output key rates of the error correction and detection modules are hence assumed to be at a maximum of 20 Mbps.

This bit stream is then privacy amplified, using a compression factor (secret fraction) of typically 5–20 %, depending on the link conditions and the requirements on the security parameter. Our link measurements resulted in a derived required range for the compression factor between 6.5 and 11.5 % (cf. [25] Table 1), for a security parameter of $\epsilon_{QKD} = 4 \times 10^{-9}$. Assuming a maximum secret fraction of 20 % (i.e. the highest ratio of output rate to input rate), the required maximum output rate of the privacy amplification module is 4 Mbps, at 20 Mbps input key rate (together with 24 Mbps of random matrix data rate). We meet this requirement by implementing the PA with a $k = 512$ and $w = 64$ configuration (cf. Table 1). Although measured secret key rates using the quantum channel are currently limited to 145 kbps, our QKD distillation engine supports the processing of secret key rates of up to 4 Mbps on a single FPGA device.

5.2 Scalability

Considering improved devices on the quantum physics and optical systems layer, the QKD system is required to post process increasingly higher rates of incoming bits. The presented QKD distillation engine scales favourably with higher input/detection rates, since the employed LDPC decoder (which is computationally the most complex module in the post-processing system, also regarding optimizations) has in its current implementation already a throughput headroom of about 10× (depending on the error rate) compared to the currently assumed maximum requirement of 20 Mbps. This is despite it running with only half the nominal clock frequency of the other modules (i.e., the LDPC decoder contains the critical path of the system). To leverage this headroom it is required to increase the privacy amplification throughput, where higher rates can be achieved without much effort by scaling up the number of parallel MAC units k, as shown in Table 1. Hence, this requires only a linear increase in area of the PA module for extending the available final secret key rate linearly. This approach is particularly feasible when targeting newer, large FPGAs such as Virtex-7 devices with more than 1 million available LUTs.

With higher FEC and PA rates the communication requirements in terms of bandwidth on the classical communication channel also increase. The main bottleneck in this regard however would be the increased amount

Table 2 FPGA device utilization with key distillation module breakdown.

	Alice		Bob	
Module	LUT	BRAM	LUT	BRAM
Sifting	0.8 %	54.8 %	3.9 %	0.2 %
Service-channel	4.8 %	9.1 %	4.9 %	8.5 %
Front-end other	2.0 %	8.9 %	6.7 %	3.5 %
Key-verification (debug only)	0.1 %	0.7 %	1.0 %	13.9 %
LDPC decoder / syndrome coding	9.2 %	7.5 %	1.7 %	0.7 %
Hash function + frame checking	1.3 %	0.5 %	1.3 %	1.1 %
RAM arbiter	0.9 %	3.7 %	0.9 %	3.5 %
Privacy amplification	4.8 %	0.0 %	4.8 %	0.0 %
Random stream expansion	2.0 %	0.5 %	3.7 %	0.0 %
Key manager	0.6 %	4.6 %	0.7 %	4.3 %
External RAM controller	9.7 %	5.0 %	9.8 %	4.8 %
PCI-Express	2.0 %	2.7 %	2.5 %	2.6 %
Total utilization	38 %	98 %	42 %	43 %

Table 3 Comparison of QKD distillation engines

	Zhang et al. [27]	Tanaka et al. [23]	This work
Devices (per user)	2× Altera Cyclone III EP3C120	6× FPGA (unknown type)[a]	1× Xilinx Virtex-6 LX240T
Clock frequency [MHz]	20/80	–[a]	125
FPGA resources	7707 LEs	–[a]	63302 LUTs
Internal BRAM [kbit]	1049	–[a]	14711 / 6455
External RAM [Mbit]	128 (SRAM)	40960 (?)	2048 (DDR2)
Sifted key rate [Mbps]	0.185	≈ 2.4 (measured)	20
Max. secret key rate [Mbps]	0.070	1.5	4

[a]Tanaka et al. [23] does not provide any further technical system details regarding the employed FPGA devices and their operating conditions for the realization of the QKD distillation engine

of data that needs to be transmitted to perform the sifting operation (it accounts for 95 %+ of the traffic [25]). However, currently available (classical) optical high speed transceivers should not pose a limiting factor for the overall communication requirements of the distillation engine, even when processing considerably higher qubit input rates.

6 Conclusion

Increasing detection rates of today's QKD systems can only be sustained with appropriate real time secret key distillation engines delivering the required high post-processing throughput. We show that a full secret key distillation engine supporting secret key rates of up to 4 Mbps can be realized in a single-FPGA (per user) QKD setup, while remaining highly configurable, which allows adjustment of distillation parameters to the QKD installation environment and its constraints, e.g., the fiber distance.

From a system level perspective it is imperative for a QKD system to always validate the provable security aspect of the system, at every step of the design process. This is especially important during integration and optimization of the different signal processing blocks, which are connected but are spread out over the two different parties. For example, information must often not be revealed at any given time (e.g., when it is ready for further processing in a data flow sense), but must be delayed accordingly to not break the security of the QKD system. These additional constraints add another considerable layer of complexity to the system design and verification process.

We achieve a high secret key rate with a moderate hardware resource utilization by custom designing all involved signal processing blocks to the specific requirements of the underlying communication and cryptographic tasks. Since the developed LDPC decoder performing the forward error correction has still enough headroom for even higher data rates and the throughput of the privacy amplification is mainly resource limited and can efficiently be parallelized, we expect secret key rates to scale well for future implementations targeting larger FPGA devices.

Acknowledgments The authors greatfully acknowledge Julien-Kenji Izui, Xavier-Christian Paillard, Gregory Trolliet, Fabien Vannel from HEPIA Geneva and Olivier Guinnard from the University of Geneva for their contributions to some key system components that are beyond the scope of this paper. We also acknowledge the financial support of the Swiss Nano-Tera program for the QCRYPT project.

References

1. IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC)and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput. IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009) pp. 1–565 (2009).
2. Bennett, C., Bessette, F., Brassard, G., Salvail, L., & Smolin, J. (1992). Experimental quantum cryptography. *Journal of Cryptology*, *5*, 3–28.
3. Bennett, C.H., & Brassard, G. (1984). Quantum cryptography: public key distribution and coin tossing. In *Proceedings of the IEEE international conference on computers, systems and signal processing* (pp. 175–179). New York: IEEE Press.
4. Canetti, R. (2001). Universally composable security: a new paradigm for cryptographic protocols. In *42nd annual symposium on foundations of computer science, FOCS* (pp. 136–145).
5. Carter, J., & Wegman, M.N. (1979). Universal classes of hash functions. *Journal of Computer and System Sciences*, *18*(2), 143–154.
6. Cui, K., Wang, J., Zhang, H.F., Luo, C.L., Jin, G., & Chen, T.Y. (2013). A real-time design based on FPGA for expeditious error reconciliation in QKD system. *IEEE Transactions on Information Forensics and Security*, *8*(1), 184–190.
7. Gisin, N., Ribordy, G., Tittel, W., & Zbinden, H. (2002). Quantum cryptography. *Reviews of Modern Physics*, *74*, 145–195.
8. ID Quantique: Quantis Random Number Generator (2014). http://idquantique.com/random-number-generators/products.

9. Jouguet, P., & Kunz-Jacques, S. (2014). High performance error correction for quantum key distribution using polar codes. *Journal of Quantum Information and Computation*, *14*(3–4), 329–338.
10. Korzh, B., Walenta, N., Lunghi, T., Gisin, N., & Zbinden, H. (2014). Free-running InGaAs single photon detector with 1 dark count per second at 10 % efficiency. *Applied Physics Letters*, *104*(8).
11. Krawczyk, H. (1994). LFSR-based hashing and authentication. In Y. Desmedt (Ed.), *Advances in Cryptology - CRYPTO 94*, Lecture Notes in Computer Science, (Vol. 839 pp. 129–139). Berlin Heidelberg: Springer.
12. Lin, S., & Costello, D. (2004). *Error control coding: fundamentals and applications*. Pearson-Prentice Hall.
13. Liveris, A., Xiong, Z., & Georghiades, C. (2002). Compression of binary sources with side information at the decoder using LDPC codes. *IEEE Communications Letters*, *6*(10), 440–442.
14. Lorunser, T., Querasser, E., Matyus, T., Peev, M., Wolkerstorfer, J., Hutter, M., Szekely, A., Wimberger, I., Pfaffel-Janser, C., & Neppach, A. (2008). Security processor with quantum key distribution. In *International conference on application-specific systems, architectures and processors (ASAP)* (pp. 37–42).
15. Muehlberghuber, M., Keller, C., Gürkaynak, F., & Felber, N. (2013). FPGA-based high-speed authenticated encryption system. In *VLSI-SoC: from algorithms to circuits and system-on-chip design*, IFIP Advances in Information and Communication Technology, (Vol. 418 pp. 1–20). Berlin Heidelberg: Springer.
16. Pearson, D. (2004). *High-speed QKD reconciliation using forward error correction*, (pp. 299–302).
17. Portmann, C. (2014). Key recycling in authentication. *IEEE Transactions on Information Theory*, *60*(7), 4383–4396.
18. Renner, R., & König, R. (2005). Universally composable privacy amplification against quantum adversaries. In J. Kilian (Ed.), *Theory of cryptography*, Lecture notes in computer science, (Vol. 3378 pp. 407–425). Berlin Heidelberg: Springer.
19. Sharon, E., Litsyn, S., & Goldberger, J. (2004). An efficient message-passing schedule for LDPC decoding. In *Proceedings of the 23rd IEEE convention of electrical and electronics engineers in Israel* (pp. 223–226). IEEE.
20. Stucki, D., Brunner, N., Gisin, N., Scarani, V., & Zbinden, H. (2005). Fast and simple one-way quantum key distribution. *Applied Physics Letters*, *87*(19), 194, 108.
21. Studer, C., Preyss, N., Roth, C., & Burg, A. (2008). Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes. In *42nd Asilomar conference on signals, systems and computers* (pp. 1137–1142). IEEE.
22. Takesue, H., Honjo, T., Tamaki, K., & Tokura, Y. (2009). Differential phase shift-quantum key distribution. *IEEE Communications Magazine*, *47*(5), 102–106.
23. Tanaka, A., Fujiwara, M., Yoshino, K., Takahashi, S., Nambu, Y., Tomita, A., Miki, S., Yamashita, T., Wang, Z., Sasaki, M., & Tajima, A. (2012). High-speed quantum key distribution system for 1-Mbps real-time key generation. *IEEE Journal of Quantum Electronics*, *48*(4), 542–550.
24. Tomamichel, M., Lim, C.C.W., Gisin, N., & Renner, R. (2012). Tight finite-key analysis for quantum cryptography. *Nature Communications*, *3*, 634.
25. Walenta, N., Burg, A., Caselunghe, D., Constantin, J., Gisin, N., Guinnard, O., Houlmann, R., Junod, P., Korzh, B., Kulesza, N., Legr, M., Lim, C.W., Lunghi, T., Monat, L., Portmann, C., Soucarros, M., Thew, R.T., Trinkler, P., Trolliet, G., Vannel, F., & Zbinden, H. (2014). A fast and versatile quantum key distribution system with hardware key distillation and wavelength multiplexing. *New Journal of Physics*, *16*(1), 013, 047.
26. Wegman, M.N., & Carter, J. (1981). New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, *22*(3), 265–279.
27. Zhang, H.F., Wang, J., Cui, K., Luo, C.L., Lin, S.Z., Zhou, L., Liang, H., Chen, T.Y., Chen, K., & Pan, J.W. (2012). A real-time QKD system based on FPGA. *Journal of Lightwave Technology*, *30*(20), 3226–3234.

Jeremy Constantin received the B.Sc. degree in computational engineering from FAU Erlangen-Nürnberg, Germany, in 2007, and the M.Sc. degree in electrical engineering and information technology from ETH Zürich, Switzerland, in 2011.

He is currently pursuing the Ph.D. degree in electrical engineering with the Telecommunications Circuits Laboratory at the Ecole Polytechnique Federal de Lausanne (EPFL), Switzerland. His current research interests are in VLSI design, techniques for low-power processor design, and approximate computing.

Raphael Houlmann holds a degree in Electronics Engineering from the HE-Arc in Switzerland. He had various positions in the industry in Switzerland and Australia as an ASIC/FPGA designer. His current position is as FPGA specialist, part time at the Group of Applied Physics of the University of Geneva and part time at the University's spin-off ID Quantique. Quantum and classical cryptography are his main fields of interest.

Nicholas Preyss obtained the degree of Bachelor in Electrical Engineering and Information Technology and the degree of Master in Electrical Engineering and Information Technology at ETH Zurich in 2009 and 2011, respectively.

He is currently working towards a PhD degree at the Telecommunication Circuits Laboratory of EPF Lausanne. His research focus is on baseband signal processing and system design for wireless communication systems.

Nino Walenta received the Diplom degree in physics from the University of Potsdam (Germany) in 2007, and the Ph.D. degree in physics from the University of Geneva (Switzerland) in 2013. Currently, he is principle research scientist for quantum information and communication at Battelle Memorial Institute in Columbus (Ohio). His research has been concerned with quantum communication and quantum optics, with focus on implementations for quantum key distribution and quantum networks.

Hugo Zbinden has a PhD from the University of Berne, Switzerland. He is currently associate professor in the Group of Applied Physics of the University of Geneva and head of the Quantum Technologies group. His research has spanned various areas from optical sensors, single photon detectors, quantum communication and the foundations of quantum mechanics.

Hugo Zbinden has published more than 150 peer reviewed papers. He is associate editor of the Journal of Quantum Information and Communication.

Pascal Junod received a M.Sc. in computer science from ETH Zürich in 2000 and a Ph.D. in cryptography from EPF Lausanne in 2005. Presently, he is a professor of information security at the University of Applied Sciences and Arts Western Switzerland (HES-SO/HEIG-VD) in Yverdon-les-Bains (Switzerland). His research interests include applied cryptography, software security and ethical hacking.

Andreas Burg (S'97-M'05) was born in Munich, Germany, in 1975. He received his Dipl.-Ing. degree from the Swiss Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, in 2000, and the Dr. sc. techn. Degree from the Integrated Systems Laboratory of ETH Zurich, in 2006.

In 1998, he worked at Siemens Semiconductors, San Jose, CA. During his doctoral studies, he worked at Bell Labs Wireless Research for a total of one year. From 2006 to 2007, he held positions as postdoctoral researcher at the Integrated Systems Laboratory and at the Communication Theory Group of the ETH Zurich. In 2007 he co-founded Celestrius, an ETH-spinoff in the field of MIMO wireless communication, where he was responsible for the ASIC development as Director for VLSI. In January 2009, he joined ETH Zurich as SNF Assistant Professor and as head of the Signal Processing Circuits and Systems group at the Integrated Systems Laboratory. Since January 2011, he has been a Tenure Track Assistant Professor at the Ecole Polytechnique Federale de Lausanne (EPFL) where he is leading the Telecommunications Circuits Laboratory.

In 2000, Mr. Burg received the "Willi Studer Award" and the ETH Medal for his diploma and his diploma thesis, respectively. Mr. Burg was also awarded an ETH Medal for his Ph.D. dissertation in 2006. In 2008, he received a 4-years grant from the Swiss National Science Foundation (SNF) for an SNF Assistant Professorship.

He has served on the TPC of various conferences on VLSI, signal processing, and communications. He was a TPC co-chair for VLSI-SoC 2012 and is a TCP co-chair for SiPS 2017. He served as an Editor for the IEEE Transaction of Circuits and Systems in 2013 and is on the Editorial board of the Springer Microelectronics Journal and the MDPI Journal on Low Power Electronics and its Applications.