MindCrypt: The Brain as a Random Number Generator for SoC-Based Brain-Computer Interfaces

Guy Eichler
Dept. of Computer Science
Columbia University
New York, New York, USA
guyeichler@cs.columbia.edu

Biruk Seyoum
Dept. of Computer Science
Columbia University
New York, New York, USA
biruk@cs.columbia.edu

Kuan-Lin Chiu
Dept. of Computer Science
Columbia University
New York, New York, USA
chiu@cs.columbia.edu

Luca P. Carloni
Dept. of Computer Science
Columbia University
New York, New York, USA
luca@cs.columbia.edu

***Abstract*—True random number generation on resource-constrained devices is challenging due to inherent hardware limitations; these limitations affect the ability to find a reliable source of randomness with high throughput and sufficient entropy. As recent developments in the field of Brain-Computer Interfaces (BCI) suggest a wide range of future applications that require random numbers, we investigate the usability of electrocorticography-based neural data as seeds for random number generation. We develop algorithms that generate random bits from brain data and evaluate the quality of randomness by using the NIST SP 800-22 test suite. We implement the algorithms as hardware random bit generators (RBGs). Then, we integrate these implementations as hardware accelerators in MindCrypt, a heterogeneous System-on-Chip (SoC) that is equipped with a host processor to run BCI applications. In MindCrypt, applications use our RBG accelerators as random number generators (RNGs) and prime number generators. FPGA prototypes of MindCrypt running software applications on a RISC-V processor that invoke our accelerators show improvements of 376x in throughput and 4885x in energy efficiency compared to using state-of-the-art Linux-based RNGs. By transferring random bits with point-to-point (P2P) communication between the RBG accelerators and cryptographic accelerators, we gain 6.1x in performance and 12.4x in energy efficiency compared to direct memory access (DMA). Finally, we explore the efficacy of a partially reconfigurable FPGA implementation of MindCrypt that dynamically optimizes the throughput of random number generation in a resource-constrained BCI SoC.**

***Index Terms*—SoC, HLS, BCI, RISC-V, P2P, FPGA, DPR**

I. Introduction

As a subset of the Internet-of-Things (IoT) ecosystem, wearable brain-computer interfaces (BCI) are at the verge of taking a bigger part in everyday life [1]–[3]. Often constrained by area and power limitations, wearable real-time BCI devices need to execute diverse compute workloads for Machine Learning (ML) [4]–[6], adaptive control algorithms [7], [8], high-throughput secure communication [2], [9] and privacy preservation [10]. However, these applications require access to a true high-entropy source of randomness. For example, ML algorithms use randomness to generate better predictions and provide higher accuracy [5], [11] while security primitives use random cryptographic keys [12], [13].

In IoT devices, traditional ways of harvesting entropy and generating random numbers can become a bottleneck due to resources constraints and lightweight architectures [14]–[17]. For this reason, the use of Physically Unclonable Functions (PUFs) has become ubiquitous [18]–[20]. While adding a PUF to a device usually requires minimal additional hardware, classic PUFs rely on physical properties of the device that result from manufacturing processes and often suffer from sensitivity to varying environmental conditions that can lead to questionable reliability in random number generation [18]. As an alternative, sensor-based PUFs that rely on raw sensor data have been proposed [21]. A reliable, high-throughput, sensor-based PUF that takes advantage of sensor-based brain data offers robustness for wearable BCIs that are expected to function in a constantly changing environment.

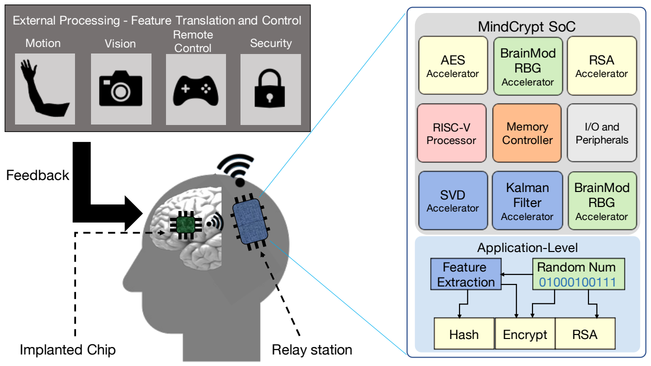

Fig. 1: MindCrypt SoC as part of a BCI-based IoT device.

To achieve this goal, we present MindCrypt, a configurable, programmable, and scalable solution that provides real-time random number generation for SoC-based wearable BCIs and their applications. A MindCrypt SoC integrates hardware accelerators embedded with a custom algorithm, BrainMod, which generates random bits from in-vivo electrocorticography sensor-based brain data [22]. Fig. 1 shows a potential real-world BCI system that is based on a MindCrypt SoC inside a wearable relay station, where the on-chip RBG accelerators are employed to provide random numbers to different BCI applications running on the IoT device.

We develop prototypes of the MindCrypt SoC on FPGA with various compositions of accelerators. With software applications running on a 64-bit CVA6 RISC-V processor [23], we evaluate random number generation and prime number generation compared to the Linux-based `/dev/urandom` and `rand()` RNGs. We implement a state-of-the-art brain-based RBG [24] in hardware, integrate it in MindCrypt, and compare it against BrainMod. By integrating crypto accelerators (AES, SHA2, RSA) in the SoC, we investigate the communication between the RBG accelerators and the crypto accelerators via point-to-point (P2P) and direct memory access (DMA).

In the case of an area-constrained implementation on FPGA, we use Dynamic Partial Reconfiguration (DPR) and multiplexing of accelerators in order to optimize the throughput of random number generation on the MindCrypt SoC.

Our results confirm that BCI devices, with access to real-time brain data, can achieve high quality randomness and efficient random number generation for various applications.

We break down our contribution into the following points:

1) The implementation and the statistical analysis of two BCI-based RBG algorithm.
2) Two hardware accelerators, BrainMod RBG and BrainBit RBG, that can be reconfigured at runtime to generate random numbers of different lengths from brain data.
3) Generation of both random numbers and prime numbers tested on real brain data that outperforms Linux-based random number generation.
4) The design of many-accelerator SoCs that support P2P and DPR, which demonstrate a dataflow of efficient random number generation for various applications.
5) The open-source release of MindCrypt to support other research projects in the BCI community.

II. Background

BCI System-Level Design. Modern BCI systems generally include three sub-systems: *(i)* an implanted neural interface used as a sensor, *(ii)* an intermediate mobile relay station located close to the human body that is used for preliminary computations on real-time data, and *(iii)* a distant machine with a wireless connection to the relay station for additional processing [1], [2], [9], [25], [26]. As the neural interface and the relay station of a BCI system are implemented on platforms with strict hardware constraints, they share the aforementioned limitations with most IoT devices. But unlike most IoT devices, BCI systems can potentially exploit an abundant source of randomness within reach: the brain.

The main guidelines when designing a wearable BCI system are: *(i)* provide enough resolution and throughput of brain data in order to extract meaningful information about neuronal activity [27]–[30], *(ii)* provide sufficient computational power to enable real-time execution of prediction algorithms within the boundaries of the reaction time of the brain [25], [31], and *(iii)* meet ultra-low-power constraints for a wearable device in a body-area network (BAN) [2], [32], [33].

BCI Applications and Randomness. Fig. 1 presents a BCI system that includes an implanted chip, a relay station, and an external device to support potential applications of BCI. The implanted chip is the neural interface and usually includes thousands of electrodes that can each record and stimulate small populations of neurons [29], [30], [34]. The chip should send the recordings wirelessly without losing information to the relay station [26], [28]. The relay station is an SoC that receives a large amount of brain data from the chip, executes signal processing to extract features, and provides real-time predictions of the intentions of the brain [25]. These predictions are sent to an external device for further processing. The external device controls other devices depending on the desired application. One example application is translating a motion intent from the motor cortex of an individual into moving a prosthesis [7]. Another example is reconstructing images from the visual cortex of the brain and even using a stimulation feedback to repair lost vision [35]. The system can also be used to control objects like drones and vehicles [36]. Finally, distinct features from brain data can be used as biometric passwords for secure authentication [37].

To be realized as real-world IoT, BCI systems need to support secure communication and ML-based computation [2]–[5], [9], [11]. Security primitives, such as AES [12] and RSA [13], are utilized by various applications and require random keys to enable information security and privacy [14]. Kalman Filter is a widely used online learning algorithm for continuous control in BCIs, but it needs to be periodically calibrated or to be used in combination with Reinforcement Learning (RL) [5], [6]. RL and other ML algorithms make heavy use of randomness to support adaptiveness [5], [6], [11].

Random Number Generation in IoT. In practice, random number generation (RNG), which is based on random bit generation (RBG), is done by using an algorithm that takes advantage of multiple existing sources and combines them to harvest entropy. For instance, Linux generates randomness from the recordings of system-level events and the timing of interrupts from various I/O sources [16], [17], [38]. In IoT, limited hardware resources and lack of user inputs make the generation of random numbers a challenging task [14], [15]. For that reason, traditional approaches of computationally heavy numerical methods and sampling hardware components are either not feasible or inefficient [11], [17].

One popular approach for random number generation in IoT is to use a Physical Unclonable Function (PUF) [19]–[21]. PUFs are based on manufacturing variations introduced during the fabrication of an integrated circuit (IC), which make it impossible to create an identical device with the same circuit characteristics. The PUF takes an input "challenge" and, by combining it with unique physical characteristics, outputs a "response" that cannot be replicated by any other device. A strong PUF that is reliable and able to generate a wide space of responses can be used as a RNG [20].

However, integrating PUFs in constrained systems is a challenging task due to unreliability caused by sensitivity to device aging and environmental noise (temperature and voltage) [18], [20], [39]. In contrast, wearable devices are expected to maintain correct function and reliability in a constantly changing environment. Sensor-based PUFs have been proposed for these devices [21], [40], [41]. These PUFs are lightweight and include the implementation of algorithms that generate random bits from a combination of sensor data. Depending on the algorithm and the sensor involved, this type of PUF can be used to build a more robust RNG.

The Brain as a Random Number Generator. Empirical evidence on the existence of chaos in the brain [42] suggests a certain level of unpredictability when recording interactions between neurons. When the resolution is lower and the analysis is made on bigger parts of the brain, unique patterns can be found to the level of authenticating an individual [37]. Experiments with neurophysiology [24] and EEG [43], [44] showed that by applying mathematical transformations as

```
1:  global variables: $\sigma, \mu, R, L, d, h$

2:  function BrainBit($x$)
3:      Initialize: $result = None$
4:      if $|x - \mu| \leq R\sigma$ then
5:          $y = x - (\mu - R\sigma)$
6:          $z = 2 \times R\sigma$
7:          $w = \lfloor (y/z) \times L) \rfloor$
8:          $result = w \mod 2$
9:      return $result$

10: function BrainMod($x$)
11:     Initialize: $result = None$
12:     if $|x - \mu| \leq R\sigma$ then
13:         $y = x - \mu$
14:         $z = y \times 2^d$
15:         $w = \lfloor z \rfloor \mod 2^h$
16:         $result = (\sum_h w_b) \mod 2$
17:     return $result$
```

Notations:
R, L, d, h are user defined parameters
μ is the estimated average over all values
σ is the estimated standard deviation over all values
w_b are the bits of the variable w

Fig. 2: Random bit generation algorithms.

RBG algorithms on fine-grained high-resolution brain data, we can retrieve streams of bits that pass statistical tests for randomness [45]. When produced with sufficient throughput, these bits can be used to construct cryptographic keys [24].

As we validate the randomness of bits that are generated from modern neural interfaces [26], we open a path for integrating a sensor-based PUF that relies on high-resolution brain data in BCI systems.

III. Randomness Analysis

Progress in the ability to record electrocorticography (ECoG) data directly from the brain was recently reported [22]. The results were stored in a database that holds the data recorded from 1024 electrodes (channels) attached to the visual cortex of the brain of a non-human primate. The data, which have been released in the public domain, can be used by any data analysis tool. With the availability of an ECoG database, we can implement RBG algorithms and run them on data from the database to generate random bits and test if they satisfy the state-of-the-art NIST statistical tests [45].

Brain-Based RBG Throughput. The basic throughput of a brain-based RBG depends on the recording ability from the neural interface. We analyzed data from the visual cortex of a non-human primate [22] and focused on local field potentials (LFP) [46]. We report the results of a subset that contains 78 million data points from 64 electrodes, which was recorded over a span of 41 min [22]. This translates to a throughput of one value per $0.03 msec$ from the neural interface, or a data rate of $33KHz$. However, the full dataset provides recordings from 1024 electrodes in the same amount of time, thus yielding a maximum data rate of $533KHz$. In general, neural interfaces are expected to be scalable and support simultaneous recording from even more electrodes [26], [30]. The rest of the throughput is determined by the RBG algorithm and its implementation, as discussed in the rest of the paper.

Algorithm for Random Bit Generation. As a baseline, we implemented the RBG algorithm described by Szczepanski *et al.* [24]. We named it BrainBit and described it in Fig. 2. BrainBit takes a data value x and calculates if the value does not deviate, beyond a specific threshold, from the estimated average of the dataset. The threshold is controlled by the estimated standard deviation, and by the user defined parameter R. Eventually, only the values within the range $[\mu - R\sigma, \mu + R\sigma]$

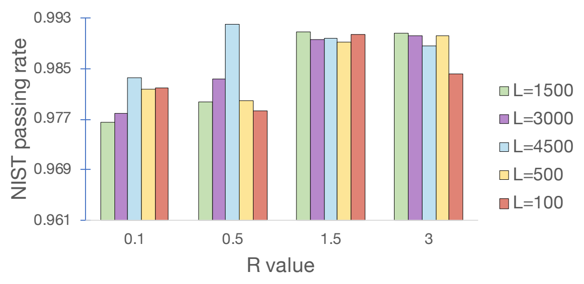

Fig. 3: Statistical analysis of the bit streams generated from BrainBit running on electrophysiological brain data.

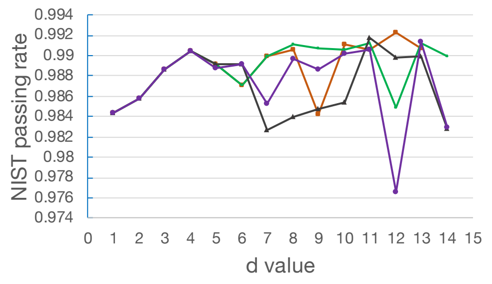

Fig. 4: Statistical analysis of the bit streams generated from BrainMod running on electrophysiological brain data.

are passed for bit computation. The computation is controlled by another parameter L, which sets how many sub-intervals the range $[\mu - R\sigma, \mu + R\sigma]$ will be divided into. In line 7 of Fig. 2, the variable w provides the number for the sub-interval where the value x is found. If the number is even, the result of the computation is 0 and if it is odd, the result is 1.

The BrainBit algorithm is based on large multiplications and divisions which might not be optimal for hardware implementations. For this reason, we created our own algorithm, BrainMod, which emphasizes hardware-efficient techniques. BrainMod is reported in Fig. 2. The algorithm begins by applying the same threshold control as in BrainBit. Then, it calculates the distance of x from the average μ, and multiplies it with 2^d. In line 15, the variable w holds the integer part of z after applying a modulo operation with 2^h. As a final step, the sum of the lower h bits in w is checked for oddness and the result is set accordingly. As BrainMod does not require any division operators, and it uses powers of 2 for multiplications and modulo operations, we expect it to provide better resource utilization in a hardware implementation.

Statistical Analysis. As mentioned previously, we ran BrainBit on the data from the chosen LFP dataset, which provided 78 million data points. We ran the RBG algorithms with different configurations to generate streams of bits. Then, we checked the randomness of the bit streams with the NIST SP 800-22 statistical test suite [45]. The suite outputs a pass or fail score for each type of test, and the number of tests passed for each type. In our analysis, we focus on configurations that provided a pass score for all the types of tests, and analyze the passing rate as the total sum of individual tests that passed.

Fig. 3 reports the results of the bit streams generated by BrainBit with respect to the values of R and L. All combinations of R, L provided high passing rates of over 96%, which match the standards of other RBGs [11]. For

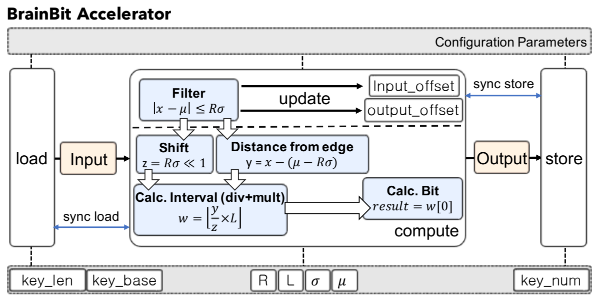
BrainBit Accelerator
Configuration Parameters
load
Input
Filter $|x-\mu| \le R\sigma$
update
Input_offset
output_offset
sync store
Shift $z = R\sigma \ll 1$
Distance from edge $y = x - (\mu - R\sigma)$
Calc. Interval (div+mult) $w = \left\lfloor \frac{y}{z} \times L \right\rfloor$
Calc. Bit $result = w[0]$
compute
Output
store
sync load
key_len
key_base
R
L
σ
μ
key_num

Fig. 5: The architecture of the BrainBit accelerator.

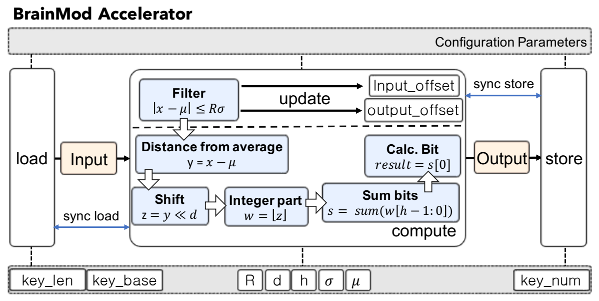

Fig. 6: The architecture of the BrainMod accelerator.

$R = 1.5$, all the values of L produced very similar passing rates of around 99%. For other values of R, the passing rates were less consistent and went down to a minimum of 97.6%. The number of values that passed the threshold condition of BrainBit (Fig. 2 line 4) changed significantly when we modified R. $R = 1.5$ yielded 91% bits from the original dataset, $R = 3$ yielded 99%, $R = 0.5$ yielded 49%, and $R = 0.1$ yielded only 10%. We conducted the experiments of Section VI with $R = 1.5$ and $L = 1500$, as they give a good compromise between passing rate and bit yield.

Fig. 4 shows the passing rate results of the bit streams generated by BrainMod with respect to the values of d and h when $R = 1.5$. Overall, the passing rate did not go below 97%, similarly to BrainBit, and meets the standards of other RBGs [11]. In BrainMod, the scaling factor 2^d can be implemented as a shift operation of size d, and the modulo with 2^h can be implemented by selecting the lower h bits of the integer part of z. For this reason, when d is small we can see that the results for different values of h are identical. For $d > 5$, the integer part of z (line 14 in Fig. 2) differentiates between the tested values of h and we could see more variations. When applying the algorithm, we aim at obtaining both good statistical results and good performance. For this reason, we used $\{h = 12, d = 7\}$ in our experiments.

In Section IV, we implement the RBG algorithms as hardware accelerators for better performance, better energy efficiency, and to remove the need of the processor in generating random numbers, which is preferred for IoT devices [15].

IV. Random Bit Generation Accelerators

Fig. 5 and 6 show the architectures of the RBG accelerators for BrainBit and BrainMod, respectively. We designed the accelerators in C/C++ and synthesized them with Vivado HLS. Our implementations extend the original algorithms (Fig. 2) to generate random bit streams of fixed length (keys).

The accelerators include private local memories (PLMs) to store inputs and outputs [47]. The PLMs are implemented as multi-bank memories that expose multiple read/write ports and their size is configurable at design time. The accelerators are configured to operate on a 32-bit fixed-point datatype.

Communication and Computation Configurability. To handle the communication with main memory and other accelerators in the SoC, both accelerators use configuration registers that set the size and structure of the input and output data. As such, `key_len` sets the length of a key to produce, while `key_base` indicates the upper bound for the total number of keys that can be generated from the data. `key_num` holds the number of keys the accelerator will produce consecutively. The configuration registers `R`, `L`, `d`, `h`, σ, and μ configure the computation for different datasets of brain data.

Data-Flow in the Accelerators. The architecture of both accelerators (Fig. 5 and 6) consists of three main functions operating concurrently in pipeline [48]. The *load* function loads a `key_len` size of 32-bit fixed-point data values from main memory into the input PLM. The *compute* function, which is fully pipelined to speedup the computation, implements the main algorithms. In both accelerators, the computation starts by checking each value stored in the input PLM for the initial filter condition (lines 4 and 12 in Fig. 2). If the value is within the configured range, i.e., $[\mu - R\sigma, \mu + R\sigma]$, then further computation is performed. Otherwise, the value is discarded.

In BrainBit (Fig. 5), the lower edge of the range is calculated and subtracted from the input value. The total size of the range is calculated by a shift operation on $R\sigma$. As part of the algorithm, the accelerator calculates w, which is the sub-interval within the range that holds the input value (Section III). The accelerator uses a 32-bit multiplier and a 32-bit divisor for the computation. Finally, the lowest bit of w is stored in the output PLM as a random bit and part of a new key.

On the other hand, BrainMod (Fig. 6) was created to avoid the utilization of a 32-bit multiplier and a 32-bit divisor that can increase the overall area and power on FPGA. First, the average μ is subtracted from the input value. Then, a 32-bit shift operator is used and the integer part is taken from the result. The lower h bits are summed, and the lowest bit of the sum is stored in the output PLM as part of a new key.

In both accelerators, after completing enough 32-bit words to assemble a key of `key_len` bit length, the *store* function sends the key. This process continues until `key_num` is satisfied. In order to accommodate 64-bit DMA requests, the loops inside *load* and *store* include an unrolled nested loop that reads/writes two 32-bit values in parallel. The PLM is partitioned in order to enable parallel read/write operations.

Input/Output Offset Control. The first step inside *compute* discards input values that do not meet the filter criteria (Fig 2 line 4). While *load* fetches only one `key_len` at a time, any discarded bit in *compute* requires an additional DMA read request to complete a key.

To handle this, the accelerators store offsets for the input and the output in local 32-bit registers, and use synchronization between the main functions (*load, compute, store*). If one value from the input was discarded, the input offset is incremented while the output offset remains the same. The computation continues until all the input values in the PLM are read. At this point, there is a mismatch between the input and the output offsets that triggers a signal from *compute* to initiate an additional DMA read request by *load*. A signal is also sent from *compute* to stall the next write request by *store*. After the next DMA read, the input offset is set at zero while the output offset is set according to the previously finished bits in the key. When the output offset reaches `key_len` the key is finished and a signal is sent to *store* to initiate a DMA write.

V. MindCrypt SoC

MindCrypt provides the infrastructure for a many-accelerator SoC that supports configurable random number generation for applications that run on a wearable BCI device. The architecture of MindCrypt allows the combination of multiple accelerators with a RISC-V processor, a memory channel, and an I/O interface in a single SoC. A typical MindCrypt SoC combines one or more RBG accelerators with other accelerators that implement computation kernels and benefit from a high throughput of random numbers.

Target System. Fig. 1 describes a potential BCI system that is powered by a MindCrypt SoC. An implanted chip uses a large array of electrodes to record the electrical activity of neurons on the outer cortex of the brain. By using a low-power short-range wireless connection, the chip sends the recorded brain data to an external relay station which has been designed with a MindCrypt SoC. As most BCI computations are based on learning and then predicting the intention of the brain, we include efficient accelerators for Kalman Filter and Singular Value Decomposition (SVD) for feature extraction in real time [4], [5]. To extract the features, applications running on the device use random numbers generated by brain-based RBGs. Additional processing of the features by an external device is required in order to actuate and correct disabilities with a feedback mechanism. To this end, secure communication of the features to another device is done by encrypting them with an AES accelerator that uses brain-based random keys generated by one of the RBGs. Then, the RBG provides the AES keys to an on-chip RSA accelerator so that it can be sent for external processing via asymmetric encryption. The ciphertext is sent to the external device, which decrypts it with the brain-based key.

SoC Design Platform. We implemented MindCrypt by leveraging ESP, an open-source platform for the design of heterogeneous SoCs and their prototyping on FPGA [49]. ESP provides the template for a tile-based architecture that can be configured and specialized with the desired mix and placement of accelerator tiles, processor tiles, memory channel tiles and an I/O tile that manages various peripherals. ESP features a multi-plane network-on-chip (NoC) as the main on-chip interconnect. We selected the 64-bit CVA6 RISC-V core [23] as the host processor.

Accelerator Communication and Invocation. The accelerator tiles follow a loosely-coupled accelerator model [50]. The RBG accelerators were designed independently from the rest of the system so they can run separately from the processor [15]. In Section VI, we implement MindCrypt SoCs with crypto accelerators which were designed with Catapult HLS and were provided to us by the authors of HARDROID [51].

TABLE I: Resource and Power consumption in MindCrypt SoC.

Component	LUT	FF	BRAM	DSP	Power[W]	Time[ms]
BrainBit	11441	6730	4	22	0.068	89.8
BrainMod	5929	2379	4	16	0.01	88.3
CVA6	56191	35752	36	27	0.128	N/A
AES	69075	28290	14	3	0.108	4.9
RSA	126973	57941	0	0	0.277	6874.4
SHA2	32796	20756	2	0	0.408	5.1

*Execution time is for 1536 bits

Following the integration, the accelerators are connected directly to the system interconnect via a configurable tile socket [52], which is provided by ESP. The socket is configured through memory-mapped registers and handles address translation, cache coherence, DMA, and P2P communication.

P2P communication enables direct data transfer between the accelerators [53] and isolates their computation from other system components such as the processor and main memory. To take advantage of the P2P feature, we designed our RBG accelerators to work independently on arbitrary-size inputs with minimum software assistance. We also modified the crypto accelerators to support a similar feature. Enabling P2P results in a reduction of off-chip memory accesses, and consequent performance improvements and energy savings, especially when high encryption throughput is needed and multiple instances of the accelerators are used (Section VI).

VI. Evaluation on FPGA

Experimental Setup. We evaluated our MindCrypt SoCs on the Xilinx Virtex UltraScale+ VCU118 FPGA with a clock frequency of $78MHz$, which is the frequency set by the ESP platform. We developed bare-metal and Linux software applications that run on the CVA6 RISC-V processor and leverage the ESP run-time API to invoke our RBG accelerators and the crypto accelerators. For each accelerator, the applications allocate buffers whose sizes match the accelerators' data footprint. Table I reports the FPGA resources and power consumption of the accelerators and the processor (as given by post-implementation reports from Xilinx Vivado) and the execution time of each accelerator in isolation on FPGA. The RBG accelerators were running a task to generate an output of 1536 random bits, while the crypto accelerators ran for an input size of 1536 bits. We chose this specific input size as it corresponds to a key size of 1024-bit and plaintext size of 16 32-bit words for RSA. For a fair comparison, we set the same input size also for the other crypto accelerators, and matched the output size of the RBG accelerators accordingly. The power consumption of BrainMod is almost $7\times$ lower than that of BrainBit. This result was expected because of the more hardware-oriented features of BrainMod (Section III).

Performance of the RBG Accelerators. The average time that takes to read a value from memory, send it to one RBG accelerator, and generate one random bit is $0.05msec$ (Table I). While the neural interface is actively recording, the throughput of random numbers will be determined by the number of instances of the RBG accelerators and their latency. More accelerators' instances can support a higher throughput of random numbers to meet the needs of different applications.

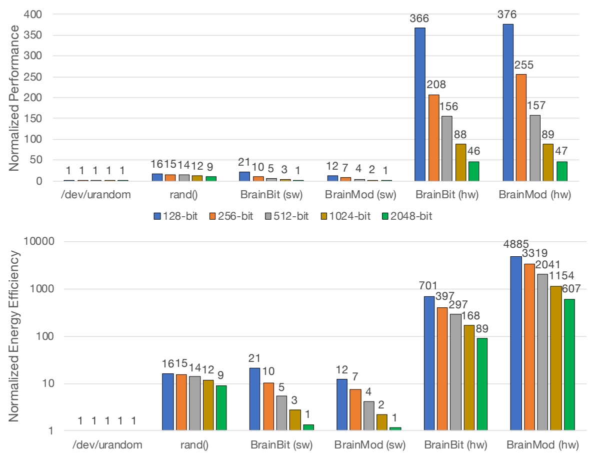

Fig. 7: Performance and energy efficiency of RNGs.

We compared the generation of different bit lengths by one instance of each of our RBG accelerators against equivalent software implementations (written in C) running on the CVA6 processor. We also evaluated the same generation by two Linux-based generators: (1) `/dev/urandom` [17] and (2) the `rand()` function in C. The interface to `/dev/urandom` provides the option to request any specific number of random bytes, while `rand()` provides 32-bit words, which we concatenated into the specific bit lengths. Fig. 7 depicts our results normalized to the performance of `/dev/urandom`. The RBG accelerators provide a maximum of 376× better performance for 128-bit, and a minimum of 46× for 2048-bit. For 128-bit, BrainMod provides a maximum of 4885× better energy efficiency, and a minimum of 607× for 2048-bit. BrainMod provides 7× better energy efficiency over BrainBit.

Random Prime Number Generation. As the RSA execution time is the largest among the accelerators (Table I), the effective throughput of the system can depend on the number of RSA instances in the system and the utilization of the RSA accelerator. Since RSA depends on the availability of prime numbers, we investigated our RBG accelerators as prime number generators. This can also provide more information about the entropy and quality of the brain as a source for randomness in combination with BrainBit and BrainMod.

We configured the accelerators to generate 1000 256-bit, 512-bit and 1024-bit random numbers and implemented the Miller-Rabin probabilistic primality test [54] to run on the CVA6 processor. We also performed the same test on numbers generated using `/dev/urandom` and `rand()`. We repeated the test for 100 iterations. The average amount of prime numbers generated from each RNG is summarized in Table II. The amount of prime numbers generated by our RBG accelerators is equivalent to `/dev/urandom`. The amount produced by `rand()` is slightly higher but we observed obvious patterns and a lower quality of randomness. Fig. 8 presents the average time to generate a prime number from each of the RNGs in log scale. For 256-bit, our RBG accelerators provide an average gain in throughput of 368× over `/dev/urandom` and 11× over `rand()`. For 512-bit, the speedup provided by the RBG accelerators is 214× over `/dev/urandom` and for 1024-bit the maximum speedup is 70×. Compared to `rand()`, the

TABLE II: Average Amount of Prime Numbers / 1000 Numbers

bit length	BrainBit	BrainMod	/dev/urandom	rand()
256-bit	5.47	5.51	5.6	11
512-bit	2.87	2.83	2.59	6.65
1024-bit	1.16	1.46	1.08	2.74

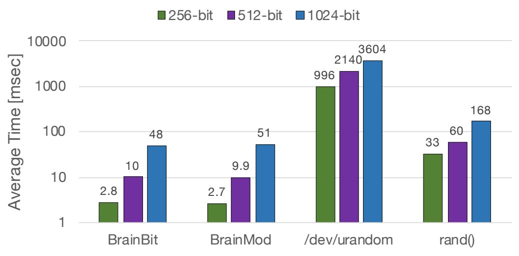

Fig. 8: Average time to generate a prime number.

throughput is increased by 6× and by 3× for 512-bit and 1024-bit, respectively.

Overall, the RBG accelerators provide a significant performance advantage over the other RNGs. The results demonstrate the potential of MindCrypt as a viable prime number generation engine for asymmetric cryptography algorithms like RSA, which rely on the availability of large prime numbers.

Performance and Efficiency of BrainBit-AES. To investigate the encryption throughput MindCrypt provides, we invoked both the RBG accelerators and the AES accelerator on FPGA. We evaluated the execution of their joined computation by scaling up the number of accelerators of each type. We ran our tests on MindCrypt SoCs containing up to four accelerators per type, and compared the cases of communication via P2P and communication via DMA.

Fig. 9 shows the normalized performance and energy efficiency of invoking the accelerators. Each AES received a total of 256-bit, 512-bit, or 1024-bit random bits, while the first 128 bits were used as encryption keys and the rest as 32-bit plaintext words. For each combination of accelerators, the results were normalized with respect to the result of one RBG accelerator and one AES communicating via DMA. We report the results from unique configurations with different number of instances for each accelerator type.

For the case of one accelerator of each type, enabling P2P provides 1.7× speedup, and 4.4× better energy-efficiency (BrainMod). For 1024-bit and with 4 instances of the RBG accelerators we achieve a maximum of 6.1× speedup and 12.4× energy-efficiency. Instead, adding more instances of AES gives moderate speedup and energy-efficiency gain. This is expected because of the execution time of AES, which is shorter than the one for the RBG accelerators (Table I).

The results show that in addition to the security advantages of isolating the generated random bits from off-chip memory and the processor, using P2P is beneficial for performance and energy efficiency. We expect P2P to show similar results when experimenting with other crypto accelerators as well.

Throughput Optimization with DPR. Environmental factors such as temperature, noise, etc., can affect the performance of the neural interface. This results in some of the brain data being discarded by the RBG accelerators leading to a reduction of the total throughput (Section IV). To confront this, we would like to increase the throughput by having

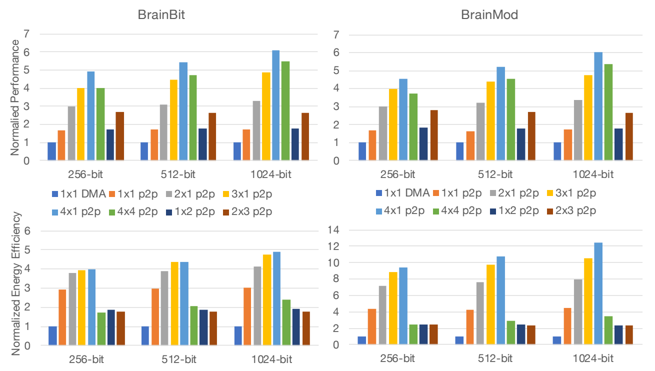

Fig. 9: Performance and energy efficiency for different SoCs with RBG×AES accelerators communicating via P2P.

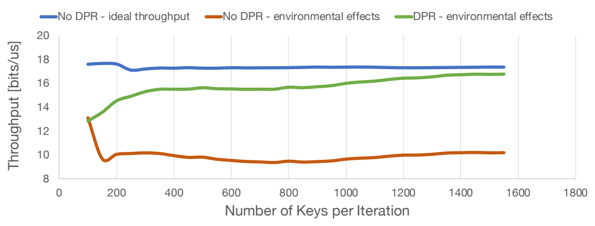

Fig. 10: DPR throughput vs. no DPR.

an SoC with multiple instances of RBG accelerators reading brain data from more channels. However, BCI-based IoT are constrained in area and resources to host multiple accelerators. As we target the relay-station part of a BCI system (Fig. 1) which can be implemented on FPGA, we augmented an FPGA implementation of the MindCrypt SoC with a DPR flow to multiplex BrainMod accelerators with other accelerators in the same FPGA area [55]. When a lower RBG throughput is detected, our software triggers a partial reconfiguration which temporarily replaces one of the less frequently used tiles with an additional BrainMod accelerator. As long as the power consumption of BrainMod is lower than the one of the other accelerator, the total power consumption will not be harmed.

Fig. 10 shows our evaluation on the performance of DPR with the generation of different numbers of keys for multiple iterations. The brain data was altered in 60% of the execution time to simulate a low throughput. For less keys, DPR is not effective and the RBG throughput remains as in the case with no DPR. For higher numbers of keys, DPR improves the throughput, which almost reaches the ideal value with no environmental effects. These results show that in the case of an FPGA-based implementation, DPR can be highly effective in maintaining the throughput of MindCrypt, while having a negligible impact on the energy efficiency [56].

VII. Related Work

Random number generation in IoT has been emphasized in literature [11], [14], [15], [39], [40]. Kietzmann *et al.* [11] investigate the generation of random numbers for various purposes by software and hardware generators from the perspective of an operating system (OS) on an IoT device. They emphasize the importance of random number generation for emerging AI and security applications on IoT devices, and discuss the growing processing demands of ML and RL in IoT, which triggers the use of hardware platforms that consist of low-power RISC-V processors and hardware accelerators. In Section VI, by running software applications on a RISC-V processor, we show that our hardware accelerators provide superior random number generation in terms of throughput and energy efficiency compared to Linux-based software RNGs.

Authentication and random number generation in IoT has been enabled via PUFs that are based on ring oscillators, radio frequency, power distribution, and more [18]–[20], [39]. However, to eliminate the need in modifying the fabrication of the IC and to increase reliability, harvesting entropy from sensor-based data has been proposed instead [21], [40], [41]. Hillerström *et al.* [21] propose to build a sensor-based PUF by combining raw data from multiple sensors and applying randomness extraction algorithms to transform it into uniformly distributed sequences. Wallace *et al.* [40] build the SensoRNG framework which collects data from on-board sensors to produce random numbers via a mixing algorithm. In Section III, we apply our RBG extraction algorithms on brain data recorded from multiple sensors (electrodes) to generate uniformly random bits.

The design of BCI systems as secure healthcare wearable devices has been discussed [1]–[3], [9], [33]. Taleb *et al.* [33] explain that security and privacy are indispensable requirements for devices in wireless body-area networks (WBANs). Lin *et. al.* [37] show how online brain data can provide authentication. Szczepanski *et al.* [24] present the only existing algorithm in literature that converts neurophysiological brain data to pseudo-random bits, and test it on brain data from a cat. They suggest that in the future the random bits can be used to generate cryptographic keys. In Section III we implement BrainBit based on the description from Szczepanski *et al.* [24]. Then, we develop the hardware-efficient BrainMod algorithm.

BCI algorithms based on adaptive ML are more successful in decoding the brain intent as the neural interface and the neural activity change over time [3]–[6]. Degenhart *et al.* [5] align neural activity from the brain of a non-human primate to body movements via ML. The implementation runs a stabilizer to support a Kalman Filter and the tuning of the model uses random values in different stages. Zhang *et al.* [6] implement a RL-based Kalman Filter, while RL is known to make heavy use of random numbers due to the use of randomized algorithms [11]. In Section IV we portray the implementation of our RBG algorithms as reconfigurable hardware accelerators. In Section VI, we show that our accelerators provide better throughput for random number generation compared to other RNGs, and, therefore, are more suitable to generate random numbers for adaptive ML applications.

To the best of our knowledge, MindCrypt is the first work that takes real brain data and realizes a full heterogeneous SoC that can provide random numbers for various applications on BCI-based IoT devices.

VIII. Conclusion

We designed MindCrypt to advance the research on brain-based random number generation to be used by applications for brain-computer interfaces. MindCrypt integrates RBG accelerators in a scalable SoC architecture, and provides a complete flow, from the availability of brain data through random bit generation to configurable random number generation, which can be mapped to high-level applications via efficient communication and resource utilization. We have released the design of MindCrypt in the public domain[1].

IX. Acknowledgements

This research was developed, in part, with funding from the Defense Advanced Research Projects Agency (DARPA), and in part with funding from the National Science Foundation (A#: 1764000). The views, opinions and/or other findings expressed are those of the authors and should not be interpreted as representing the official views or policies (either expressed or implied) of the Department of Defense, the National Science Foundation, or the U.S. Government. Distribution Statement "A": Approved for Public Release, Distribution Unlimited. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] E. Musk *et al.*, "An Integrated Brain-Machine Interface Platform with Thousands of Channels," *JMIR*, 2019.

[2] G. Udovičić *et al.*, "Wearable Technologies for Smart Environments: A Review with Emphasis on BCI," in *Proc. of SoftCOM*, 2016.

[3] W. Byun *et al.*, "Advances in Wearable Brain-Computer Interfaces From an Algorithm-Hardware Co-Design Perspective," *TCAS-II*, 2022.

[4] J. Lee *et al.*, "Hierarchical Optimal Transport for Multimodal Distribution Alignment," in *Proc. of NeurIPS*, 2019.

[5] A. D. Degenhart *et al.*, "Stabilization of a Brain–Computer Interface via the Alignment of Low-dimensional Spaces of Neural Activity," *Nature biomedical engineering*, 2020.

[6] X. Zhang *et al.*, "Reinforcement Learning-based Kalman Filter for Adaptive Brain Control in Brain-Machine Interface," in *Proc. of EMBC*, 2021.

[7] M. Vilela *et al.*, "Applications of Brain-Computer Interfaces to the Control of Robotic and Prosthetic Arms," *Handbook of clinical neurology*, 2020.

[8] C. Yang *et al.*, "Mind Control of a Robotic Arm with Visual Fusion Technology," *TII*, 2017.

[9] Q. Li *et al.*, "Brain-Computer Interface applications: Security and privacy challenges," in *Proc. of CNS*, 2015.

[10] K. Xia *et al.*, "Privacy-preserving Brain-Computer Interfaces: A Systematic Review," *TCSS*, 2022.

[11] P. Kietzmann *et al.*, "A Guideline on Pseudorandom Number Generation (PRNG) in the IoT," *CSUR*, 2021.

[12] X. Zhang *et al.*, "Implementation Approaches for the Advanced Encryption Standard Algorithm," *CASS*, 2002.

[13] D. Boneh *et al.*, "Efficient Generation of Shared RSA Keys," in *Proc. of CRYPTO*, 1997.

[14] F. Meneghello *et al.*, "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices," *IEEE Internet of Things Journal*, 2019.

[15] M. O. Ojo *et al.*, "A Review of Low-End, Middle-End, and High-End IoT Devices," *IEEE Access*, 2018.

[16] P. Lacharme *et al.*, "The Linux Pseudorandom Number Generator Revisited," *Cryptology ePrint Archive*, 2012.

[17] Z. Gutterman *et al.*, "Analysis of the Linux Random Number Generator," in *Proc. of S&P*, 2006.

[18] Y. Gao *et al.*, "Physical unclonable functions," *Nature Electronics*, 2020.

[19] P. K. Sadhu *et al.*, "MC-PUF: A Robust Lightweight Controlled Physical Unclonable Function for Resource Constrained Environments," in *Proc. of ISVLSI*, 2022.

[20] B. Chatterjee *et al.*, "RF-PUF: Enhancing IoT Security through Authentication of Wireless Nodes Using In-Situ Machine Learning," *IoT-J*, 2018.

[21] M. Hillerström *et al.*, "Sensor-Based PUF: A Lightweight Random Number Generator for Resource Constrained IoT Devices," in *Proc. of IFIPIoT*, 2022.

[22] X. Chen *et al.*, "1024-channel Electrophysiological Recordings in Macaque V1 and V4 During Resting State," *Scientific data*, 2022.

[23] F. Zaruba *et al.*, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," 2019.

[24] J. Szczepanski *et al.*, "Biometric Random Number Generators," *Computers Security*, 2004.

[25] G. Eichler *et al.*, "MasterMind: Many-Accelerator SoC Architecture for Real-Time Brain-Computer Interfaces," in *Proc. of ICCD*, 2021.

[26] Y. Wang *et al.*, "Implantable Intracortical Microelectrodes: Reviewing the Present with a Focus on the Future," *Microsystems & Nanoengineering*, 2023.

[27] D. Tsai *et al.*, "A Very Large-scale Microelectrode Array for Cellular-Resolution Electrophysiology," *Nature Communications*, 2017.

[28] A. Vázquez-Guardado *et al.*, "Recent Advances in Neurotechnologies with Broad Potential for Neuroscience Research," *Nature neuroscience*, 2020.

[29] N. Zeng *et al.*, "A Wireless, Mechanically Flexible, 25m-Thick, 65,536-Channel Subdural Surface Recording and Stimulating Microelectrode Array with Integrated Antennas," in *Proc. of VLSI*, 2023.

[30] C. Lopez *et al.*, "Design and Fabrication of CMOS-based Neural Probes for Large-scale Electrophysiology," in *Proc. of IEDM*, 2019.

[31] R. J. Kosinski, "A Literature Review on Reaction Time," *Clemson University*, 2008.

[32] A. Y. Dogan *et al.*, "Multi-core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems," in *Proc. of DATE*, 2012.

[33] H. Taleb *et al.*, "Wireless Technologies, Medical Applications and Future Challenges in WBAN: A Survey," *Wireless Networks*, 2021.

[34] I. Rachinskiy *et al.*, "High-Density, Actively Multiplexed μECoG Array on Reinforced Silicone Substrate," *Frontiers in nanotechnology*, 2022.

[35] A. Lozano *et al.*, "Neurolight: A Deep Learning Neural Interface for Cortical Visual Prostheses," *Int J Neural Syst*, 2020.

[36] A. Hekmatmanesh *et al.*, "Review of the State-of-the-Art of Brain-Controlled Vehicles," *IEEE Access*, 2021.

[37] L. Feng *et al.*, "Brain Password: A Secure and Truly Cancelable Brain Biometrics for Smart Headwear," in *Proc. of MobiSys*, 2018.

[38] N. Ferguson *et al.*, *Practical Cryptography*. Wiley New York, 2003.

[39] B. Sen, "PUF: A New Era in IoT Security," *CSI Transactions on ICT*, 2020.

[40] K. Wallace *et al.*, "Toward Sensor-Based Random Number Generation for Mobile and IoT Devices," *IEEE Internet of Things Journal*, 2016.

[41] M. P. Pawlowski *et al.*, "Harvesting Entropy for Random Number Generation for Internet of Things Constrained Devices Using On-board Sensors," *Sensors*, 2015.

[42] H. Korn and P. Faure, "Is there Chaos in the Brain? II. Experimental Evidence and Related Models," *Comptes Rendus Biologies*, 2003.

[43] C. Guangyi, "Are Electroencephalogram (EEG) Signals Pseudo-Random Number Generators?" *Journal of Computational and Applied Mathematics*, 2014.

[44] D. Nguyen *et al.*, "EEG-Based Random Number Generators," in *NSS*, 2017.

[45] L. E. Bassham III *et al.*, *Sp 800-22 rev. 1a. a Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. NIST, 2010.

[46] L. S. Kumari *et al.*, "TinyLFP: A Tiny Local-field-potential Sensor," *T-MRB*, 2022.

[47] C. Pilato *et al.*, "System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip," *TCAD*, 2017.

[48] L. Piccolboni *et al.*, "COSMOS: Coordination of High-Level Synthesis and Memory Optimization for Hardware Accelerators," *TECS*, 2017.

[49] P. Mantovani *et al.*, "Agile SoC Development with Open ESP," in *Proc. of ICCAD*, 2020.

[50] J. Cong *et al.*, "Architecture Support for Accelerator-Rich CMPs," in *Proc. of DAC*, 2012.

[51] L. Piccolboni *et al.*, "HARDROID: Transparent Integration of Crypto Accelerators in Android," in *Proc. of HPEC*, 2021.

[52] L. P. Carloni, "The Case for Embedded Scalable Platforms," in *Proc. of DAC*, 2016.

[53] D. Giri *et al.*, "ESP4ML: Platform-Based Design of Systems-on-Chip for Embedded Machine Learning," in *Proc. of DATE*, 2020.

[54] S. Ishmukhametov and B. Mubarakov, "On Practical Aspects of the Miller-Rabin Primality Test," *Lobachevskii J. Math*, 2013.

[55] B. Seyoum *et al.*, "PR-ESP: An Open-Source Platform for Design and Programming of Partially Reconfigurable SoCs," in *Proc. of DATE*, 2023.

[56] A. Nafkha *et al.*, "Accurate Measurement of Power Consumption Overhead during FPGA Dynamic Partial Reconfiguration," in *Proc. of ISWCS*, 2016.

[1] https://github.com/GuyEichler/esp/tree/mindcrypt