Challenges of Implementing Cyber-Physical Security Solutions in Body Area Networks

A. Banerjee, K. Venkatasubramanian and Sandeep K.S. Gupta
IMPACT Lab (http://impact.asu.edu)
School of Computing and Informatics
Arizona State University
Tempe, Arizona, USA
{abanerj3, kkv, sandeep.gupta}@asu.edu

ABSTRACT

Cyber-physical approach to securing Body Area Networks (BANs) provides solutions that are plug-n-play and transparent to its users. These *Cyber-Physical Security Solutions* (CPSS) actively involve characteristics from the physical environment. As a result they require a combination of signal processing (to extract features from the physical environment) and security primitives to function. In this paper we outline some of our experiences while implementing Plethysmogram based Key Agreement (PKA) - a CPSS - that uses Photoplethysmogram (PPG) based features for key agreement. Given the limited capabilities (computation, memory, power) of individual sensor nodes in a BAN, implementing CPSS for them is challenging. We therefore design Field Programmable Gate Array (FPGA) based hardware add-on for sensor nodes in a BAN, dedicated to execute PKA. The main contributions of this work are: 1) description of our experiences in implementation of PKA on FPGA platform; 2) identification of the design goals and trade-offs for validating implementation of CPSS; and 3) a discussion on the feasibility of a software implementation of PKA based on our experience gained from the FPGA prototyping.

1. INTRODUCTION

Recent technological advances in the fields of MEMS, integrated circuits, and low power design have lead to the development of health monitoring Body Area Networks (**BAN**) [10] [14]. BAN (as shown in Figure 1) is a collection of medical and ambient sensor nodes, deployed on a person (host), to continuously monitor their health parameters. These sensors collect data from their host and send it to a base station for processing and storage, usually through a wireless multihop network [21]. As BANs deal with sensitive and personal information, providing security to the inter-sensor communication within a BAN is important. Indeed the Health Insurance Portability and Accountability Act (HIPAA) mandates securing all electronically transferred health information (http://www.hhs.gov/ocr/hipaa/).

BodyNets'09, April 1–3, 2009, Los Angeles, CA.

Further, BANs are often used in mission critical contexts - necessitating them to operate unobtrusively by employing *usable* security solutions that are plug-n-play in nature and are largely transparent to the users (patients as well as caregivers) of the BAN [24]. Recent years have seen emergence of BAN security solutions with usability as the primary goal [20] [24]. These solutions take a *cyber-physical* approach to security, i.e. they make parameters derived from their physical environment - an integral part of their operation. This new class of security solutions - **Cyber Physical Security Solutions** (CPSS) - integrate signal processing with cryptographic primitives in order to provide usable security solutions in BANs. This dual requirement of cryptography and signal processing however introduces several implementation challenges in the resource constrained BANs with simple node architecture, small storage and limited energy [16].

In this paper, we study the challenges faced and trade-offs incurred in implementing a CPSS in a BAN environment. In this regard, we implement Plethysmogram based Key Agreement protocol (**PKA**) - a CPSS - first discussed in [24] in the BAN of the Ayushman [21] pervasive health monitoring platform being developed in the IMPACT Lab at Arizona State University. Implementing the signal processing requirements of PKA in the sensor nodes (Crossbow motes) is complex given their limited capabilities. A popular technique to increment the mote's computing resources is to interface Field Programmable Gate Array (FPGA) based custom boards to it as suggested in [2] [7] [17]. In this work we design such an FPGA based hardware add-on (using VHDL) for sensor nodes in the Ayushman BAN that will execute PKA.

To validate our implementation we identified a set of **design goals** that PKA has to meet: 1) **Accuracy**: This design goal ensures that the approximations made during the implementation of CPSS in any platform do not lead to loss of security. For example, in the case of PKA we have to guarantee that it enables plug-n-play key agreement between sensor nodes in a BAN without allowing adversaries to identify the key; 2) **Low Latency**: The implementation should be efficient in terms of latency of operation. This is an important aspect for CPSS as it is usually implemented on systems which are time-critical in nature, such as BANs; and 3) **Minimal Resource Usage**: This ensures the feasibility of implementing CPSS in terms of the resources utilized by the implementation. We believe that these goals can be used for validating any CPSS.

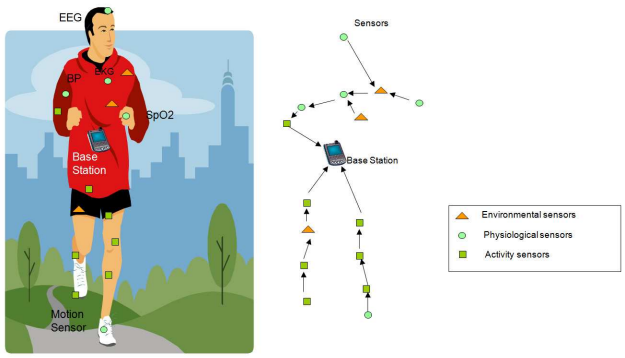

Figure 1: Body Area Network with Environmental and Physiological sensors

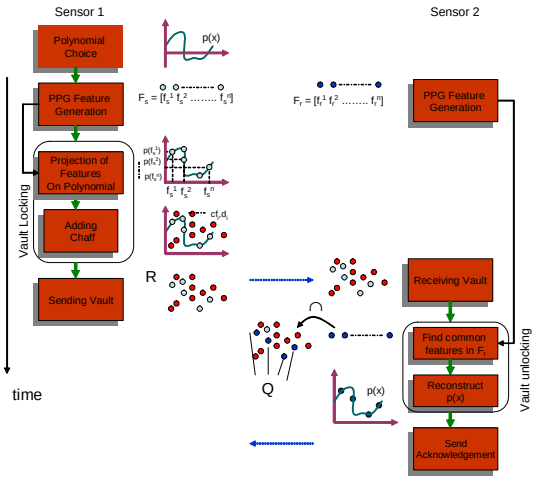

Figure 2: Key Agreement with PKA Protocol [24]

The main **contribution** of the paper is an enumeration of our experiences - challenges, design goals and trade-offs - in implementing a CPSS, based on an FPGA based implementation of PKA. Based on our experiences from the implementation we also discuss the feasibility of a software implementation of PKA. The availability of a viable software implementation of PKA is advantageous as it reduces the overall cost of out-fitting every sensor in the BAN with additional hardware. To the best of our knowledge this is the first work which discusses implementation issues specific to cyber-physical security primitives.

The rest of the paper is organized as follows: Section 2 presents the preliminaries: a description of PKA, and our approach to implementing it. Section 3 discusses the challenges faced in VHDL implementation of PKA CPSS, followed by Section 4 which evaluates the VHDL implementation of PKA in terms of the design goals and brings forth the trade-offs incurred in the implementation process. Section 5 discusses the possible challenges of implementing a CPSS on a software platform. Section 6 presents the related work, followed by Section 7 which concludes the paper.

2. PRELIMINARIES

In this section we provide an overview of PKA [24], then describe our approach to implementing PKA on FPGA and finally provide details on the data collection methodology.

2.1 Plethysmogram based Key Agreement

Plethysmogram based Key Agreement protocol (PKA) takes a cyber-physical approach to security. Figure 2 illustrates the basic operation of PKA.

- Initially, both the sensor nodes measure the host's PPG signal. A set of frequency domain features are then generated by performing a windowed FFT on the PPG signals and then detecting the peaks in the FFT coefficients. Each of the peak-index and its corresponding peak-value (magnitude) are recorded as tuples and combined to form the features. Frequency domain features are used in order to reduce the need for synchronizing the measurement of physiological signals at the sensor nodes [20].

- These tuples (features) are then quantized and concatenated to form a feature vector. Once the feature vectors have been generated, one of the two communicating sensor nodes (designated as the *sender*) generates a random symmetric key (128bits long, longer keys can also be used) which it then hides using the elements (features) of the feature vector obtained from the PPG signal.

- For the purpose of hiding the key, a *fuzzy vault* [8] cryptographic primitive is used. The hiding process works as follows - 1) The sender generates a vth order polynomial, the coefficients of which are populated by the secret key which is to be hidden. 2) It then computes the polynomial at each of the elements in the feature vector. 3) Each element in the feature vector and its projection on the polynomial forms a set of legitimate coordinates of the form (x-value, y-value). 4) The legitimate points are then obscured by adding a large number of bogus coordinates called the *chaff points*. The chaff points are random in nature, their x-values are not a part of the feature vector, nor are their y-values an accurate projection of the x-values on the polynomial. This set of legitimate points and chaff points is called a fuzzy vault.

- The vault is then transmitted to the other sensor node (designated as the *receiver*) via the wireless medium. The receiver upon receiving the vault, identifies $v+1$ elements from the vault whose x-values are identical to its own features and tries to re-construct the polynomial hiding the secret key using Lagrangian interpolation.

- Once the receiver has generated the correct polynomial it can obtain the hidden key from the coefficients. It then sends back an acknowledgment which is basically a Message Authentication Code (MAC) computed on a random number using the newly agreed upon key.

Any adversary eavesdropping on this conversation cannot distinguish the legitimate points from the chaff and has to potentially try all possible combinations of size $v+1$ from the vault to arrive at the correct key, which can be prohibitively expensive [24].

2.2 Approach

The original design of PKA was verified by extensive Matlab simulations based on actual data, and the results were presented in [24]. However, Matlab eases many of the challenges involved in implementing signal processing primitives in the sensor nodes of a BAN. Limited capabilities of the sensor nodes make the implementation of PKA in a BAN, a non-trivial task. In this work, we therefore explore the design of a specialized hardware that can be interfaced with individual sensor nodes in a BAN, to which the computational requirements of PKA can be transfered. (We also talk about direct implementation of PKA on motes in Section 5.) We utilize Field Programmable Gate Array (FPGA) boards for prototyping our design[1]. We used the XILINX Spartan II XC18V02 FPGA platform for this purpose. The hardware board has - a separate crystal clock that runs at 20 MHz (but is capable of scaling to a maximum clock speed of 200 MHz), and 56K block RAM where each block RAM is 4 Kb in size. The VHDL programming platform was used to perform the prototype implementation of PKA. We used ModelSim PE as the simulation software for the VHDL implementation.

2.3 Physiological Data Collection

Before we go into the details of VHDL implementation, we briefly review the physiological signal (PPG in our case) collection process as described in [24]. The data forms the basis of validating our VHDL implementation of PKA and its comparison with our benchmark Matlab implementation. We collected PPG data from a group of 10 volunteers in the IMPACT lab. We used Smith Medical pulse oximeter boards (specifications can be found at http://www.smithsoem.com/ applications/ oxiboards.htm) to collect the PPG data from the volunteers. The volunteers were asked to sit upright with their hand firmly placed on a desk; an oximeter sensor was placed on the index finger of each hand. Data was collected for about five minutes from each subject at a sampling rate of 60Hz.

3. VHDL IMPLEMENTATION

In this section we give detailed explanation of the VHDL implementation of PKA. We begin with the preliminaries about the various stages of PKA, and then move on to a detailed description of the implementation of each of the stages, and design choices we had to make in each case.

3.1 Preliminaries

The PKA implementation consisted of six main stages: 1) FFT Computation, 2) Peak Detection, 3) Quantization, 4) Polynomial Evaluation, 5) Chaff Point Generation and Mixing, and 6) Lagrangian Interpolation. Each of these stages requires extensive floating point operations which VHDL does not provide. Ideally we would want to build an ALU in VHDL for addressing this. However, that would require around 40000 gates [4]. To satisfy our *minimal resource utilization* design goal we did not go for an ALU design in VHDL. Further floating point numbers were not represented according to the IEEE 754 standard as it is complex and requires almost 47% of the logic cells present in an FPGA [13]. Instead a fixed-point representation was used. As we shall see, this approximation did not adversely affect the accuracy of PKA.

[1]Such FPGA boards can also be directly interfaced with the sensor nodes in the BAN. Recent years have seen the emergence of several systems FPGA based hardware add-ons for Crossbow motes [2] [7] [17]. In such a system the motes are interfaced with boards containing an FPGA. The FPGA has its own crystal clock, I/O ports and RAM and communicates with the mote using UART connection.

3.2 Numeral Representation

In our prototype the floating point numbers are represented using fixed-point scheme that makes the implementation of arithmetic operators easier. The conversion to the fixed-point representation is a three step process - 1) The floating point numbers are represented in binary-point format; 2) The binary-point representation is shifted seven places to the right and all the bits to right of the binary-point are ignored; 3) The bits to the left of the binary point are then used to represent the floating point number in the 32 bit *fixed-point* representation. The principal idea behind the process is to multiply the floating point number by 128 and to store the integer portion of the result. This effectively provides a representation of the floating point number with a precision of two decimal places. Given, the fixed-point representation of floating point numbers, we then developed a framework for basic arithmetic operations on floating point numbers such as 32 bit floating point addition, subtraction, multiplication, division, comparison and exponentiation [15]. Apart from these, we also had to perform complex arithmetic operations (for FFT computations). We addressed this issue by representing complex numbers using two registers for the real and imaginary parts. Complex arithmetic was performed by separately manipulating the real and imaginary part of complex numbers.

3.3 Stage Details

In this section we describe in detail how each of the principal stages of PKA were implemented using VHDL.

FFT Computation: The Cooley Tukey (CT) algorithm [6] was implemented in VHDL to compute FFT. FFT computation is extremely memory intensive. In order to meet the *minimal resource usage* design goal we had to employ strategies to reduce the memory footprint of the FFT computation (measured by the number of register bits required for the implementation). The solution to this problem was an implementation of FFT with only a single *butterfly* structure[2]. No attempt was made to parallelize the butterfly operation, on streams of data, as that would have increased the memory footprint of the system. For our specific implementation we required a 256 point FFT. According to the CT algorithm the computation of FFT is divided into several stages (for

[2]Butterfly operation is the basic computation in FFT. The butterfly structure has two complex floating point inputs and two complex floating point outputs. The computation is done in two stages; first, a complex addition and a complex subtraction are done on the inputs, then each of the outputs is multiplied by appropriate twiddle factors.

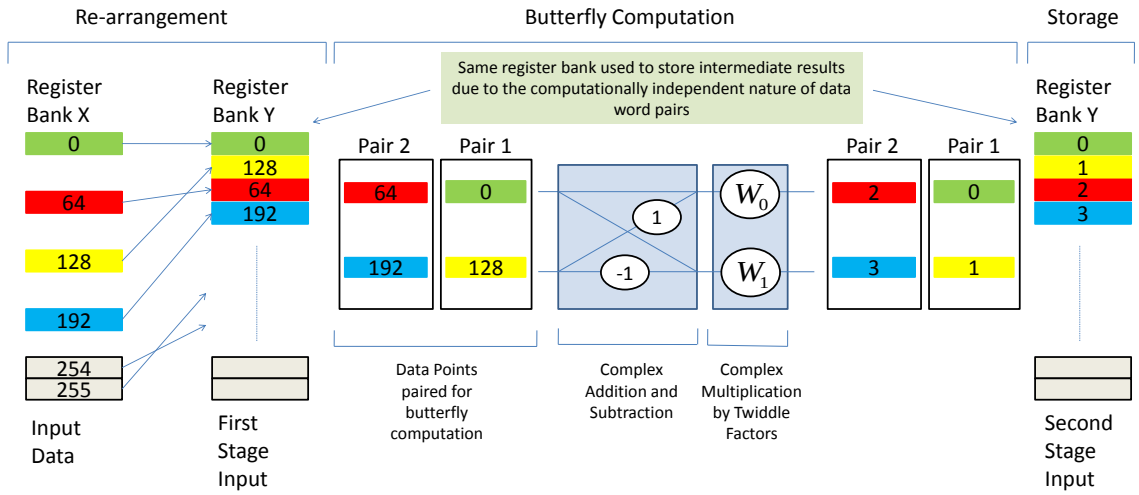

Figure 3: Implementation logic for FFT computation

256 point FFT computation there are 8 such stages) where in each stage a smaller point FFT is calculated. Figure 3 shows the implementation logic of a particular stage of the FFT computation. We first organized the data in the bit-reversed [12] order and stored it in a register bank X that contained 256 registers each 32 bits long (data word). Each stage of FFT had three phases: 1) *Re-arrangement*: In this phase the data from the previous stage of implementation was re-organized and stored in a register Y (size of Y = size of X) such that, each pair of data elements on which the butterfly operation would be performed were in successive words of the register. This re-arrangement ensured that after the butterfly computation the output data elements can be stored in the same location as the input data elements; 2) *Butterfly*: The butterfly operation was performed on pairs of data elements that were fed serially into the butterfly. Butterfly operation consisted of two complex additions and two complex multiplications by the twiddle factor [3]; and 3) *Storage*: The results of the butterfly computation on the data words were then stored in their respective locations in the register Y (feasibility of such storage is ensured by the re-arrangement phase). This approach to FFT computation helps in reducing its memory footprint as we use only a single butterfly and only two register banks (each containing 256 data words each 32 bits long) for the entire operation. This FFT implementation has a greater latency than any parallelized computation of FFT but it helps in keeping the memory footprint low.

Peak Detection: Once the FFT was implemented, peak based features were extracted from it. Peak detection is an essential step in this regard and was done by implementing a *slope detector* (that detects a positive or negative slope in the FFT coefficient curve) and a *threshold detector* (that compares the difference in two FFT values with a threshold). Figure 4 illustrates the peak detection process. The FFT coefficients were fed one by one to the register RegA (a parallel in parallel out 32-bit shift register) which was triggered by a clock. The output of RegA was connected to the input of RegB and thus RegB contained the FFT coefficient that was present in RegA in the previous clock cycle. The contents of RegA was compared with RegB using a 32 bit comparator that outputs *true* if value in RegA was greater than that in RegB and *false* otherwise. A *true* output of the comparator, suggested an upward slope in the FFT coefficient curve, which eliminated the possibility of a peak. We therefore moved on and compared the next two FFT coefficients. However, if RegB was greater than RegA it suggested an abrupt change in slope from positive to negative. This indicated that the value in RegB could be a possible peak. The clock to RegB was therefore gated in order to store the possible peak. The value of RegB was then compared with successive FFT coefficients using the *threshold detector* to see whether there was a significant decrease in the values of the FFT coefficients. If the difference in the value of the FFT coefficient was greater than a previously set *threshold* then content in RegB was considered as a peak and was stored in the register bank.

Quantization: Once the peak index and their magnitudes (peak values) were detected, they were quantized into one of many levels. These levels were determined based on the standard deviation and mean of the peak values. This step involved exponentiation, which was achieved by doing repeated addition and shifting using shift registers. Care had to be taken to avoid overflow at the registers, which is further discussed in Subsection 3.4.

Polynomial Evaluation: Once the features were quantized, a *vth* order polynomial was selected, whose coefficients were generated randomly, using the Mid-Square method [11]. The polynomial evaluation based on the features involved basically multiplying the feature values to the polynomial coefficients and adding them.

Chaff Point Generation and Mixing: The feature points and their polynomial projections were obfuscated using chaff points which were again generated using a random number

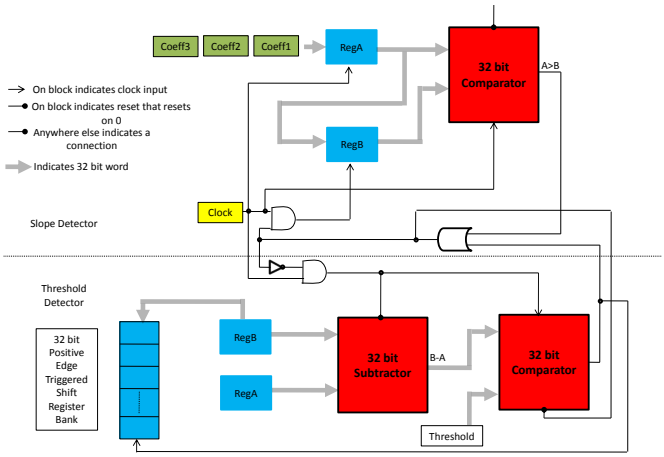

Figure 4: Implementation logic for Peak Detection

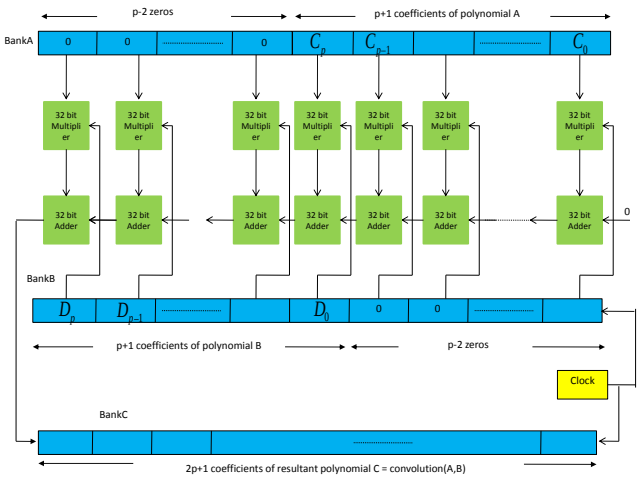

Figure 5: Implementation logic for Polynomial Convolution

generator. Extra register banks were required for the storage of the chaff points, where in order to store 1000 chaff points 2000 registers were required (each 32 bit in size) amounting to about 8 KB of memory.

Lagrangian Interpolation: The Lagrangian interpolation algorithm for decoding polynomials from their projections involved the computation of *convolution*[3] of two polynomials. The convolution of two polynomials each of order p was computed by first storing them into register banks with $2p-1$ number of 32 bit shift registers. Figure 5 shows the implementation of polynomial convolution. For one of the polynomials, the coefficients were stored in the register bank BankA and the bank was padded with $p-2$ zeros to the left. The other polynomial was stored in the register bank BankB and was padded with $p-2$ zeros to the right. Then the register bank BankB was slid over BankA by shifting BankB to the right. For each slide $2p-1$ multiply and accumulate operations were performed. The result of the multiply and accumulate operation was stored in another register bank BankC with $2p+1$ entries. This register bank stored the coefficients for the resulting polynomial.

3.4 Overflow Issues

An important issue to be considered during the implementation of the aforementioned stages is overflow. We suggest the following two simple strategies to overcome this issue:

- **Division before Addition**: Where ever possible perform division operation before the addition operation. For example, during the quantization phase, calculation of mean and standard deviation of FFT coefficients were required. These involved addition operation followed by a division operation. The accumulate operation often lead to overflow when executed before the division operation.

[3]Convolution means multiplication of two polynomials to generate a third polynomial. If the two polynomials that are being multiplied have orders a and b respectively then the resultant polynomial has order $a+b$.

- **Scaling Down Large Numbers**: It is often useful to scale down large numbers by a factor before executing multiply operations in order to avoid overflow. For example, during the evaluation of the polynomial, at each feature point, the feature values were scaled down by a factor of 1000 so that their post-evaluation projections do not overflow.

4. EVALUATION

A number of approximations had to be made for implementing PKA protocol on VHDL. These implementation decisions need to be validated in order to ensure the correct operation of the PKA protocol. We evaluate our implementation by showing its compliance with the design goals. We also discuss the several trade-offs that were required to be considered during the implementation.

4.1 Compliance with Design Goals

In this section we demonstrate that our implementation of PKA (a CPSS) in VHDL meets the three design goals we set forth earlier.

Accuracy: The primary approximation we made was the fixed-point representation of floating point numbers. This resulted in the overall loss of precision but did not adversely affect our results, especially at the pivotal stage of FFT computation. We verified this by comparing the FFT computed on VHDL with the FFT computed by our benchmark Matlab version. Figure 6 shows the comparative results. The root mean square error calculated as a percentage of the mean of the FFT coefficients is around 0.94% for VHDL implementation. The accuracy of implementation of the peak detection module was tested by comparing the number of peaks obtained in the VHDL implementation with the number of peaks obtained in the Matlab benchmark. The results of the comparison are shown in the Table 1. Figure 7 compares the peak-values and peak indexes of the FFT coefficients obtained from Matlab and VHDL. The results show that the peak indexes do not differ but the peak values do sometimes, primarily due to the quantization process. How-

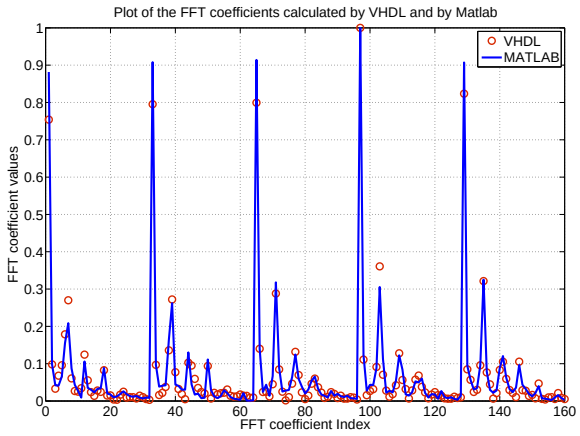

Figure 6: Comparison of FFT coefficients computed by VHDL and Matlab

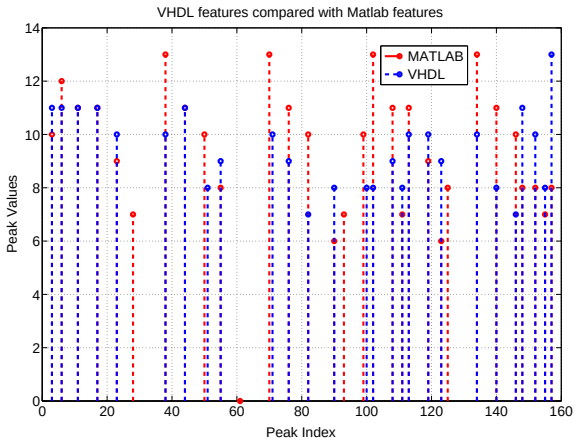

Figure 7: Comparison of Peaks in FFT Coefficients between VHDL implementation and Matlab of PKA

ever, this loss of accuracy with respect to the benchmark implementation is not detrimental to the vault construction in PKA, as there is a substantial difference between common features from same and different subjects. By choosing a polynomial order in between these two values (our choice was an 8th order polynomial), as specified in [24], we can ensure the accurate operation of the protocol thus meeting the design goal.

Low Latency: The latency of our implementation is evaluated on the basis of time taken for the execution of the protocol. We measured the number of clock cycles taken by our implementation of PKA and then multiplied it by the clock speed of our reference FPGA. Table 2 summarizes the results. The total time taken for the execution of PKA at the sender side is 32.2 msec and that on the receiver side is 59 msec after the measurement phase of the physiological signal. These results were obtained by considering the clock speed of FPGA to be 20 MHz [1]. The receiver takes more time to execute the protocol as it involves the computation of Lagrangian interpolation in addition to FFT computation and peak detection, resulting in a significant increase in the number of clock cycles.

Minimal Resource Usage: The primary requirement of this design goal is to keep the hardware requirement of the implementation below the available amount of resource in the FPGA system. In order to evaluate this design goal we propose a new metric - *memory footprint*. We define *memory footprint* of a VHDL implementation as the number of bits that are being used by all the variables that are declared in the implementation. Table 2 shows the memory footprint of the sender and the receiver. From the results we can see that our implementation has a memory footprint much less than the total amount of RAM (56K*4Kb=28MB) present in our FPGA custom board thus meeting the minimum resourced usage design goal.

4.2 Trade-Offs

The adherence to the design goals presents a number of trade-offs during implementation. Below we list the trade-offs with respect to our implementation of PKA. We believe that these will affect any CPSS implementation and should be carefully considered.

Accuracy vs. Complexity: There is an inherent tradeoff between the accuracy and the resource usage design goals. Intuitively higher accuracy in implementation requires higher amount of resource. In the specific example of PKA we did not implement an IEEE 754 compliant representation of floating point numbers due to the complexity of the approach and its potential high memory usage. Instead a much simpler fixed-point representation of floating point numbers was used that gave us the required accuracy while not stressing the memory usage.

Latency vs. Resource Usage: Trade-off exists between the latency and resource usage design goals where in order to keep resource usage within limits, performance degradation is incurred. In PKA implementation we incurred this trade-off during the FFT computation. We did not implement a parallelized version of FFT as that would incur more memory usage. However, the implementation approach that we took increased the latency of the FFT computation but it enabled us to perform FFT using minimal resource (low memory footprint).

5. SOFTWARE IMPLEMENTATION

In this section we discuss the viability of a software implementation of PKA (on sensor nodes in a BAN) based on important insights drawn from the VHDL implementation. Software implementation of the protocol has many advantages compared to a hardware add-on:

- The hardware is customized to perform only a single task and cannot be adapted easily to meet the changing requirements of the mote. Software version of the protocol can be updated on the fly or its parameters tweaked easily.
- The hardware has its own powering needs which may potentially strain the mote batteries. Software implementations are usually not so power intensive.

Table 1: Comparison of number of peaks in Matlab and VHDL

Parameters	Matlab	VHDL
Average number of peaks	30	26.5
Number of common peaks for sensor nodes in same BAN	12	10
Number of common peaks for sensor nodes in different BAN	2	1.7

- Each mote needs to be interfaced with the board, which can be expensive and time consuming process. No additional interfacing is required if the mote itself can implement PKA.

In this section we provide a high level analysis of the issues in implementing CPSS in mote software with specific reference to PKA. The sensor nodes in the BAN are implemented using Crossbow motes that have a 8MHZ processor, 4KB RAM, utilize a ZigBee radio to communicate and are powered with 2AA batteries. The implementation platform of Crossbow motes (used in the Ayushman system's BAN) is quite different from an FPGA. In an FPGA there is no Arithmetic Logic Unit (ALU) however in motes there is a 32 bit fixed-point ALU implemented. Hence, many of VHDL components implementing basic arithmetic operations are not required with motes. Again we do not have to concentrate on gate level or register level specifications of components, algorithmic specification of components in TinyOS and NesC (mote programming language) will suffice. Despite these differences there are certain inherent similarities in the operation of the two platforms. Both VHDL and TinyOS do not support floating operations and have a fixed-point representation of numbers, and both the FPGA and the mote hardware are at an inherent disadvantage with respect to supporting signal processing applications. The prototype development of the protocol in VHDL helped us to identify several important challenges and optimization options that were applicable to software based implementation. We summarize the important ones below:

- The implementation of a floating point arithmetic unit in VHDL can also be applied for motes.
- The FFT implementation technique in VHDL is applicable for implementation in motes as it helps to keep memory usage within limits.
- The procedure followed in the implementation of the quantization, polynomial evaluation and Lagrangian interpolation methods can also be employed here, using the 32 bit fixed-point representation of the floating point numbers.
- Overflow issue while executing floating point operations should be taken into account. The methods employed to avoid them in VHDL can be easily extended for the mote environment.
- Efficient management of chaff points is required to implement the protocol in motes in order minimize the memory required.

A very important constraint in the software based approach to the implementation of CPSS is the limitation in the available amount of resources in the mote hardware especially memory. This may require additional design choices so as to meet the *Minimum Resource Usage* design goal. For example, it is already mentioned in Section 4 that the generation and storage of chaff points takes about 8 KB of RAM memory for 1000 chaff points. Many of the present day motes only have 8 KB RAM memory which means that blindly generating chaff points and storing them would leave no space for other operations. Thus an efficient handling of memory is needed while implementing PKA directly in the mote.

6. RELATED WORK

The study of the challenges and trade-offs of implementing a CPSS in a BAN is novel and has not been conducted so far. The authors in [22] provide a survey of current directions in the area of security solutions for pervasive healthcare. The idea of a cyber-physical approach to security was first proposed in [5] and [23] - where the authors proposed the use of physiological signals for hiding secret between two sensor nodes. However, no working system was developed by them. [20] and [24] proposed security infrastructure that took an environmentally coupled approach towards security. The authors used EKG (Electrocardiogram) and PPG (Photoplethysmogram) signals to provide authentication to sensor nodes within the same BAN. However, the authors did not do any implementation in real world systems and hence could not provide the design guidelines or challenges for actuating such a system in a BAN. In [9] the authors describe an implementation of a mote based underwater sensor network where it implements signal processing applications in the mote such as FFT. This solution is considerably more complex due to the computation of 1024 point FFTs and higher. Further, they have higher latency as well due to the memory transfers between ROM and RAM. The authors in [18] and [19] provide discussions on the thermal safety issues in BAN taking a cyber-physical approach to the communication scheduling and thermal-aware routing problems. However they do not provide insight on the challenges in securing the inter-sensor communication in a BAN.

7. CONCLUSIONS

In this paper we presented the challenges involved in the implementation of a new class of security protocols which are cyber-physical in nature - Cyber Physical Security Solutions (CPSS) - which combine signal processing with security primitives. We identified design guidelines for the implementation of a CPSS from our experiences while implementing a specific solution Plethysmogram based Key Agreement (PKA) on FPGA (using VHDL). There were inherent trade-offs within the design goals that required intelligent decisions at several stages of the implementation. Based on our experience with the VHDL implementation of PKA we provided insights into the possible challenges and trade-offs in a software based implementation of any CPSS in a BAN. We envision that the challenges and trade-offs

Table 2: Clock Cycles and Memory Footprint of implementation of PKA

Module	Clock Cycles	Memory Footprint(KB)
Transmitter	6,433.15	47.35
Receiver	11,779.80	45.3

are applicable in general to the realization of any CPSS in resource constrained environments.

Acknowledgments

A. Banerjee is supported by the Science Foundation of Arizona (SFAz) fellowship. This research was funded in part by National Science Foundation grants CNS-0617671 & CNS-0831544. The authors would like to thank Tridib Mukherjee of IMPACT Lab, for providing useful insights.

8. REFERENCES

[1] In *Spartan-II 2.5V FPGA Family:Complete Data Sheet*, September 2003.

[2] M. Azimi-Sadjadi, G. Kiss, B. Fehér, S. Srinivasan, and Á. Lédeczi. Acoustic source localization with high performance sensor nodes. In *Proceedings of SPIE*, volume 6562, page 65620Y, 2007.

[3] G. Bi and Y. Zeng. *Transforms and Fast Algorithms for Signal Analysis and Representations*. Springer, 2004.

[4] M. Buhler and U. G. Baitinger. Vhdl-based development of a 32-bit pipelined risc processor for educational purposes. In *Electrotechnical Conference, 1998. MELECON 98., 9th Mediterranean*, volume 1, pages 138–142 vol.1, 1998.

[5] S. Cherukuri, K. Venkatasubramanian, and S. K. S. Gupta. BioSec: A Biometric Based Approach for Securing Communication in Wireless Networks of Biosensors Implanted in the Human Body. pages 432–439, October 2003. In Proc. of Wireless Security and Privacy Workshop 2003.

[6] J. W. Cooley and J. W. Tukey. *Mathematics of Computation*, 19(90):297–301, 1965.

[7] H.-J. Hof. *Algorithms for Sensor and Ad Hoc Networks*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007.

[8] A. Juels and M. Wattenberg. A Fuzzy Commitment Scheme. pages 28–36, Nov 1999. In Proc. of ACM 9th Conf. on Comp. & Comm. Sec.

[9] R. Jurdak, A. G. Ruzzelli, G. M. OŠHare, and C. V. Lopes. Mote-based underwater sensor networks: opportunities,challenges, and guidelines. February 2008.

[10] I. Korhonen, J. Parkka, and M. Van Gils. Health monitoring in the home of the future. *Engineering in Medicine and Biology Magazine, IEEE*, 22(3):66–73, May-June 2003.

[11] P. A. W. Lewis and E. J. Orav. *Simulation Methodology for Statisticians, Operations Analysts, and Engineers*. CRC Press, 1989.

[12] C. V. Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics, 1992.

[13] Louca, L. and Cook, T.A. and Johnson, W.H. Implementation of IEEE single precision floating point addition and multiplication on FPGAs. *FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on*, pages 107–116, Apr 1996.

[14] R. Paradiso, G. Loriga, and N. Taccini. A wearable health care system based on knitted integrated sensors. *Information Technology in Biomedicine, IEEE Transactions on*, 9(3):337–344, Sept. 2005.

[15] S. Salivahanan and S. Arivazhagan. *Digital Circuits and Design*. Vikas Publishing House PVT LTD, 2004.

[16] L. Schwiebert, S. Gupta, and J. Weinmann. Research challenges in wireless networks of biomedical sensors. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 151–165. ACM New York, NY, USA, 2001.

[17] T. Ahola and P. Korpinen and J. Rakkola and T. Ramo and J. Salminen and J. Savolainen. Wearable FPGA Based Wireless Sensor Platform. *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 2288–2291, Aug. 2007.

[18] Q. Tang, N. Tummala, S. Gupta, and L. Schwiebert. Communication scheduling to minimize thermal effects of implanted biosensor networks in homogeneous tissue. *IEEE Transactions on Biomedical Engineering*, 52(7):1285–1294, 2005.

[19] Q. Tang, N. Tummala, S. Gupta, and L. Schwiebert. TARA: thermal-aware routing algorithm for implanted sensor networks. *Lecture Notes in Computer Science*, 3560:206–217, 2005.

[20] K. Venkatasubramanian, A. Banerjee, and S. Gupta. Ekg-based key agreement in body sensor networks. *Computer Communications Workshops, 2008. INFOCOM. IEEE*, pages 1–6, April 2008.

[21] K. Venkatasubramanian, G. Deng, T. Mukherjee, J. Quintero, V. Annamalai, and S. K. S. Gupta. Ayushman: A Wireless Sensor Network Based Health Monitoring Infrastructure and Testbed. In *Distributed Computing in Sensor Systems*, pages 406–407, July 2005.

[22] K. Venkatasubramanian and S. Gupta. Security solutions for pervasive healthcare. *Security in Distributed, Grid, Mobile, and Pervasive Computing*, page 349, 2007.

[23] K. Venkatasubramanian and S. K. S. Gupta. Security for Pervasive Health Monitoring Sensor Applications. pages 197–202, December 2006. In Proc. of the 4th International Conference on Intelligent Sensing and Information Processing.

[24] K. K. Venkatasubramanian, A. Banerjee, and S. K. S. Gupta. Plethysmogram-based secure inter-sensor communication in body area networks. *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7, Nov. 2008.