# Delay Aware, Reconfigurable Security for Embedded Systems

Tammara Massey, Philip Brisk, Foad Dabiri, William Bishop, Ricardo Oliveira, Majid Sarrafzadeh

## ABSTRACT

Wireless embedded systems, especially life-critical body-area networks, need security in order to prevent unauthorized and malicious users from injecting traffic and accessing confidential data. Coupled with the security costs in system performance and power consumption, embedded systems are also restricted by the type of security that can fit in their limited memory. The *Dynamic Security System* (DYNASEC) architecture is a reconfigurable security system that allows a central node to program other nodes with different levels of security. A delay-aware heuristic based on throughput and encryption decides when the level of security should be changed under various timing constraints. The goal is to maximize the strength of the security while meeting the deadline. This experimental analysis of a reconfigurable electrocardiogram (ECG) application validates the efficacy of the DYNASEC architecture in a body area network. Our experiments demonstrate that DYNASEC enables lightweight medical embedded systems to dynamically optimize security levels to meet timing constraints in a body sensor network.

## Categories and Subject Descriptors

B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

## General Terms

Performance, Design, Experimentation, Security.

## Keywords

Performance evaluation, security, embedded systems, body area networks, quality of service, medical applications, adaptable systems.

## 1. INTRODUCTION

Security is becoming increasingly important in wireless embedded systems, especially when the data being transmitted is life-critical and/or confidential via legal mandate. Nowhere is this more important than body-area networks where motes embedded on the patient's body could be monitoring a potentially fatal health condition and/or providing the controls for a drug delivery system that could save the patient's life in the result of an anomaly [12]. Security is challenging, however, because embedded systems typically have limited battery supplies and processing power coupled with the fact that real-time applications have stringent timing constraints that cannot be violated. Within the security community, the trend has been toward stronger cryptographic algorithms with increased processing requirements; meanwhile, increased wireless communication rates have placed further strain on battery lifetimes. The result is called a *security-processing gap*, a term which recognizes the collective disparity between security requirements and the processing capabilities of embedded processors [23]. Wireless communication in networked embedded systems, furthermore, is limited by bandwidth and power. Such systems must minimize the number of packets sent in order to conserve precious power resources.

Wireless networks are more vulnerable to security attacks than wired networks, because the malicious intruder simply needs to activate an antenna rather than physically compromising a wire. An attacker can easily eavesdrop or inject data into an unprotected network simply by receiving or transmitting. Consequently, data integrity and confidentiality must be ensured while maintaining network availability; however, approaches to security must deviate from those employed for wired networks due to limited computational and communication capabilities of the nodes in the network.

*Sensor Operating System* (SOS) is an operating system that provides code migration between different embedded systems [11]. Code migration is a service that enables one embedded system to download a program or module to be executed on another. Code migration is essential for creating networked embedded systems that can be reconfigured after deployment.

Code migration, unfortunately, makes SOS particularly vulnerable to attacks on authentication, whereby an attacker injects a malicious program onto a node which will then execute it. This type of attack can be countered by using symmetric key cryptography that authenticates each message with a MIC (Message Integrity Code), a secure checksum of the message. Snooping is prevented by encrypting the messages with either the Skipjack or RC5 ciphers. In the abstract, this solution is not particularly novel; however, an effective implementation that meets timing constraints, despite the security processing gap, and minimizes power consumption while providing at least the minimum amount of required security is difficult to achieve.

As a motivating example, consider a wireless network in the healthcare industry. The Health Insurance Portability and Accountability Act (HIPAA) mandates that any healthcare professional who "maintains or transmits health information shall maintain reasonable and appropriate administrative, technical, and physical safeguards to ensure the integrity and confidentiality of the information; to protect against any reasonably anticipated threats or hazards to the security or integrity of the information; and unauthorized uses or disclosures of the information." If the sensors attached to the device detect abnormal health-related activity, then additional processing power is required and extra packets must be transmitted to take care of the situation within hard real-time constraints [12].

Clearly, the medical condition of the patient experiencing an abnormal health problem must take precedence over the confidentiality of the data. Here, we assume that the cost of security under normal operating conditions is too high to allow the deadline to be met. Consequently, a reconfigurable type of security that lowers the strength of the security while enabling the system to reach its deadline is necessary; however, the security

should be lowered no further than the absolute minimum that will permit the system to meet its deadline.

To meet these needs, we have designed and implemented *Dynamic Security System* (DYNASEC), a reconfigurable security architecture for *Sensor Operating Systems (SOS)*. The primary contributions of this paper include:

(1) The DYNASEC architecture, which ensures message integrity and confidentiality when programs are uploaded wirelessly on embedded systems.
(2) A lightweight delay-aware system that selects the maximum level of security while meeting timing constraints.
(3) An experimental analysis in a simulation that establishes DYNASEC's ability to maximize security while meeting timing constraints.

The overall goal of DYNASEC is to dynamically maximize security settings in order to meet timing constraints. This sets DYNASEC apart from other established security protocols for embedded networks that have not addressed timing constraints in-depth [2][16][10][19][13]. DYNASEC's reconfigurable security architecture has two modes: (1) integrity and (2) integrity+encryption. Integrity verifies that messages were not modified in transit; encryption ensures that only authorized nodes can read the information, which is broadcast across an otherwise insecure wireless channel. DYNASEC also implements two lightweight cryptographic protocols: Skipjack and RC5.

The remainder of the paper is organized as follows. Section 2 discusses related research in security in embedded systems. Section 3 follows by describing the design and implementation of DYNASEC's reconfigurable security architecture. Section 4 describes the light-weight medical application that ran on top of the reconfigurable security architecture. Section 5 describes the experimentation of delay in a simulation and explains the algorithm for the dynamic allocation of security. Finally, Section 6 culminates with conclusions and future work.

## 2. RELATED WORK

DYNASEC has been designed and implemented to meet the security and power utilization needs of the next generation of networked embedded systems. One of the most important design challenges for such systems is flexibility [22]. DYNASEC allows an embedded system sufficient flexibility to adapt to changing system requirements and to reprogram the embedded device via wireless code migration.

Prior research has focused on implementing optimal security protocols on embedded systems or analyzing power consumption on embedded systems [8][29][16][10][19][13][21]. Previous work has also mentioned flexibility, reconfigurability, and adaptive execution of security protocols as future work [25][23][21] but has not tackled the problem directly.

TinyPK [29] is a security scheme that provides authentication and key exchange between an external party and a sensor network. It is based on the well-known RSA cryptosystem using e=3 as the public exponent. RSA is public key cryptography whose public operations are very fast compared to other public key technology computations. The security properties of the low exponent variant of RSA have been extensively studied [2]. The purpose of these variants is to reduce the runtime overhead of the algorithm for both wireless embedded devices and high-performance servers that are heavily loaded; clearly, the former pertains to this paper. Moreover, backwards compatibility is preserved.

To make TinyPK practical for low power sensor devices, the authors [29] designed the system to implement only public key operations, data encryption and signature verification in the sensor network.

DYNASEC, in contrast to TinyPK, allows the cryptographic algorithms to change depending on timing constraints. In DYNASEC, if timing constraints do not permit the encryption to be used, it can be swapped in favor of a lighter-weight protocol. Likewise, if a lighter-weight protocol (in terms of power consumption) is available and can meet the security constraints, then DYNASEC replaces the power expensive encryption to further reduce power consumption.

Malan et al. [16] describe the first known implementation of elliptic curve cryptography (ECC) for embedded systems on the mica2 mote. ECC has smaller keys than other types of cryptography with the same level of security. Prior to this work, it was assumed—although never experimentally verified—that public key cryptography was too computationally expensive for wireless embedded systems. Fortunately, the authors were able to establish that public key cryptography is actually tractable on constrained embedded systems. Gura et al. [10] describe an implementation of RSA and ECC on mica2 motes using optimized assembly code. Their algorithm reduces the number of memory accesses in the mote. DYNASEC, once again, is not limited to any single cryptographic algorithm.

The Timed Efficient Stream Loss-tolerant Authentication (TESLA) protocol [19] uses symmetric cryptographic functions to achieve asymmetric properties. Asymmetric properties prevent an unauthorized receiver who has somehow computed the secret key to a message from injecting malicious data into the network, effectively impersonating the sender. TESLA requires that all nodes in the network be loosely synchronized with the source, as well as agree on the timing structure and key disclosure delay. DYNASEC on the other hand does not need to be synchronized because all timing measurements are done within one node.

TinySec [13] is a link layer security scheme for wireless sensor networks. TinySec provides link-by-link security, access control, message integrity, and confidentiality. Packet level authentication is provided using MAC (Message Authentication Code) and confidentiality is provided through Skipjack and RC5 encryption. DYNASEC provides similar functionality as TinySec in that it encrypts data bit by bit as it is transported over the radio. However, DYNASEC also it gives flexibility to the system because it allows it to switch between the different levels of security.

## 3. DYNASEC SYSTEM DESIGN

DYNASEC offers link-by-link integrity and encryption that has been implemented in an SOS. To ensure integrity, DYNASEC computes a Message Integrity Code (MIC) over the header and payload of the packet. The payload can also be encrypted after the MIC has been computed.

### 3.1 Operating System and Hardware

SOS, Sensor Operating System, [11] is an operating system designed specifically for embedded systems and uses a message passing system to sever ties between the core operating system

and individual applications (modules). Modules can be loaded or removed at run time without interrupting the core operating system. Rather than using a main function, modules implement a message handler in the form of a single switch/case block that directs messages to their module specific code.

SOS's ability to dynamically upload modules makes it especially vulnerable to attacks on integrity, which is of particular importance in sensor networks. Users should be able to trust the values detected by the sensors that is collected and forwarded to more powerful centralized nodes for processing. To ensure integrity, unauthorized parties cannot be allowed to modify messages or inject their own messages into the network. If an attacker can inject a message into the network that uploads a module or program to be executed, then the entire network will operate at the attacker's whim.

SOS can run on a Mica2 mote device [5]. The Mica2 mote has three components: a processor board, a sensor board, and a programming board. The processor board is a MPR400CA, which uses an Atmel ATmega 128L microprocessor with 128KB of internal flash memory. This microprocessor simultaneously executes sensor processing and the radio/network communication stack. The sensor board is a MTS310CA, and contains a light sensor, a temperature sensor, and acoustic sensor, a sound sensor, a 2-axis accelerometer, and a 2-axis magnetometer. The programming board is a MIB510 [5]. Additional sensor boards can also be added to the Mica2 mote, such as an ECG sensor board [7].

The implementation for DYNASEC was done in the kernel of SOS. The security was implemented in the kernel because the kernel is more secure against malicious code. The security levels however increased the size of the kernel. The DYNASEC kernel takes up 39% of the internal memory. On the other hand, the original SOS kernel takes up 17% of the internal memory. If an application program for the mote was memory intensive, only one type of encryption algorithm could be implemented in the kernel and the key size.

The reason for the increased kernel size in DYNASEC is due to the tables (e.g. s-boxes) required for each different cryptographic algorithm; the size of the code itself is not nearly so large. This will be an inevitable affect of any cryptographic implementation on a memory-constrained device, unless new cryptographic algorithms are designed specifically to minimize the size of the data segment. Although an interesting area for future research, this is beyond the scope of DYNASEC.

## 3.2 Authentication

DYNASEC uses cipher block chaining (CBC-MIC), for computing and verifying MICs. CBC-MIC is efficient and fast, relies on a block cipher, and minimizes the number of cryptographic primitives implemented in memory. CBC-MIC is provably secure, but the messages must be a standard size. Bellare, Kilian and Rogaway [1] suggest three alternatives for generating MICs for variable sized messages. The lightweight variant that XORs the message length with the first plaintext block is implemented in DYNASEC. This XOR variant is more attractive than the other two variants developed because the length of the message is not needed until the end of the computation.

A MIC size of 4 bytes was chosen for DYNASEC. With a 4 byte MIC, an attacker has a 1 in $2^{32}$ chance in blindly forging a valid MIC for a particular message. This number may not be large enough in conventional networks, but in sensor networks, this provides an adequate level of security. Attackers can try to flood the channel with forgeries, but on a 19.2kb/s channel, one can only send 40 forgery attempts per second, so sending $2^{31}$ packets at this rate would take over 20 months! The small size of the MIC is kept to 4 bytes because the majority of power consumption comes from radio due to the increased header [13].

## 3.3 Encryption

DYNASEC protects against attacks on data integrity and eavesdropping under the assumption that the mote is not physically compromised. DYNASEC does not attempt to protect against replay attacks. Secure encryption requires two design decisions: selecting an encryption scheme and specifying the initialization vector (IV) format. In DYNASEC, an 8 byte IV and CBC is used.

Skipjack is an asymmetric block cipher developed by the National Security Agency (NSA) and is implemented in DYNASEC because it performs well in embedded microcontrollers [13]. Skipjack uses a Feistal network with an s-box that uses permutations of numbers ranging from 0 to 255. Skipjack has an 80 bit key length and uses 32 rounds. Two round functions, Rule A and Rule B, are used and they alternate every 8 rounds [27]. DYNASEC uses an 80 bit key for Skipjack encryption.

The CBC algorithm uses Cipher Text Stealing (CTS) as described in Schneir's Applied Cryptography and RFC-2040 [26]. In CTS, the ciphertext can be the same size as the plaintext, even when the plaintext is not a multiple of the block size. Incremental decryption and encryption allows the incoming data stream to be processed one byte at a time. Incremental decryption and encryption allows for cipher operations to be done as soon as each block of data is received from the network. This allows decryption to be pipelined with the arrival of more data.

RC5 uses a simple block cipher, but has a variable block and key size. RC5 uses a Feistal-like network with modular additions and XORs. RC5 also has data dependent rotations or rounds that may make it susceptible to cryptanalysts [24]. The number of rounds in our implementation is twelve. Twelve rounds were chosen because this is the minimum amount of rounds for the RC5 protocol. The minimum was chosen to stay coherent with our goals to keep the power utilization of the encryption as low as possible. Since the RC5 key size is variable, DYNASEC allows the user to change the size of the keysize. The key lengths used in our experiments are 80 and 160 bits. The RC5 and Skipjack implementations are based on those described by Karlof et al. [13].

RC5 is a stronger cryptographic algorithm than Skipjack, and it also runs faster. However, RC5 requires that a pre-computed key schedule to be stored in memory taking up 104 bytes for each key, 2.6% of the total RAM in a Mica2 mote. However, RC5 is patented, making it less appealing for open-source and academic projects [13].

## 3.4 Keying

In cryptographic design, a good rule of thumb is to use different keys for different applications. A DYNASEC key refers to a pair

of keys, one for encrypting data, and one for computing MICs; both keys are 64 bytes. The simplest keying mechanism uses a single network-wide key among the authorized nodes. A network-wide key provides a baseline level of security, maximizes usability, and minimizes configuration. Any authorized node can exchange messages with any other authorized node, and all communication is encrypted. Messages from unauthorized nodes are rejected.

DYNASEC has four 64 byte keys as input to CBC-MIC and CBC-encryption. Different keys are used for different levels of security so that if one level is hacked, the attacker does not have the key to the other levels. To improve the strength of the keys, future work would include changing the keys dynamically over time.

## 3.5 Communication

The message format of SOS has nine components: destination id, source id, destination address, source address, message type, payload length, data, and status flag. In SOS, the payload is dynamically allocated via calls to malloc, enabling messages of arbitrary length, with a maximum size of 256 bytes. All components except the *flag* byte, which has two free bits, are transmitted over the radio. To differentiate among security modes, SOS has been modified to transmit the *flag* byte over the network. In addition, the *flag* byte can signal high priority messages.

The Initialization Vector (IV) consists of 8 bytes in total. The first four bytes are the destination address (2 bytes), message type (1 byte), and message length (1 bytes). The last 4 bytes are initialized with random numbers when the mote initializes, and is incremented by one after each packet is sent over the network. A 4 byte MIC (Message Integrity Code) is also included at the end of the payload.

An important design decision was to keep the original message structure allowing for backwards compatibility with pre-existing SOS applications. As a message passing system, SOS uses the same message structure for all kinds of communication: (1) between different nodes using the network and (2) between different modules within the same node. The new DYNASEC fields were incorporated in the original SOS message format; however, additional changes were required to the radio driver, since the new fields transmitted over the radio are dependent on the security mode used by the application.

The data was encrypted as it was sent over the radio bit by bit. The advantage of encrypting the data over the radio is to avoid saving two large data structures for the encrypted and decrypted version. This, however, puts a delay constraint on the data. The data cannot spend more time being encrypted/decrypted than it takes to send a byte.

The Mica2 radio driver was modified for DYNASEC to accommodate the different security modes. Power is saved by encrypting data as it goes over the radio bit-by-bit, eliminating the need to store the data twice. The Mica2 radio driver in SOS is a state machine, with different submachines for transmission and reception.

Figure 1a and Figure 1b show the diagram for the transmission and reception submachines respectively. The new states, RXSTATE_DYNASEC and TXSTATE_DATA_DYNASEC, were added to each submachine to handle authentication and encryption in the radio. In the case of the transmission submachine, the code checks if the appropriate flag bits are set for DYNASEC.

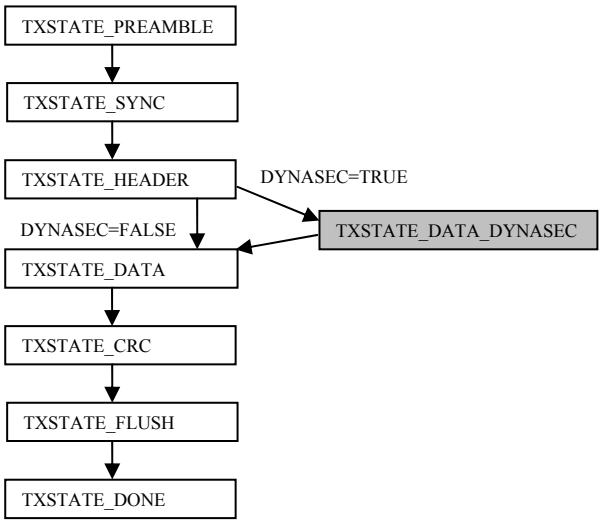


**Figure 1a:** Radio State Diagram – TX_STATE Submachine.

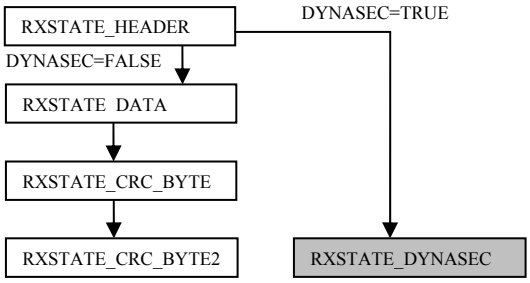


**Figure 1b:** Radio State Diagram – RX_STATE Submachine.

If only authentication is set, then the initialization vector is not sent over the radio. In this case the MIC computation routine is called and the MIC is computed over the entire packet, headers and payload. If encryption is set, then the IV and the MIC are sent over the radio, and the encryption routines are called to encrypt only the payload of the message.

When receiving a packet, the code waits for the reception of the flag byte. When this byte is received, the appropriate bits are checked and, depending on the security modes used to send the packet, the MIC/de-encryption routines may be called.

The communication function used by SOS to send packets over the network was modified to enable Skipjack cryptography. Skipjack works with blocks of 8 bytes size. Therefore, the communication function was changed so that when the length is smaller than 8 bytes, a data pointer would be allocated and padded with extra zeros to bring the data to 8 bytes.

# 4. LIGHTWEIGHT MEDICAL SYSTEM APPLICATION

DYNASEC is built specifically for resource constrained lightweight devices. Due to the portability requirements of Body Area Networks (BANs), they are comprised mainly of lightweight

medical systems with miniature embedded sensors that monitor physiological activities of the body or probe the outside environment for harmful chemicals, dangerous radiation levels, and a more general score of events. Even though ubiquitous BANs may contain various types of non-invasive and in-vivo sensors, our application focuses specifically on the electrocardiogram (ECG) sensor.

A sensor board for the ECG sensor was developed by Fulford-Jones et al. [7]. The CodeBlue project [15] includes a sensor-network-based ECG application; unlike DYNASEC, however, CodeBlue does not attempt to dynamically vary security to meet timing constraints.

## 4.1 Heart Detection Algorithm

A distinctive characteristic of the ECG signal is that transmitting the waveform of the ECG takes up a lot of bandwidth. When several ECG waveforms are transmitted over the network, the heavy network load of the waveforms reduces throughput in the network. Additionally, high noise interference from movement of the patient results in an undecipherable waveform created while signal processing on the sensor. Therefore, a lightweight ECG heartbeat extraction algorithm was developed to extract the heartbeat of the patient from the ECG waveform and to be resilient to noise. The ECG extraction algorithm is based on the SQRS algorithm [20], which can be found on MIT's physionet website [18]. The algorithm has been modified to improve performance when tested over the MIT-BIH database of ECG waveforms with different forms of simulated noise superimposed on the underlying signals. Specifically, the thresholding method was modified to make it more robust to noise experienced on the ECG leads. The original SQRS algorithm extracts features by using the derivation of the waveform to classify the features. From experimental testing on the MIT-BIH database and on ECG sensor boards attached to embedded systems, the algorithm's absolute threshold to classify features occasionally misclassified features due to noise. Our algorithm implemented a moving threshold with a lower and an upper value that were independent of one another and resulted in it being more robust to the specific types of noises experience on the embedded ECG medical systems. Due to memory constraints on the lightweight medical device, additional noise classification was removed from the algorithm in order to make it as small and lightweight as possible.

The ECG algorithm first passes the incoming signal through a simple FIR filter with coefficient weights [1.5, 0, -1.5]. Assuming a sampling frequency of 100 Hz, this gives a weighted derivative of the incoming signal with a max gain near 36 Hz achieved with the original FIR filter employed by the SQRS algorithm. This signal is then tested to see if it passes either an upper or lower threshold, both of which are dynamically updated. If the signal passes the upper and lower signals in an alternating fashion within 190 milliseconds of each occurrence, the detection of an R wave is declared. If more than four threshold crossings occur in an alternating fashion within 190 milliseconds of each other, the algorithm waits to see if more occurrences occur within 240 milliseconds of each other to prevent artifacts from being detected as beats.

The heart rate detection algorithm was tested over 48 ECG recordings in the MIT-BIH arrhythmia database of ECG recordings [18]. Combined noise including abrupt baseline shifts and drift, power line noise, EMG noise and motion artifact noise was superimposed on each recording, and the mean positive predictivity[1] and sensitivity[2] was measured and were found to be 97.6%, 91.9% respectively. The types and characteristics of super imposed noise were based on work by Freisen et al. [6] and observations of ECG waveforms recorded on portable lightweight medical systems [17].

To determine heart rate, the algorithm calculates the reciprocal of the interval between the times of detected R waves. After five seconds of no detected beats, the algorithm declares a heart rate of zero beats per minute. Upon receiving zero beat per minutes, a health care professional would check on the patient to see if the leads have fallen off or failed or if the patient has died.

## 4.2 Granularity of ECG waveform

The frequency of the ECG waveform can also be modified to vary the granularity of the received waveform. Obviously, a higher frequency will capture more data and reveal a more precise waveform; however, if several ECG devices are in the vicinity, a lower frequency could improve the overall network capacity.

Several assumptions are made in our observations. First, we assume that the Skipjack and RC5 cryptographic algorithms from [13] are secure. Secondly, we assume that several ECG devices will be in the same vicinity and may result in congestion. In an application such as triage [17], this is a valid assumption. Triage occurs during a mass casualty incident when hundreds or thousands of people are injured. If an electronic triage system was remotely monitoring the vital signs of the patients, many ECG motes would be in close vicinity to one another.

# 5. EXPERIMENTATION

DYNASEC was run in a simulation to analyze the power utilization and processing delay on the Mica2 mote. Specifically, we analyzed how the different levels of security and various packet sizes affect delay and power. In order to quantitatively analyze the delay and power, the code was run in the Avrora simulation environment [28]. Avrora simulates the embedded systems platform by running the actual microprocessor program. Avrora finds a good medium between cycle-accurate simulation, which can require excessive simulation time, and lower granularity functional simulations. Avrora recognizes that many embedded systems go to sleep on a regular basis and triggers simulation only when an event is at the head of the queue. Since Avrora is cycle-accurate, it models all low levels events in the application. Accurate Prediction of Power Consumption (AEON) is a power model built within Avrora that used actual measurements for calibration and validation [14].

DYNASEC has four levels of security: L0: plaintext; L1: MIC authentication; L2: Skipjack encryption with a 64 bit key; and L3: RC5 encryption with a 64 bit key. L0 is the least secure, while L3 is the most secure. Each level of security has a processing delay

[1] Predictivity is defined as true_positives/(true_positives + false_positives) and gives the percent of declared beats that were truly beats.

[2] Sensitivity is defined as true_positives/(true_positives + false_positives) and gives the percent of declared beats that were truly beats.

from the encryption and decryption associated with it. As the level of security increases, the processing delay also increases. Our system changes the amount of information that is transmitted and the type of encryption in order to meet timing constraints in the network. This heterogeneous network can be composed of different types of devices with different security capabilities and constraints. For simplicity, we consider that all the devices in the network are embedded ECG medical systems that can change to all four levels of security. We also assume that the forwarder does not adjust the level of security when forwarding packets in the network. DYNASEC can send data in three sizes: 1 byte for heart rate, 50 bytes for low ECG waveform sample frequency, and 100 bytes for high ECG waveform sample frequency.

## 5.1 Processing Delay Measurements

The processing delay for the four levels of security was measured at different granularities (100 bytes, 50 bytes, and 1 byte) were measured for normalized time periods. As can be seen in Figure 2, modifying the size of the packet has a large effect on delay. Also, as the security levels increases, the processing delay also increases. Switching from the heart rate extraction algorithm to a waveform greatly reduces the number of packets that can be sent out into the network by approximately 75 fold. Therefore, when the network becomes overloaded and cannot support all the data, a lower level of security will selected by the light-weight embedded systems. Also, in the case when one node has a large amount of high priority data that must reach the sink, the nodes in the networks can modify their system so that all systems meet their deadlines. By optimizing the level of security, DYNASEC allows a healthcare provider to extract enough information to effectively monitor patients and still comply by HIPAA guidelines. Maximal security at all times would be ideal, but is unfortunately unrealistic for the current generation of wireless embedded devices employed in BANs.

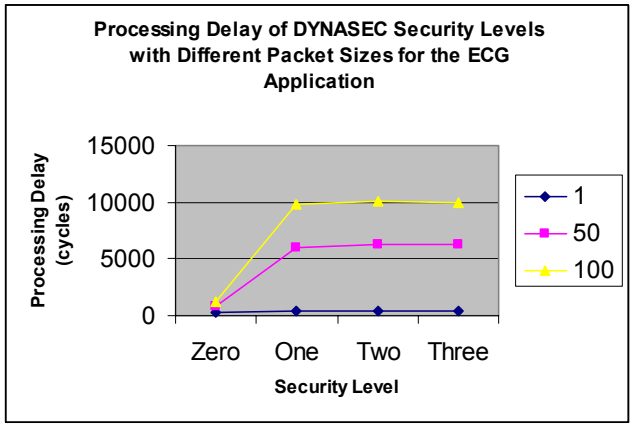


**Figure 2:** Processing Delay of DYNASEC Security Levels with Different Packet Sizes for the ECG application

## 5.2 Power Measurements

Detailed power measurements (represented in cycles) were taken for a single node for start-up and normalized periods with varying timeouts. The power was measured for the first four security levels in DYNASEC. All security levels had similar distributions of power among the CPU, leds, radio, sensor board, and flash components. The variance between the percentages for the first four security levels was less than one percent. As can be seen in Figure 3, most of the power is consumed by the radio. The second largest consumer of power in the mote is the CPU as it does processing of encryption and other functionalities.

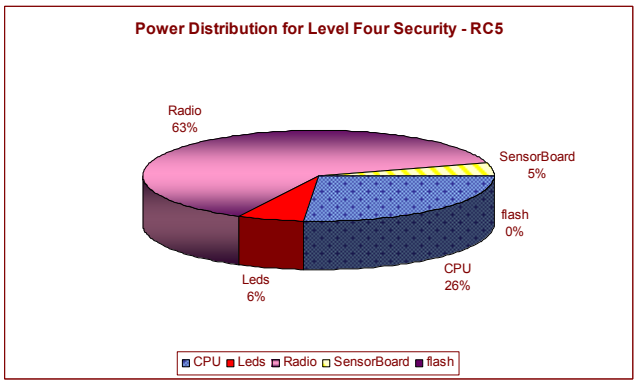


**Figure 3:** Power Distribution in Mica2 Mote for Level Four Security

Due to the fact that the radio takes up the most power and our optimizations are on the encryption and decryption are on the CPU, the power levels did not vary greatly between different security levels. Therefore, power was not used as a metric for changing the security level.

## 5.3 Dynamic Security Allocation

Dynamic security allocation is modeled as a budgeting problem on a directed acyclic graph (DAG) representing communication links in a network. It is assumed that data is collected at sensors and routed through acyclic paths toward a centralized collection of nodes for further processing. Unlike prior work on budgeting in VLSI/CAD (e.g. [9]), DYNASEC must compute a solution efficiently in real-time rather than optimally offline.

Let $V = \{v_1, \ldots, v_N\}$ be the nodes in the network. Communication links in the network are organized as a DAG $G(V, E)$, where $V$ is the set of nodes and $E$ is a set of edges. An edge $(v_j, v_k) \in E$ indicates the existence of a communication link from $v_j$ to $v_k$. The delay of $e$, denoted $D(v_j, v_k)$ is the time required to transmit a packet across the link from $v_j$ to $v_k$.

Each security algorithm $s_i$ has an associated delay $d_i$, the time required to encrypt the data. $q_i$ is the probability that $s_i$ will be broken by an adversary at a single node and $p_i = 1 - q_i$ is the probability that $s_i$ will not broken. We assume that there are $M$ different security algorithms. In the case of this paper, $M = 5$.

Let $c_{ji} = 1$ if node $v_j$ selects security algorithm $s_i$ and 0 otherwise. A legal solution to the problem defined above assigns exactly one security algorithm to each node in the network.

**Definition 1:** The security of a system, $S$, is defined to be the aggregate probability that there are no security failures:

$$S = \prod_{j=1}^{N} \sum_{i=1}^{M} c_{ji} p_i \qquad (1)$$

**Definition 2:** A path $P$ is a sequence of nodes $P = \langle v_1, \ldots, v_K \rangle$ such that there is a link from $v_j$ to $v_{j+1}$, $1 \leq j \leq K-1$. The delay of $P$, denoted $D(P)$, is computed as follows, and includes the delay of the security algorithm, which is performed at $v_1$.

$$D(P) = \sum_{i=1}^{M} c_{1i} d_i + \sum_{j=1}^{K-1} D\left(v_j, v_{j+1}\right) \quad (2)$$

In a DAG, a *source* is a node with no predecessors and a *sink* is a node with no successors. The longest possible line of communication in a DAG is from a source to a sink. Let *P** be the set of all paths originating on sources and terminating on sinks.

Lastly, *T* is defined to be a global timing constraint, which must be met in order for the embedded system to transmit the necessary packets in order to respond to an anomaly.

The problem statement and formulation are as follows:

**Problem Statement:** Given a network modeled as a DAG *G(V,E)*, assign security algorithm $s_i$ to each node such that the overall security of the system is maximized while meeting the global timing constraint.

**Problem Formulation:**

Maximize: $S$ (3)

Subject to the Following Constraints:

$$\sum_{i=1}^{M} c_{ji} = 1 \quad \forall j \quad (4)$$

$$\forall P \in P^* \quad D(P) \le T \quad (5)$$

Constraint (4) ensures that exactly one security algorithm is assigned to each node. Constraint (5) ensures that each source-to-sink path in the DAG satisfies the timing constraint. Since every possible communication-path is a sub-path of some source-to-sink path, constraint (5) satisfies every possible communication path in the DAG.

Currently, we have implemented a relatively efficient greedy algorithm that solves the problem as described above. In the algorithm, each node dynamically lowers its security level if it cannot achieve a desired level of throughput, and dynamically raises its security level if its current throughput level is significantly higher than the minimal threshold. On a node-by-node basis, this approach will ensure that constraint (5) is satisfied; however, there is no guarantee that the value of *S*, in equation (3), will be anywhere close to maximal.

Complex solvers for budgeting problems, however, are not realistic in this situation. The reason is that the motes already have limited memory and a solver that uses offline methods such as linear programming or network flow would be too large; likewise, the speed of the solution is practically more important than its quality, since the overall response of the network must be rapid in life-critical situations. As stated earlier, solving the budgeting problem effectively and efficiently is future work, and is intended solely to augment and improve our current solution.

# 6. CONCLUSION

In conclusion, an authentication and encryption scheme with four levels of security was created in DYNASEC. The application layer had two distinct security modes: (1) authentication and (2) authentication and encryption. The dynamic uploading of modules in the SOS operating system makes it especially vulnerable to attacks on authentication. An attacker can upload its own malicious programs onto other nodes in the network. We have countered against this attack by using symmetric key cryptography and authenticating each message with a MIC (Message Integrity Code), a secure checksum of the message. We have also guarded against snooping by encrypting the messages with SKIPJACK or RC5.

A reconfigurable security system was presented that switched between different security levels and packet sizes depending on timing constraints. DYNASEC's architecture gives the embedded system the flexibility to adapt to the dynamic environment where it is deployed. DYNASEC takes advantage of SOS's unique architecture and use the SPI interrupt handler to send only the necessary headers over the network. Other operating systems for embedded systems, such as TinyOS, cannot dynamically change the header length without manually reprogramming the embedded system. DYNASEC's adaptability gives systems a *reasonable* amount of security while meeting timing constraints.

The heart rate detection algorithm produces reliable results while operating under considerable environmental and human noise, such as noise created by muscle activity and respiration. Our reconfigurable application uses the heart rate detection algorithm to modify its packet size in response to network and system conditions. These reconfigurable features make it practical to deploy ECG devices in a broad range of care providers, patients, and environments.

# 7. ACKNOWLEDGMENTS

Excluded for blind review.

# 8. REFERENCES

[1] Bellare, M., Kilian, J. and Rogaway, P. The Security of the Cipher Block Chaining Message Authentication Code. *Journal of Computer and System Sciences*, 61 (3): 362-399, December 2000.

[2] Boneh, D. and Shacham, H. Fast variants of RSA. *In RSA Laboratories' Cryptobytes*, vol 5 no. 1, pages 1-8, Winter/Spring 2002.

[3] Brickell, E., Denning, D., Kent, S., Mahler, D. and Tuchman, W., *SKIPJACK Review*, Interim Report, July 28, (1993), 8 pages.

[4] A. Cerpa, J. L. Wong, L. Kuang, M. Potkonjak and Deborah Estrin, "Statistical Model of Lossy Links in Wireless Sensor Networks." *In ACM/IEEE Fourth International Conference on Information Processing in Sensor Networks (IPSN'05)*, 2005.

[5] CrossBow, Technologies. http://xbow.com.

[6] Friesen, G. M., Jannett, T. C., Jadallah, M. A., Yates, S. L., Quint, S. E., and Nagle, H. T. A Comparison of the Noise Sensitivity of Nine QRS Detection Algorithms," *IEEE Transactions on Biomedical Engineering, vol. 37*, no. 1, pp. 85-98, 1990.

[7] Fulford-Jones, T., Wei, G-Y., and Welsh, M. A Portable, Low-Power, Wireless Two-Lead EKG System, In *Proceedings of the 26th IEEE EMBS Annual International Conference*, San Francisco, CA, USA, September 2004.

[8] Ganesan, P., Venugopalan, R., Peddabachagari, P., Dean, A., Mueller, F., and Sichitiu, M. Analyzing and Modeling Encryption Overhead for Sensor Network Nodes. *In*

*Proceedings of Wireless Sensor Networks and Applications (WSNA 2004)*, San Diego, CA, 2003.

[9] Ghiasi, S., Huang, P-K., and Jafari, R. Probabilistic delay budget assignment for synthesis of soft real-time applications. *IEEE Transactions on VLSI Systems*, vol. 14, no. 8, August, 2006, pp. 843-853.

[10] Gura, N., Patel, A., Wander, A., Eberle, H., and Shantz, S. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. *In Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, Cambridge, MA, 2004.

[11] Han, C., Rengaswamy, R., Shea, R., Kohler, E., and Srivastava, M. SOS: A Dynamic Operating System for Sensor Networks. *In Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys 2005)*, Seattle, WA, 2005.

[12] Jafari, R., Dabiri, F., Brisk, P., and Sarrafzadeh, M. Adaptive and Fault Tolerant Medical Vest for Life Critical Medical Monitoring". *In Proceedings of the 20th ACM Symposium on Applied Computing (SAC 2005)*, March 2005, Santa Fe, NM.

[13] Karlof, C., Sastry, N., and Wagner, D. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. *In the Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004),* Baltimore, MD, 2004.

[14] Landsiedel, O., Wehrle, K., and Gotz, S. Accurate Prediction of Power Consumption in Sensor Networks. *In Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, Sydney, Australia, 2005.

[15] Malan, D. J., Fulford-Jones, T., Welsh, M., and Moulton, S. CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care, *in Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2004)*, Imperial College, London, U. K., April, 2004.

[16] Malan, D.J., Welsh, M., and Smith, M.D. A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography. *First IEEE International Conference on Sensor and Ad Hoc Communications and Networks*, Santa Clara, California, 2004.

[17] Massey, T., Gao, T., Welsh, M., Sharp, J., and Sarrafzadeh, M. "The Design of a Decentralized Electronic Triage System." American Medical Informatics Association (AMIA 2006). Washington, DC, Nov. 2006.

[18] MIT-BIH Arrhythmia Database Directory. *Harvard-MIT Division of Health Sciences and Technology, Biomedical Engineering Center.* 1997.

[19] Perrig, A., Canetti, R. Tygar, J.D. and Song, D. The TESLA Broadcast Authentication Protocol, *RSA Cryptobytes*, 2002.

[20] Pino, E., Ohno-Machado, L., Wiechmann, E., and Curtis, D. Real-Time ECG Algorithms for Ambulatory Patient Monitoring. *Proceedings of AMIA 2005 Annual Symposium*, Washington, D. C., USA, 2005.

[21] Potlapally, N. Ravi, S., Raghunathan, A., Jha, N. Analyzing the Energy Consumption of Security Protocols. *In the Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED 2003)*, Seoul Korea, 2003.

[22] Ravi, S., Raghunathan, A., Kocher, P., and Hattangady, S. Security in Embedded Systems: Design Challenges. *ACM Transactions on Embedded Computing Systems,* Vol 3, No 3, 2004. Pages 461-491.

[23] Ravi, S., Raghunathan, A., Potlapally, N. And Sankaradass, M. System Design Methodologies for a Wireless Security Processing Platform. *In Proceedings of Design Automation Conference (DAC 2002),* New Orleans, Louisana, 2002.

[24] Rivest, R. The RC5 Encryption Algorithm. *In the Proceedings of the 1994 Leuven Workshop on Fast Software Encryption (Springer 1995),* pages 86-96.

[25] Schaumont, P., Verbauwhede, I., Sarrafzadeh, M., and Keutzer, K. A Quick Safari Through the Reconfiguration Jungle. *In Proceedings of Design Automation Conference (DAC 2001),* Las Vegas, CA. 2001.

[26] Schneir, Bruce. Applied Cryptography. John Wiley & Sons, 1996.

[27] SKIPJACK and KEA Algorithm Specifications, Version 2.0, http://csrc.nist.gov/encryption/Skipjack-kea.htm.

[28] Titzer, B., Lee D., and Palsberg, J. Avrora: Scalable Sensor Network Simulation with Precise Timing. *In Proceedings of International Symposium on Information Processing in Sensor Networks (IPSN),* Los Angeles, California, 2005.

[29] Watro, R., Kong, D., Cuti, S., Gardiner, C., Lynn, C., and Kruus, P. TinyPK : Securing Sensor Networks with Public Key Technology, *In Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2004)*, Washington, D.C., 2004.