Noname manuscript No.
(will be inserted by the editor)

MEM-DnP - A Novel Energy Efficient Approach for Memory Integrity Detection and Protection in Embedded Systems

Satyajeet Nimgaonkar · Mahadevan Gomathisankaran · Saraju P. Mohanty

Received: date / Accepted: date

Abstract The pervasiveness of modern day embedded systems has led to the storing of huge amount of sensitive information in them. These embedded devices often have to operate under insecure environments and are hence susceptible to software and physical attacks. Thus, security has been and will remain one of the prime concerns in the embedded systems. Although a lot of hardware and software techniques have been proposed to provide high levels of security, they are hampered by the trade-offs created by the design constraints in embedded systems. This paper presents a novel energy efficient approach for *MEMory integrity Detection and Protection* (MEM-DnP). The key feature of the proposed MEM-DnP is that it can be adaptively tune a memory integrity verification module by using a sensor module. This significantly reduces the energy overheads imposed on an embedded system as compared to the conventional memory integrity verification mechanisms. The simulation results show that the average energy saved in the combined detection and protection mechanism ranges from 85.5% to 99.998%. This is substantially higher compared to the results achieved in basecase simulations with traditional memory integrity verification techniques.

S. Nimgaonkar
Trusted Secure Systems Lab (TSSL)
Department of Computer Science and Engineering
University of North Texas, Denton, TX 76203, USA
E-mail: satyajeetnimgaonkar@my.unt.edu

M. Gomathisankaran
Trusted Secure Systems Lab (TSSL)
Department of Computer Science and Engineering
University of North Texas, Denton, TX 76203, USA
E-mail: mgomathi@my.unt.edu

S. Mohanty
NanoSystem Design Lab (NSDL)
Department of Computer Science and Engineering
University of North Texas, Denton, TX 76203, USA
E-mail: saraju.mohanty@unt.edu

Keywords Embedded Systems · Memory Integrity Detection · Memory Integrity Protection · Sensor Module

1 Introduction

The past decade has seen an advent of low-power wireless networks of embedded systems that opened doors for new sets of applications to interface digital world with the physical. From personal-digital assistants (PDAs) and smart-mobile phones to network routers, networked sensors and smart cards, home appliances to critical defence systems, the embedded systems have become ubiquitous in this era of computing. Embedded systems, in contrast to general purpose computing systems, are dedicated application-specific systems designed to serve a specific task within a larger system or as an independent system, depending on target application. They enjoy an unprecedented popularity due to their cost effectiveness, high performance, portability and ease of accessibility. As the dependence on these devices increase so also does the wealth of digital information stored in them. This information may also include confidential and sensitive personal data like secret passwords, credit card numbers and bank account numbers etc. Thus security becomes a primary concern in embedded systems.

To aggravate this security issue, most of these devices have to operate in a physically insecure environment. This allows the adversary to have complete control of the computing node — supervisory privileges along with complete physical and architectural object observational capabilities. In addition, they are also vulnerable to network based and software attacks. A recent Mobile and Smart Device Security Survey [3] conducted by Mocana Corporation in Spring 2011, revealed that 65% corporate personel require a regular attention from their I.T Staff for non-PC based device attacks. The Cyber Security Watch Survey [2] conducted by CSO, the U.S. Secret Service, the Software Engineering Institute CERT Program at Carnegie Mellon University and Deloitte in January 2011, disclosed that more than 58% of attacks are caused by outsiders i.e by unauthorized access to network systems and data, thus contributing to a staggering annual monetary losses of $123,000 per organization.

In an effort to mitigate these security threats, researchers have proposed secure processor architectures- ABYSS [33], AEGIS [31], Arc3D [17,18], TIVA [16], Hide [38] and XOM [21] etc. These architectures deploy hardware cryptographic security mechanisms of Memory Encryption [6,13–15] to provide confidentiality and Memory Integrity Verification (MIV) to preserve the integrity of the application data. Memory encryption encrypts the data before writing to the memory and decrypts it after reading. This prevents an adversary from observing the confidential application data. MIV protects the data from being modified by an adversary, thus preserving the state of the application.

Though, these secure processor architectures are effective against software, network based and physical attacks, the security mechanisms implemented in them are computationally intensive and account for excessive energy consump-

tion. An embedded system is highly resource constrained. Most of them are battery powered and it is essential to have minimal energy consumption to achieve high speed and performance. Thus security measures implemented in embedded systems suffer from trade-offs imposed by energy consumption.

With this motivation, *MEM-DnP*, a novel energy efficient memory integrity detection and protection mechanism in embedded systems, is proposed in this paper. This paper focusses on reducing the energy consumption of MIV mechanism without sacrificing its security. The rest of the paper is organized as follow. Section 2 describes the MIV property along-with the possible attacks on memory integrity. Section 3 discusses the prior research related to this paper. Section 4 presents the proposed *MEM-DnP* Mechanism followed by detailed experimental evaluation in Section 6. Finally Section 7 presents the Conclusion and direction of future research.

2 Memory Integrity Verification — A Brief Discussion

Before presenting the proposed *MEM-DnP* mechanism, it is important to discuss the MIV property, possible attacks on memory integrity and some commonly used solutions to subside these attacks. This brief discussion provides the completeness to the paper and the reader can get a feel of the research without referring to external resources. Also a significant amount of research [30, 34, 8] etc. is directed towards explaining the Memory Integrity Verification mechanism in detail.

The MIV property can be defined as follows: A processor communicates with memory $\mathcal{M}$. Memory $\mathcal{M}$ has two attributes, addresses A and contents V. It maintains associations between addresses and contents. A read of memory at address A denoted by $\mathcal{M}[R, A]$ returns the value associated with A. A write into memory address A of value V is denoted by $\mathcal{M}[W, A, V]$. A write of A with value V immediately followed by a read of address A must return value V. As memory reads and writes have a notion of time, the model needs to to associate time T with reads and writes as $\mathcal{M}[R, A, T]$ and $\mathcal{M}[W, A, V, T]$.

Figure 1 shows the possible attacks on memory integrity — *splicing* attack, *spoofing* attack, and *replay* attack. In splicing attack the adversary modifies the associations between the memory addresses and its values. For example, if value V_A is associated with address A and value $V_{A'}$ is associated with address A', a splicing attack would return the value $V_{A'}$ for a memory read corresponding to address A. In a spoofing attack the adversary modifies the value V_A to a random value V'_A. In a replay attack the adversary modifies the association between the value and the time. For example, if value V_A is associated with address A at time T and the value V^*_A is associated with address A at time T^*, and $T < T^*$, a replay attack would return the value V_A when a memory read is performed at time $t > T^*$.

In cryptography a hash function, $\mathcal{H}$, is used to protect the integrity of a message. A sender creates a message authentication code (MAC) using a secret key K together with the message m as $MAC_m = \mathcal{H}(m||K)$. It sends both the

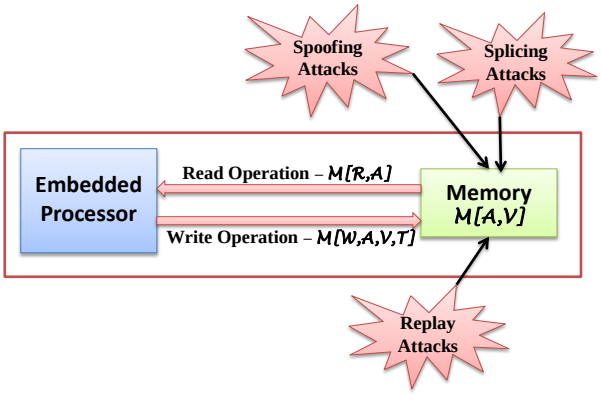


Fig. 1 Attacks on Memory Integrity in a Embedded System.

message m and the corresponding MAC_m to the receiver. The receiver, which shares the secret key K with the sender, can verify the integrity of the message by recomputing the MAC. The security properties, collision resistance, pre-image resistance and second pre-image resistance, of hash function $\mathcal{H}$ ensures that any modification to message m or MAC_m go undetected with negligible probability.

One solution for the memory integrity problem is to use message authentication codes. Processor generates a MAC $h_{V,A} = \mathcal{H}(V||A||K)$ for every memory write that stores a value V in address A. This MAC, $h_{V,A}$, is stored in the memory. On every memory read the processor computes the MAC $h'_{V,A}$ and verifies it against the MAC stored in the memory. This solution can protect memory integrity against splicing and spoofing attacks. A replay attack will still succeed as the MAC does not have any notion of time and hence the adversary could replay both the value and its corresponding MAC.

Protecting memory against replay attacks requires the processor to have some memory about the recentness of the value. A Merkle Hash Tree, shown in Figure 2, provides an optimal solution for this problem, requiring least amount of on-chip memory. A Merkle hash tree of address range $[A, A + k]$ creates a tree of hashes with $k+1$ leaves corresponding to addresses $A, A+1, \ldots, A+k$. Any write to an address $A + i$ in this address space can modify all the hash values from the leaf node corresponding to $A+i$ upto the root of the tree. Any read from address $A + i$ needs to check the hash values along the path from the leaf node $A + i$ upto the root of the tree. The root of the tree is stored in the trusted processor storage, so that it cannot be tampered. All the other tree nodes along with the leaf nodes can be kept in the untrusted memory.

3 Related Prior Research

Software only security approaches fail in securing embedded systems as the adversary can physically observe or tamper with the state of applications. To

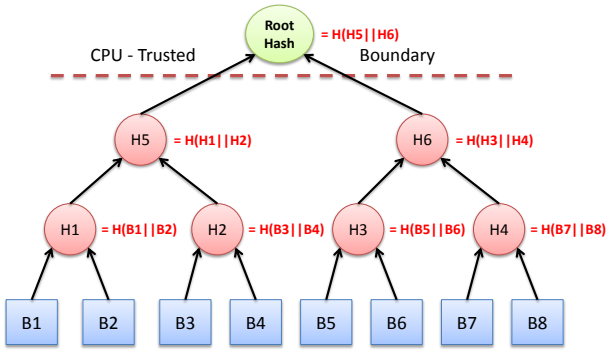


Fig. 2 A Merkle Hash Tree provides time status of memory value.

achieve high level of security, the root of trust must be entrusted on the hardware of these embedded systems. Hence as seen in Section 1 a lot of research has been done in developing secure processor architectures to provide confidentiality and integrity in embedded applications. Even though the hardware approaches provide reliable security against physical and software attacks, considerable modifications are needed to be done before they can be adopted in an energy constrained embedded system.

3.1 Architectural Approaches

Many researchers have proposed energy efficient architectures for embedded systems that aim at providing utmost security while consuming limited energy. For instance, Shi et al. in [29] present a secure and fast architecture for authenticating shared memory. To incorporate memory authentication in their architecture, the authors propose a new scheme of Authentication Speculative Execution that not only is efficient but also offers lower average performance degradation of less than 5%. In the paper [9], the authors present a model for key masking to achieve minimal energy overhead in embedded systems. The experimental results reveal that the technique supports up to 2.5% energy overhead savings. Power-Smart System-On-Chip Architecture [32], presents an architecture for preventing sensitive information leakage via timing, power and electromagnetic channels. This architecture depends on a current sensing module to measure the power and current consumption of the system. They achieve significant success in measuring the current consumption of the system while limiting the power overhead to less than 12% of the total power. In [27], Roger et al. describe an efficient hardware mechanism to protect integrity of softwares by signing each instruction block during program installation with a cryptographically secure signature. This technique serves as a secure and performance efficient alternative to conventional memory integrity verification module. While [10], presents an architectural approach to address mem-

ory spoofing attacks. The data protection techniques proposed in this paper, achieves high level of security with significantly low performance overhead.

3.2 Memory Encryption and Authentication Approaches

Along with architectural techniques, there has been significant research in improving the performance of existing memory integrity verification mechanisms. In [8], the authors combine caches and hash trees to deliver the most performance efficient memory integrity verification scheme. The paper presents CHash and LHash with varying cache block sizes to analyze the performance overhead of each configuration. Similarly in [28], an Address Independent Seed Encryption (AISE) is proposed along with Bonsai Merkle Trees (BMT). Simulation results prove that this technique reduces the overhead on the system by 12% to 2% as compared to traditional approaches. Some other researches [35] and [19], present new low overhead encryption algorithms and cache level tuning to achieve relatively low performance degradation.

Although the above techniques are efficient and secure, the proposed *MEM-DnP* mechanism achieves average energy savings in the range of 85.5% to 99.99%. This reduction is significant as compared to the above mechanisms. Also another novelty of our mechanism is that it uses sensors to detect any potential attack on the memory system.

4 MEM-DnP: The Proposed Detection and Protection Mechanism

The cryptographic MIV mechanism is secure against splicing, spoofing, and replay attacks on the memory. However, the MIV mechanism is a computationally expensive process and consumes overwhelming energy, thus deteriorating the performance of an embedded system. As a case study, Potlapally et. al. [26] present a detailed analysis of energy consumed by commonly used hashing algorithms. This is shown in Table 1. Due to this energy consumption, these hashing algorithms cannot be directly used to secure embedded systems. Hence the need arises to modify the existing MIV mechanism so that they consume minimal energy while preserving their security.

Table 1 Energy Consumption in Memory Integrity Verification Algorithms [26]

	Algorithms					
	MD2	MD4	MD5	SHA	SHA1	HMAC
Energy Consumption (μJ/Byte)	4.12	0.52	0.59	0.75	0.76	1.16

4.1 MEM-DnP Architecture

The intuition behind this approach is as follows. Embedded systems typically employ several sensors to interact with its environment. These sensors can be used to detect any physical attack on the memory. An attack would cause some fluctuations in current, power dissipation or thermal dissipation in the system. Thus corresponding sensors can be used to measure these fluctuations. MIV is performed only when the fluctuations exceed a pre-determined value. This is in contrast to the traditional operation, where the integrity verification process must be performed for all the memory blocks read from the off-chip memory. Thus the MIV process consumes a lot of energy to preserve the integrity of the data in the memory. The use of sensors within the processor chip boundary, to measure critical physical properties has been a rich area of research for many years. For example Oh et al. [25], McGowen et al. [22], and Zhang et al. [37], deploy hardware sensors to measure power dissipation and thermal dissipation in the CPU. Similarly, Muresan et.al. [32] make use of a Current Sensor Module to predict the power consumption of their architecture. Similarly, in this research the hardware sensors in embedded systems are used to detect an attack and only use the MIV in case the attack was detected. This reduces the energy consumption of the system as the MIV mechanism is not used during normal execution.

The general architecture of the *MEM-DnP* Mechanism is given in Figure 3. The Sensor Module(SM), represents the hardware sensors in an embedded system. The SM constantly monitors the memory bus for any fluctuations. The architecture contains a specialized cache — *Hash Cache* to represent the MIV mechanism. The hash cache contains the *Hash Address* and the corresponding *hash* of each memory block written to the off-chip memory. This hash address and the hash is used to verify the integrity of the memory. The processor chip is trusted and hence the SM and Hash cache are located inside the processor boundary. At this time, it is important to emphasize that the research in this paper is only aimed towards MIV and not towards Encryption/Decryption mechanism. An Encryption/Decryption block is included in the general architecture of the MEM-DnP, only to represent a more general secure processor architecture.

The functioning of the architecture can be described in terms of its memory operations, as shown in Figure 4. Here, there are two basic operations — *Write Operation* and *Read Operation*. The memory write operation is shown in Figure 4(a). Here, before writing a memory block to the off-chip memory, the CPU computes its corresponding *hash* and *hash address* and stores it in the *hash cache*. The memory block is then encrypted and stored in the memory.

During the memory read operation, as shown in Figure 4(b), the memory block read from the memory is first decrypted. The SM continuously monitors the memory bus to detect an attack. It maintains a threshold value (V_T) for the fluctuations. If the fluctuations in the system exceed this V_T, it is concluded that there is an anomaly on the system. Only in that case, the hash address of the data is re-computed. The hash address is checked against all

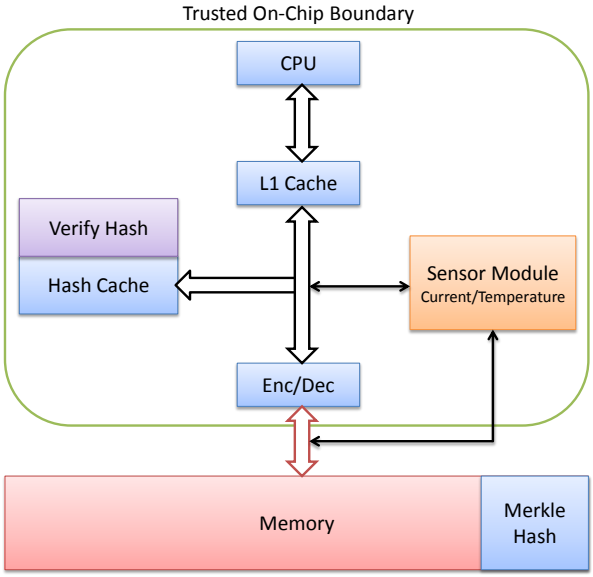
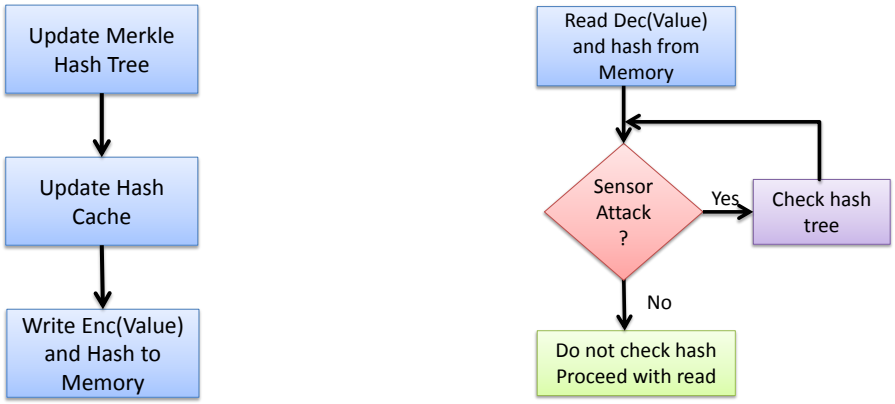


Fig. 3 MEM-DnP Architecture



(a) Write Operation (b) Read Operation

Fig. 4 Memory Operations in MEM-DnP Architecture

the addresses in all the levels of the hash cache. If there is a *hit* in the hash cache, the hash of the data is checked against the hash that is stored in the hash cache. If the hash matches then it is concluded that the state of the data is valid, if no then, it is concluded that the data is corrupted and there is an attack on the system. In such a case, the CPU aborts any operation on this data. This process is recursive and continues for all the memory reads performed in the window of fluctuations exceeding V_T . During this time, if the fluctuations stabilize and fall below the V_T, then it is concluded that

the threat has subsided. At this point, the hash address verification process terminates. The SM and the MIV (hash cache) modules are coupled with each other so that the MIV is only performed when required i.e. at the time of extreme fluctuations or discrepancies. The experimental results in Section 6, prove that this mechanism yields significant energy savings as compared to traditional MIV mechanism.

The pseudo code of the function to compute the hash address of the memory block, is given in Algorithm 1. The function takes three arguments — level of *hash tree*, *memory block address* and *memory block size*. The *Hash Offset* and *Hash Tree Size* are pre-initialized. At first, the memory block number is calculated based on the block address, level and its offset address. Using the memory block number and Hash Tree Size, the hash number for that particular block is computed. Finally, the hash address is calculated using the hash offset, level, hash number and the hash size. This algorithm is later used in the simulations to measure the total number of hash invocations for a particular benchmark per L1 Data cache accesses.

Algorithm 1 Function hash_addr(level, block_addr, block_size) to compute Hash Address of the memory block

Require: HashOffset[Max_Hash_Levels]
Require: HASH_TREE_SIZE
Ensure: level < Max_Hash_Levels
  block_number ⇐ (block_addr - BlockOffset[lvl-1])/block_size
  hash_number ⇐ block_number / HASH_TREE_SIZE
  hash_addr ⇐ HashOffset[level] + (hash_number × hash_size)
  return (hash_addr)

4.2 Disjoint Hash Trees

The SM continuously monitors the memory bus, while the CPU operates on the memory blocks. The MIV module(hash cache) works closely with the SM to initiate verification when discrepancies in the readings are detected. At this time, the MIV module starts re-computing the hash address and the corresponding hash for the verification purpose. This process is cyclic and continues until the fluctuations are stabilized. To reduce the energy consumption in the verification phase, this paper presents a novel mechanism of Disjoint Hash Trees, shown in Figure 5. The idea behind this approach is to divide the original Merkle hash Tree into smaller trees. Based on the location of the infected memory block, a suitable disjoint tree is selected for the verification process. The MEM-DnP mechanism relies on the precision of the SM to accurately locate the infected memory block. The primary advantage of this technique is that the hash tree is not re-constructed over the entire memory. Instead, only that disjoint tree is re-constructed which contains the infected memory block. At this point, it is important to emphasize that the number of Disjoint Trees

constructed in the memory directly relates to the number of secrets that can be stored on-chip. In a traditional implementation of a Merkle hash tree, only one *root hash* i.e. secret is stored on-chip. Whereas in the proposed mechanism, multiple *root hashes* i.e. secrets corresponding to each Disjoint tree are stored on-chip. Hence this approach exploits the availability of memory space on-chip to deliver energy savings during the integrity verification process.

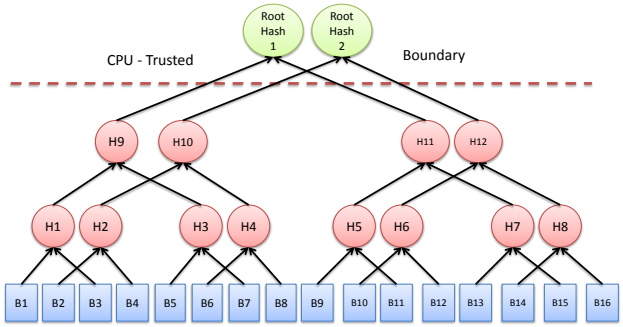


Fig. 5 Disjoint Hash Tree

5 Relation between Process Variations in Sensors and V_T

The measurements recorded from a sensor comprises an error profile that can be attributed to the distribution of noise that affects its readings. The reason for this noise is rooted in Process Variations [12] [11] [23] that occur in the VLSI circuit of a sensor. As shown by Zhang et. al. [36], the noise related to the process variations is the *phase noise* of the circuit. The assumption that "*all transistors are alike*" is invalid in the case of nanochips. It is essential that the variations in two transistors in a chip or different chips in the same design are considered in any design decisions to make circuits robust and to improvise the outcome targeted for the Design for Manufacturing (DFM). The device parameters, chip performance, and the chip yield are affected by process variations. The electrical parameters and the overall performance of the chip is profoundly affected by the process variations and the effects are also evident in the variation in power and delay and other attributes of the chip. The process variations can either be inter die or intra die, it can be random or systematic, it can be correlated or uncorrelated or it can be spatial or temporal. If this variation is not considered, it may lead to significant design errors and loss of yield.

Therefore for past several years, researchers have attempted to find alternative design approaches to ensure that the yield of the VLSI circuit has minimal impact from process variations. There are three main types of approaches to model process variations - statistical design approach, post silicon process

compensation/correction and avoidance of variation induced failures. Statistical design methodology has been widely looked into as an efficient method for yield under process variation. Whereas the post-silicon process compensation/correction has also been extensively explored. It detects process variations using the on-chip process sensors and it may also involve manufacturing test and compensating/correcting circuit parameters that may have deviated due to process variations. And finally avoidance of variations-induced failures is based on the concept of critical path isolation that makes a circuit subject to voltage scaling while still being robust to parametric variations.

In this paper we have chosen the statistical modeling technique to model the process variations (directly related to the sensor noise) in a sensor. The noise samples in a sensor are typically modeled as continuously valued random variables, while the noise waveforms are modeled as random processes. A continuous random variable X with non-negative density F, is specified by its probability density function (PDF) and is given in Equation 1.

$$f_X(x) = \int_{-\infty}^{\infty} f_X(x)\,\mathrm{d}x. \tag{1}$$

Given a PDF for a random variable, it is then possible to calculate the mean (μ) and the variance (σ^2) for it. Both μ and σ^2 are used to estimate the power of noise or the random variable X and are given in Equations 2 and 3.

$$\begin{aligned} \mu_X &= \int_{-\infty}^{\infty} x f(x)\,\mathrm{d}x, \\ \mu_X^2 &= \int_{-\infty}^{\infty} x^2 f(x)\,\mathrm{d}x \end{aligned} \tag{2}$$

$$\sigma_X^2 = \mu_X^2 - (\mu_X)^2. \tag{3}$$

The most common type of random variable used to model noise is the Gaussian random variable. Also, the noise distribution in sensors follow a Gaussian distribution with a zero mean [7]. It is shown in Figure 6. The PDF in a Gaussian distribution is given in Equation 4

$$f(x) = \frac{1}{\sigma_X\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_X}{\sigma_X})^2} \tag{4}$$

In the case of MEM-DnP approach, the sensor module measures many independent signals each having their individual error/noise profiles. Thus mathematically a sensor module can be viewed as a Gaussian process P with 1...N individual continuous valued random variables. While modeling the V_T for the sensors, it is important to know the relationship between individual signals at different time instants. For ease of understanding, lets consider X and Y as two individual normally-distributed random variables with zero mean. Let F(X) and F(Y) be the PDFs of X and Y respectively. Hence the co-variance of X and Y is given in Equation 5.

$$COV(X,Y) = F(XY) - F(X)F(Y) \tag{5}$$

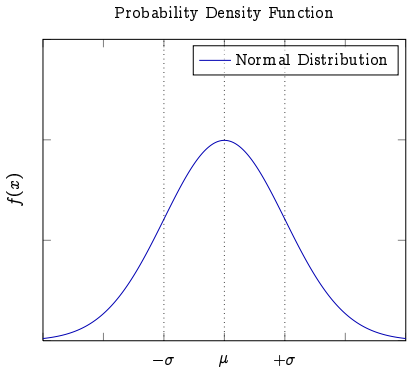


Fig. 6 Gaussian Distribution Curve

However since both X and Y are individual random variables, $F(XY) = F(X)F(Y)$. Therefore $COV(X,Y) = 0$. Thus the two individual random variables X and Y in the same Gaussian process are uncorrelated. Similar is also true for 1...N random variables in the same Gaussian process. Now, since the individual random variables are uncorrelated, the resultant variance of the Gaussian process (σ_P^2) is nothing but the product of variances of N individual random variables in the process, given in Equation 6.

$$\sigma_P^2 = \sigma_1^2 * \sigma_2^2 * \sigma_3^2 .. * ..\sigma_N^2 \tag{6}$$

In the case of a sensor, there are false positives and false negatives affecting its reading. A false positive measurement is the one that will indicate that there is an attack but in reality there is none. Whereas a false negative measurement is the one where there is an actual attack and the sensor failed to detect it. While modeling such a system it is important that both false positives and false negatives are kept in a minimal allowable range. In the MEM-DnP approach, the emphasis is on false positives. In the MEM-DnP simulations, a Random Value (RV) is embedded to simulate one such false positive that indicates fluctuations in the readings. The details about RV are discussed in Section 6.2.2. The Threshold Value V_T is directly related to RV. For a N-bit RV the probability that a fluctuation occurs is given by $\frac{1}{2^N}$. Also in the case of sensors, its mean (μ) and variance (σ^2) is modeled during its design process so as to accommodate the random noise in its readings. Thus given the values of μ, σ and RV it is possible to compute the tolerance (τ) of the sensor circuit. The tolerance τ of a sensor circuit is a point on the Gaussian distribution curve, beyond which we consider that the false positives occur in the system and that it is under an attack. This can be best explained with an example below.

As a case study, Figure 7 presents a statistical characterization of process variation in a 2-input NAND logic gate. This example represents a plot of propagation delay in a 2-input NAND gate. This NAND gate is designed, tested

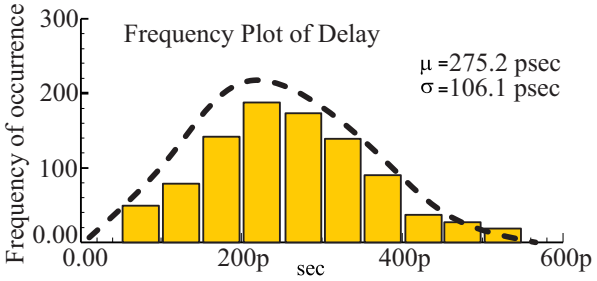


Fig. 7 Example: Propagation Delay PDF in a 2-input NAND Gate

and characterized in a 45 nm CMOS technology. We then used Monte Carlo simulations [5] to translate the process and design variations in the input to the Propagation Delay at the output. For these simulations, the parameters considered for the CMOS transistors in the NAND gate were channel length (L), device width (W), gate oxide thickness (T_{ox}), device threshold voltage (V_{th}) and supply voltage (V_{dd}). It was observed that the propagation delay follows a Gaussian distribution. For ease of simulation, modeling and understanding we have only considered a single 2-input NAND gate. However, a realistic circuit may contains hundreds or even thousands of these 2-input NAND gates. Since the propagation delay PDF is Gaussian in nature, the input distribution at each gate is statistically independent. Therefore in the case of N NAND gates in a sensor circuit, the mean (μ) and the variance (σ^2) of the distribution is given in Equation 7.

$$\mu_{SENSOR} = \sum_{i=1}^{N} \mu_{NAND_i} \qquad \sigma_{SENSOR} = \sqrt{\sum_{i=1}^{N} \sigma^2_{NAND_i}} \tag{7}$$

Similarly, a sensor circuit may contain several logic. But since the output distribution of each gate is Gaussian in nature, the overall PDF of the sensor will also be Gaussian in nature. For instance if $X, Y...N$ are the Gaussian output distributions of the logic gates in a sensor circuit, the overall PDF of circuit $= X + Y + ... + N || X * Y * .... * N || X \oplus Y \oplus .... \oplus N$ is also Gaussian in nature. This curve has a μ=275.2 psec and σ=106.1 psec. As discussed earlier, since the value of μ and σ is known, it is possible to compute the τ for the sensor circuit for RV=16, 20 and 24. In a Gaussian distribution curve this can be achieved using a Cumulative distribution function (CDF) given in Equation 8.

$$CDF = \frac{1}{2}[1 + erf(\frac{x - \mu}{\sqrt{2\sigma^2}})] \tag{8}$$

Here *erf* stands for Error function that is typically used for measurement in probability and statistics. For a random variable X, the CDF represents the probability of X as - $P(X) < x$ on the Gaussian curve. However, to calculate

τ we will fix the probability as - $P(X) > x$. Thus this probability will be (1 - CDF). Therefore by applying to Equation 8, we will get Equation 9.

$$1 - CDF = 1 - \frac{1}{2}[1 + erf(\frac{x-\mu}{\sqrt{2\sigma^2}})] = \frac{1}{2}[1 - erf(\frac{x-\mu}{\sqrt{2\sigma^2}})] \tag{9}$$

Now for RV=16, the probability of a false positive occurring in the sensor's reading is: P(FP)= $\frac{1}{2^{16}}$. In order to compute the value of τ, we will fix this probability. Hence consider that (1 - CDF) = P(FP) = $\frac{1}{2^{16}}$. Also the co-ordinate "x" on the Gaussian curve will represent the value of tolerance - τ of the sensor, where the false positives will occur. Hence given the values of RV, μ and σ, the goal is to compute the value of τ.

$$\frac{1}{2^{16}} = \frac{1}{2}[1 - erf(\frac{\tau-\mu}{\sqrt{2\sigma^2}})] \tag{10}$$

Hence by solving Equation 10, the value of τ can be computed. Therefore solving Equation 10 gives us the value of τ as: $\tau = \mu + 435.01 \times 10^{-12} =$ 710.21 psec. Similarly the tolerance of the sensor can be computed for RV=20 and 24 as well. The tolerance value is a crucial parameter that indicates, where on the Gaussian curve, a false positive reading might be recorded, based on the μ, σ and RV. At this point it is important to understand that the values of μ, σ and RV should be optimally selected during designing and modeling the sensor circuit, so that the value of τ is low. A higher value of τ may lead to severe issues. This is shown in Figure 8. This figure shows a

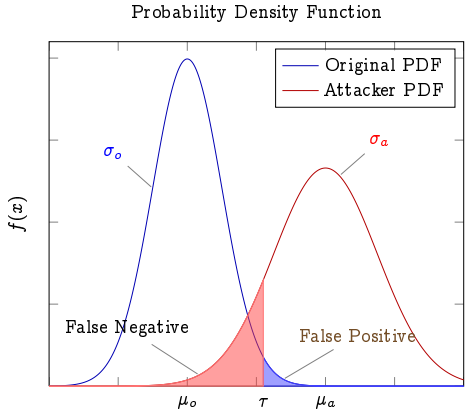


Fig. 8 Tolerance of Sensor as compared to Attacker PDF

Gaussian distribution curve for a typical sensor circuit. It is represented as an Original PDF with mean μ_o and standard deviation σ_o. Now consider that an attacker is performing attacks on the system and the sensor is responsible for measuring the fluctuations in the system. The attacker distribution also

Table 2 Cache Configurations

Cache	Specifications
L1-D Cache	8KB, 2-way, 32B Line
L1-I Cache	16KB, 2-way, 32B Line
L2-D Cache	None
L2-I Cache	None

follows a Gaussian distribution, represented by an Attacker PDF with mean μ_a and standard deviation σ_a. In such scenarios, the value of tolerance τ plays a significant role in the overall security of the system. As the τ is closer to μ_o, the probability of a false positives is more than the probability of false negatives. But as the τ moves further away from μ_o, the overlap of Attacker PDF on Original PDF increases and thus the probability of false negatives increases than the probability of false positives. This is a critical security condition as the sensor will fail to detect a legitimate attack measurement. Therefore the value of τ should be optimally designed depending on the value of μ_o, σ_o and RV.

6 Experimental Results

This section presents detailed analysis of the energy consumption involved in traditional MIV process. This is compared to the energy consumption of the proposed *MEM-DnP* Mechanism.

6.1 Simulation Framework

The simulation framework is based on Simplescalar Tool Set [4], which is configured to execute ARM binaries. Since the primary goal of this paper is to demonstrate the energy efficiency of the proposed MEM-DnP mechanism in embedded systems, MiBench [20] embedded benchmark suite has been used, that best replicates the variety of practical applications run on embedded devices. The results obtained from different benchmark programs are presented, to thoroughly demonstrate the efficiency of the proposed mechanism. All the simulations performed are cache based simulations, using the *sim-cache* simulator in simplescalar. The Cache configurations used for the simulations is given in Table 2. Here, the Level 1 Cache configurations are selected to match the typical configurations of Embedded ARM processors [1].

6.2 Energy Consumption

6.2.1 Baseline Simulations

As described in Section 2, MIV is performed by constructing a Merkle Hash Tree in the memory and storing the final hash, known as the root hash in the on-chip memory storage. For every memory read, the Merkle hash tree is re-constructed and the resulting root hash is compared with the one already present on-chip. If both the values match, then it is concluded that the memory block is verified else it is infected. Typically, this is implemented by partitioning the memory into fixed sized blocks (usually same as the cache size) and generating multiple levels of hashes. Hence energy is consumed while generating the hash from the memory all the way up to its root. To measure this, we have presented Algorithm 1 in Section 4. Here the algorithm is used to measure the number of hash invocations required per data miss in the Level 1 Data Cache. Hence, given that the energy consumption per hash invocation is known, the results obtained from the algorithm can be used to calculate the total energy consumption of the MIV mechanism.

Table 3 Basecase Simulations showing Total Hash Invocations and the Average Hash Rate

Benchmark	DL1 Misses	Total Hash Invocations	Avg. Hash Rate
dijkstra_small	1501134	21007122	13.99
fft_large	6570198	91732597	13.96
jpeg_large	2923393	40925500	14.00
lame_large	39221353	549097030	14.00
patricia_large	9138017	127930138	14.00
qsort_large	7114792	99603844	14.00
math_large	1318621	18090557	13.72
sha_large	264246	3287898	12.44
stringsearch_large	76794	1070293	13.94
susan_large	109933	1537169	13.98
blowfish_large	5074709	70618620	13.92
crc_large	11237338	97672098	8.69

The Table 3 shows the DL1 Misses, Total Hash Invocations and the Average Hash Rate for 12 embedded benchmark applications. The average hash rate is calculated by dividing the total hash invocations by the DL1 Misses in each of the benchmark applications. Since the average hash rate is directly related to the total hash invocations, it is also directly related to the energy consumption in the MIV mechanism. More the average hash rate, more are the total hash invocations and thus more energy is consumed by the MIV mechanism and vice versa. Thus in this paper, the emphasis is on reducing the average hash rate by using a sensor module to detect possible attacks on the system. This is explained in detail in Section 6.2.2 along with its results.

6.2.2 Energy Consumption in MEM-DnP Mechanism

The SM and the MIV module work together to detect and protect from physical attacks on the memory as explained in the discussion in Section 4.1. The accuracy of sensors is responsible for precisely detecting the infected memory blocks. The functioning of MEM-DnP mechanism results in two false positives. The first false positive arises due the working of sensors. A typical non-ideal sensor would manifest some abnormal fluctuations. This is modelled in the simulation framework by using a *Random Value* (RV), which is a random number generator that can generate 16 bit, 20 bit, or 24 bit random numbers. The RV represents the probability of occurrence of such fluctuations in the sensor readings. For an N-bit RV the probability that an fluctuation occurs is given by $\frac{1}{2^N}$. It is important to stress, that these fluctuations may arise even if there is no potential attack on the system. The second false positive again relates to the working of the sensor. At the time of fluctuations in the readings, a sensor typically has the capability to auto-correct itself and stabilize its readings. For an ideal sensor the time required for auto-correction is 0. However, in practice a finite amount of time is always required for the sensor to auto-correct itself. This is modelled in the simulation framework through the use of *Window Size*. The Window Size corresponds to the number of memory accesses required for the sensors to stabilize. The Window Size can take 3 possible values — 2000, 8000, and 15000 memory accesses. Finally, the third parameter is the *Disjoint Trees* which corresponds to the number of secrets or root hashes that can be stored on-chip. It depends on the amount of memory available in the on-chip storage. The number of Disjoint Trees are tested for three values — 16, 64, and 128.

The simulation Algorithm 2 shows the steps involved in the simulation of *MEM-DnP* mechanism. A predetermined *Attack Seed* of the same size as that of the Random Value is embedded in the simulation framework. During the simulations, when the values of Random Value and Attack Seed match, it concluded that fluctuations in the sensor readings have exceeded V_T and there is an anomaly in the system. At this point, the MIV module generates the Disjoint hash tree and computes the hash address and hash of the memory blocks and proceeds with the verification process. This process is recursive i.e. the Random Value and the Attack Seed are constantly checked to detect if the match exists. If there is a match again, it indicates that the anomaly persists and the hash verification process continues. If the sensor measurements stabilize, the MIV module stops the hash verification process.

The *MEM-DnP* mechanism is tested for all possible combinations of the two false positives — Random Value and Window Size and the parameter Disjoint Trees. Using Algorithm 2, simulations were performed on the same embedded benchmarks applications that are used in basecase simulations and the new *Average Hash Rate* was recorded. The new average hash rate is compared with the one in basecase simulations. Note, lower the average hash rate, greater are the energy savings. Figures 9(a), 9(b) and 9(c) show the reduction in average hash rate for Disjoint Trees = 16, 64, and 128 respectively. The

Algorithm 2 Simulation Algorithm

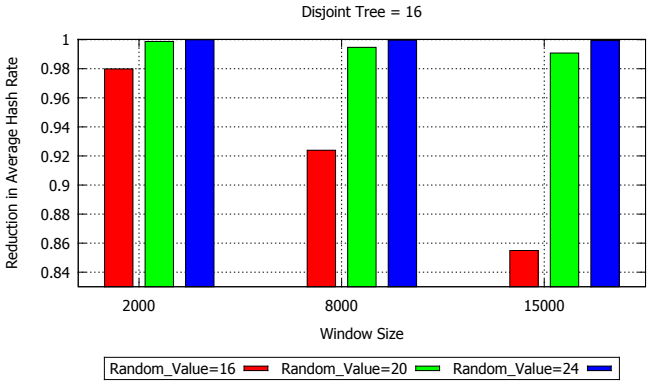
Require: Random Value, Window size, Disjoint trees.
1 SM monitors the system to detect an attack.
2 if Random attack seed = Random Block. *(Fluctuations $>V_T$. Hence there is an Anomaly)*
3 Invoke MIV for given Window size and Generate disjoint trees.
4 During this keep repeating steps 2 & 3.
5 If there is match again. *(Anomaly persists)*
6 Extend the Window size; keep generating disjoint trees.
7 Else attack has subsided.
8 Stop the MIV process.
9 Keep repeating step 2.

reduction in average hash rate is plotted for all possible values of Window Size and Random Value. Here the reduction in average hash rate is almost 1.00 i.e. 100% for Random Value - 24. This is expected as the probability that the random number generator would generate a Random Value matching the Attack is the least for value - 24 as compared to 16 and 20. Also except for Disjoint Tree-128, the reduction in average hash rate is the least for Random Value-16 and Window Size-15000. This is due to fact that the Random Value-16 will result in a higher probability of a match with the Attack seed and the number of memory accesses (Window Size) for which MIV functions is significantly higher. Hence the reduction in average hash rate will be lower.

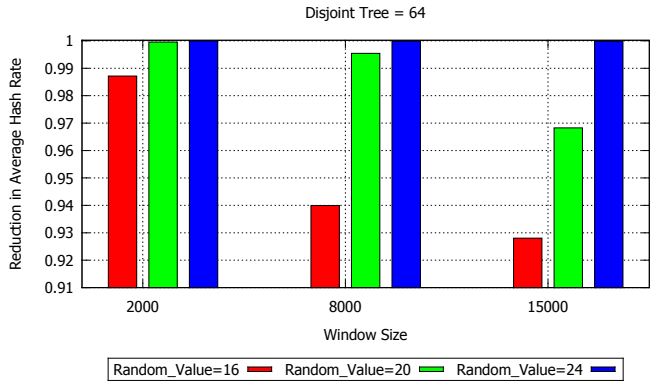
Figures 10(a), 10(b), and 10(c) show the reduction in average hash rate for Window Size = 2000, 8000 and 15000 respectively. The reduction in average hash rate is plotted for all possible values of Disjoint Tree and Random Value. Here again the reduction in average hash rate is almost 1.00 i.e. 100% for Random Value - 24, whereas for Random Value - 16 it is the least.

Finally Figures 11(a), 11(b), and 11(c) show the reduction in average hash rate for Random Value = 16, 20 and 24 respectively. The reduction in average hash rate is plotted for all possible values of Window Size and Disjoint Trees. Here the reduction in average hash rate is highest for Window Size-2000. This is expected as the number of memory accesses for which the MIV functions is significantly low and hence the average hash rate is low.

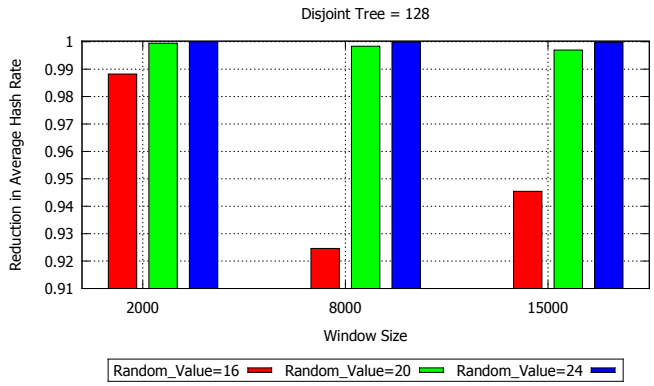
Based on the reduction in average hash rate, the *Average Energy Savings* with regards to the basecase simulations, are calculated. Tables 4, 5 and 6 show the Average Energy Savings for Disjoint Trees = 16, 64 and 128 respectively. The energy savings are least for Disjoint Tree = 16, Window Size = 15000 and Random Value = 16 with approximately 85.5%. While the energy savings are the highest for Disjoint Trees = 64, Window Size = 2000, and Random Value = 24 with 99.998%.



(a) for Disjoint Trees-16

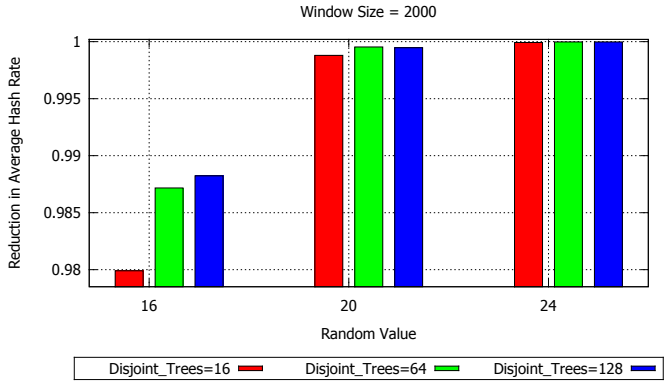


(b) for Disjoint Trees-64

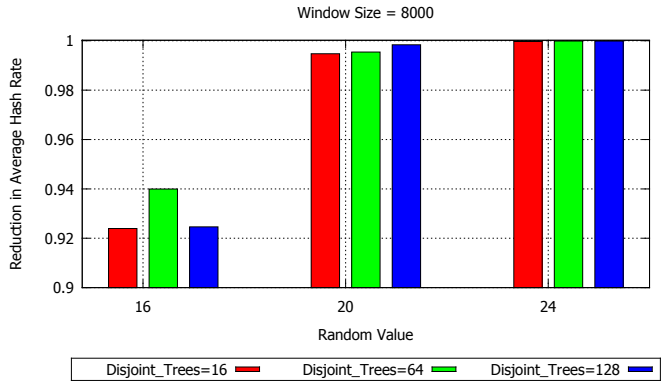


(c) for Disjoint Trees-128

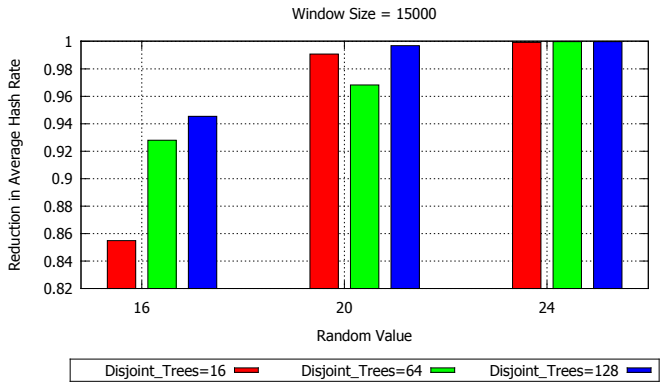
Fig. 9 Reduction in Average Hash Rate for different Disjoint Trees.



(a) for Window Size-2000

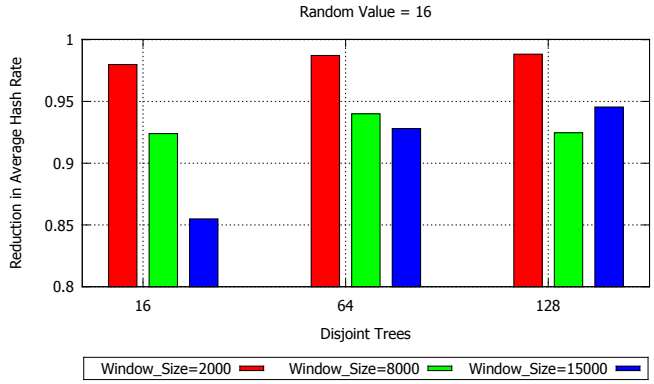


(b) for Window Size-8000

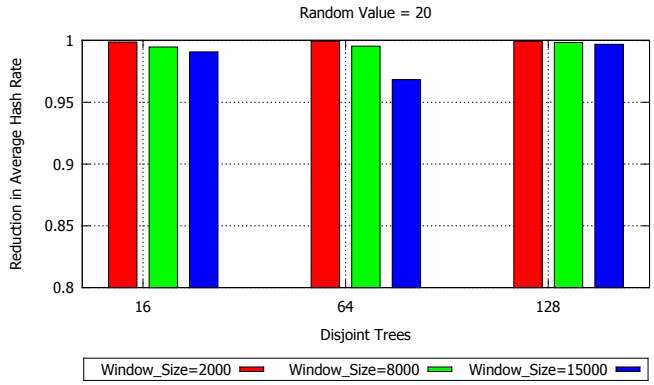


(c) for Window Size-15000

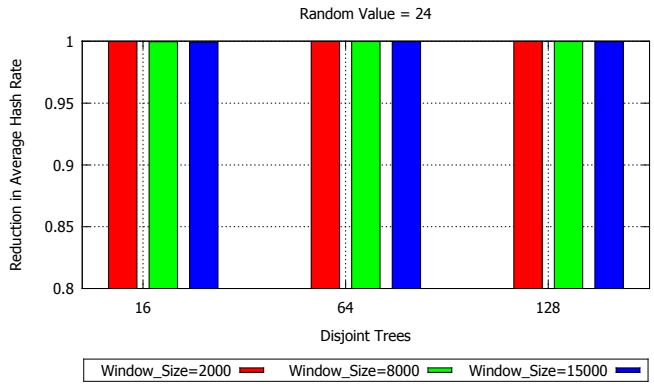
Fig. 10 Reduction in Average Hash Rate for different Window Sizes.



(a) for Random Value-16



(b) for Random Value-20



(c) for Random Value-24

Fig. 11 Reduction in Average Hash Rate for different Random Value.

Table 4 Average Energy Savings for Disjoint Tree = 16

Disjoint Tree = 16		
Window Size	Random Value	Average Energy Savings
2000	16	97.990
	20	99.880
	24	99.993
8000	16	92.394
	20	99.475
	24	99.971
15000	16	85.492
	20	99.080
	24	99.934

Table 5 Average Energy Savings for Disjoint Tree = 64

Disjoint Tree = 64		
Window Size	Random Value	Average Energy Savings
2000	16	98.717
	20	99.954
	24	99.998
8000	16	93.995
	20	99.543
	24	99.989
15000	16	92.804
	20	96.828
	24	99.985

Table 6 Average Energy Savings for Disjoint Tree = 128

Disjoint Tree = 64		
Window Size	Random Value	Average Energy Savings
2000	16	98.823
	20	99.947
	24	99.998
8000	16	92.461
	20	99.837
	24	99.989
15000	16	94.546
	20	99.696
	24	99.984

7 Conclusions

Security is a primary concern in embedded systems as they are vulnerable to physical and software attacks. Hardware security architectures can be used to provide a trusted and secure environment for embedded system. However, the hardware security mechanisms implemented in these secure architectures

result in exorbitant energy consumption. An embedded system is highly energy constrained and cannot afford to sacrifice speed and performance for security. Hence the need arises to design new energy efficient security mechanism for embedded systems.

In this paper, the focus is on reducing the energy consumed by the secure Memory Integrity Verification mechanism. Hence it proposes the *MEM-DnP* mechanism, a novel approach to achieve energy efficient memory integrity verification in embedded systems. This mechanism relies on a sensor module to detect any discrepancies in the system. The memory integrity verification is only done in an event of an anomaly i.e. the fluctuation exceed the threshold value V_T. Hence a lot of energy is saved by de-coupling the integrity verification process during normal execution of the system. The simulation results prove the average energy savings are in the range of 85.5% to 99.998% as compared to the basecase simulations with traditional memory integrity verification techniques.

Currently, research is in progress to propose purely architecture specific energy efficient mechanisms to achieve memory integrity verification in embedded systems. In addition to this, the motivation is to measure the total energy consumption of an embedded processor and compare it with the energy consumption of the cryptographic memory integrity verification mechanisms. This would give an exact measure of the energy savings that can be achieved by the proposed energy efficient MIV techniques, with respect to the entire embedded processor energy consumption.

Acknowledgements A shorter version of this research in the current archival journal is presented in the blind-reviewed conference paper [24]. In this version, we have discussed in detail the concept of Memory Integrity Verification (MIV) in Section 2, expanded the architecture to include memory operations and the hash address computation algorithm in Section 4.1, provided a theoretical basis for the optimal selection of V_T in the sensor in Section 5, added simulation framework details along with an elaborate simulation algorithm to implement *MEM-DnP* mechanism and have presented new experimental evaluation results using ARM architecture and MiBench benchmarks in Section 6.2.

References

1. ARM Architecture Reference Manual. `http://www.altera.com/literature/third-party/ddi0100e_arm_arm.pdf/` (2000)
2. Cyber Security Watch Survey. `http://www.sei.cmu.edu/newsitems/cybersecurity_watch_survey_2011.cfm` (2011)
3. Mobile and Smart Device Security Survey 2011. `http://images.tmcnet.com/tmc/whitepapers/documents/whitepapers/2011/4683-mobile-smart-device-security-survey-2011.pdf` (2011)
4. Austin, T., Larson, E., Ernst, D.: Simplescalar: an infrastructure for computer system modeling. Computer **35**(2), 59 –67 (2002). DOI 10.1109/2.982917
5. Burch, R., Najm, F., Yang, P., Trick, T.: A monte carlo approach for power estimation. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **1**(1), 63 –71 (1993). DOI 10.1109/92.219908
6. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer Verlag, Berlin, Heidelberg, New York (2002)

7. Elnahrawy, E., Nath, B.: Cleaning and querying noisy sensors. In: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, WSNA '03, pp. 78–87. ACM, New York, NY, USA (2003). DOI 10.1145/941350.941362. URL http://doi.acm.org/10.1145/941350.941362
8. Gassend, B., Suh, G., Clarke, D., van Dijk, M., Devadas, S.: Caches and hash trees for efficient memory integrity verification. In: High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on, pp. 295 – 306 (2003). DOI 10.1109/HPCA.2003.1183547
9. Gebotys, C.: Low energy security optimization in embedded cryptographic systems. In: Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on, pp. 224 – 229 (2004). DOI 10.1109/CODESS.2004.240744
10. Gelbart, O., Leontie, E., Narahari, B., Simha, R.: Architectural support for securing application data in embedded systems. In: Electro/Information Technology, 2008. EIT 2008. IEEE International Conference on, pp. 19 –24 (2008). DOI 10.1109/EIT.2008.4554261
11. Ghai, D., Mohanty, S., Kougianos, E.: Design of parasitic and process-variation aware nano-cmos rf circuits: A vco case study. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **17**(9), 1339 –1342 (2009). DOI 10.1109/TVLSI.2008.2002046
12. Ghai, D., Mohanty, S., Kougianos, E.: Variability-aware optimization of nano-cmos active pixel sensors using design and analysis of monte carlo experiments. In: Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design, pp. 172 –178 (2009). DOI 10.1109/ISQED.2009.4810289
13. Gomathisankaran, M., Keung, K.M., Tyagi, A.: Rebel - reconfigurable block encryption logic. In: SECRYPT, pp. 312–318 (2008)
14. Gomathisankaran, M., Lee, R.: Tantra : A fast prng algorithm and its implementation. In: H.R. Arabnia, K. Daimi (eds.) Proceedings of the 2009 International Conference on Security & Management, SAM 2009, July 13-16, 2009, Las Vegas Nevada, USA, 2 Volumes, pp. 593–598. CSREA Press (2009)
15. Gomathisankaran, M., Lee, R.B.: Maya: A novel block encryption function. In: International Workshop on Coding and Cryptography (2009). URL http://viper.eng.iastate.edu/gmdev/pubs/maya.pdf
16. Gomathisankaran, M., Tyagi, A.: Tiva : Trusted integrity verification architecture. In: R. Safavi-Naini, M. Yung (eds.) DRMTICS : Technologies, Issues, Challenges and Systems, First International Conference, DRMTICS 2005, Sydney, Australia, October 31 - November 2, 2005, Revised Selected Papers, pp. 13–31. Springer (2005). DOI 10.1007/11787952_2. URL http://viper.eng.iastate.edu/gmdev/pubs/drmtics05.pdf
17. Gomathisankaran, M., Tyagi, A.: Architecture support for 3d obfuscation. Computers, IEEE Transactions on **55**(5), 497 – 507 (2006). DOI 10.1109/TC.2006.68
18. Gomathisankaran, M., Tyagi, A.: Architecture support for 3d obfuscation. IEEE Trans. Computers **55**(5), 497–507 (2006). DOI 10.1109/TC.2006.68. URL http://viper.eng.iastate.edu/gmdev/pubs/ieeeTC06.pdf
19. Gordon-Ross, A., Vahid, F., Dutt, N.: Automatic tuning of two-level caches to embedded applications. In: Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, vol. 1, pp. 208 – 213 Vol.1 (2004). DOI 10.1109/DATE.2004.1268850
20. Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., Brown, R.: Mibench: A free, commercially representative embedded benchmark suite. In: Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on, pp. 3 – 14 (2001). DOI 10.1109/WWC.2001.990739
21. Lie, D., Thekkath, C.A., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J.C., Horowitz, M.: Architectural support for copy and tamper resistant software. In: Architectural Support for Programming Languages and Operating Systems, pp. 168–177 (2000). URL citeseer.nj.nec.com/lie00architectural.html
22. McGowen, R., Poirier, C., Bostak, C., Ignowski, J., Millican, M., Parks, W., Naffziger, S.: Power and temperature control on a 90-nm itanium family processor. Solid-State Circuits, IEEE Journal of **41**(1), 229 – 237 (2006). DOI 10.1109/JSSC.2005.859902
23. Mohanty, S.P., Kougianos, E.: Simultaneous power fluctuation and average power minimization during nano-cmos behavioral synthesis. In: Proceedings of the 20th International Conference on VLSI Design [23], pp. 577–582

24. Nimgaonkar, S., Gomathisankaran, M.: Energy efficient memory authentication mechanism in embedded systems. In: Electronic System Design (ISED), 2011 International Symposium on, pp. 248 –253 (2011). DOI 10.1109/ISED.2011.34
25. Oh, D., Kim, N.S., Chen, C., Davoodi, A., Hu, Y.H.: Runtime temperature-based power estimation for optimizing throughput of thermal-constrained multi-core processors. In: Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific, pp. 593 –599 (2010). DOI 10.1109/ASPDAC.2010.5419815
26. Potlapally, N.R., Ravi, S., Raghunathan, A., Jha, N.K.: Analyzing the energy consumption of security protocols. In: Proceedings of the 2003 international symposium on Low power electronics and design, ISLPED '03, pp. 30–35. ACM, New York, NY, USA (2003). DOI 10.1145/871506.871518. URL `http://doi.acm.org/10.1145/871506.871518`
27. Rogers, A., Milenkovic, M., Milenkovic, A.: A low overhead hardware technique for software integrity and confidentiality. In: Computer Design, 2007. ICCD 2007. 25th International Conference on, pp. 113 –120 (2007). DOI 10.1109/ICCD.2007.4601889
28. Rogers, B., Chhabra, S., Solihin, Y., Prvulovic, M.: Using address independent seed encryption and bonsai merkle trees to make secure processors os- and performance-friendly. In: Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on, pp. 183 –196 (2007). DOI 10.1109/MICRO.2007.16
29. Shi, W., Lee, H.H., Ghosh, M., Lu, C.: Architectural support for high speed protection of memory integrity and confidentiality in multiprocessor systems. In: Parallel Architecture and Compilation Techniques, 2004. PACT 2004. Proceedings. 13th International Conference on, pp. 123 – 134 (2004). DOI 10.1109/PACT.2004.1342547
30. Suh, G., Clarke, D., Gasend, B., van Dijk, M., Devadas, S.: Efficient memory integrity verification and encryption for secure processors. In: Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on, pp. 339–350 (2003). DOI 10.1109/MICRO.2003.1253207
31. Suh, G., O'Donnell, C., Devadas, S.: Aegis: A single-chip secure processor. Design Test of Computers, IEEE **24**(6), 570 –580 (2007). DOI 10.1109/MDT.2007.179
32. Vahedi, H., Gregori, S., Zhanrong, Y., Muresan, R.: Power-smart system-on-chip architecture for embedded cryptosystems. In: Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on, pp. 184 –189 (2005). DOI 10.1145/1084834.1084883
33. White, S., Comerford, L.: Abyss: an architecture for software protection. Software Engineering, IEEE Transactions on **16**(6), 619 –629 (1990). DOI 10.1109/32.55090
34. Williams, D., Sirer, E.G.: Optimal parameter selection for efficient memory integrity verification using merkle hash trees. In: Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on, pp. 383–388 (2004). DOI 10.1109/NCA.2004.1347805
35. Yan, C., Rogers, B., Englender, D., Solihin, D., Prvulovic, M.: Improving cost, performance, and security of memory encryption and authentication. In: Computer Architecture, 2006. ISCA '06. 33rd International Symposium on, pp. 179 –190 (2006). DOI 10.1109/ISCA.2006.22
36. Zhang, C., Srivastava, A., Wu, H.C.: Hot-electron-induced effects on noise and jitter in submicron cmos phase-locked loop circuits. In: Circuits and Systems, 2005. 48th Midwest Symposium on, pp. 507–510 Vol. 1 (2005). DOI 10.1109/MWSCAS.2005.1594149
37. Zhang, L., Bai, L., Dick, R., Shang, L., Joseph, R.: Process variation characterization of chip-level multiprocessors. In: Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE, pp. 694 –697 (2009)
38. Zhuang, X., Zhang, T., Pande, S.: Hide: an infrastructure for efficiently protecting information leakage on the address bus. SIGARCH Comput. Archit. News **32**(5), 72–84 (2004). DOI 10.1145/1037947.1024403. URL `http://doi.acm.org/10.1145/1037947.1024403`