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

Data Remanence Effects on Memory Based Entropy Collection for RFID Systems

Nitesh Saxena · Jonathan Voris

Received: date / Accepted: date

Abstract Random number generation is a fundamental security primitive. This relatively simple requirement is beyond the capacity of passive RFID (Radio Frequency Identification) tags, however. A recent proposal, Fingerprint Extraction and Random Number Generation in SRAM (FERNS), uses onboard RAM as a randomness source. Unfortunately, practical considerations prevent this approach from reaching its full potential. First, the amount of RAM available for utilization as a randomness generator may be severely restricted due to competition with other system functionalities. Second, RAM is subject to *data remanence*; there is a period after losing power during which stored data remains intact in memory. Thus, after memory has been used for entropy collection once it will require time without power before it can be reused. This may lead to unacceptable delays in a usable security application.

In this paper, the practical considerations that must be taken into account when using RAM as an entropy source are demonstrated. The implementation of a true random number generator on Intel's WISP (Wireless Identification and Sensing Platform) RFID tag is also presented, which is the first to the authors' best knowledge. By relating this to the requirements of some popular RFID authentication protocols, the practicality of utilizing memory based randomness techniques on resource constrained devices is assessed.

Keywords RFID, True Random Number Generation, Memory, Authentication Protocols

1 Introduction

The importance of Radio Frequency Identification (RFID) technology continues to grow as RFID tags see deployment in an ever expanding variety of applications and settings. Consequently, RFID security and privacy continues to be carefully scrutinized by the research community. Providing security and privacy services in RFID systems presents unique challenges due to the highly constrained nature of RFID enabled devices. There has been much work on the development of security and privacy mechanisms and protocols that attempt to take the limited capabilities of RFID tags into account. Most, if not all, of these schemes rely on the presence of one of the most fundamental cryptographic primitives, random number generation.

While modern general purpose computers have many techniques available for the generation of high quality random numbers, even this relatively simple requirement is beyond the capacity of current passive RFID tags. The Electronic Product Code (EPC) air interface specification for the most recent (Class 1 Generation 2) variety of RFID tags includes a provision for pseudorandom number generation [1]. The resulting random values are intended to be used only as a collision prevention measure, however. When combined with the economic considerations of these ultra-low cost devices, the values produced by these generators are unlikely to be of high enough quality to be used as a source of cryptographic randomness.

Nitesh Saxena
Department of Computer Science and Engineering
Polytechnic Institute of New York University
Six MetroTech Center
Brooklyn, NY 11201 USA
E-mail: nsaxena@poly.edu

Jonathan Voris
Department of Computer Science and Engineering
Polytechnic Institute of New York University
Six MetroTech Center
Brooklyn, NY 11201 USA
E-mail: jvoris@isis.poly.edu

Recently, Holcomb, Burleson, and Fu proposed an alternative approach to entropy collection titled Fingerprint Extraction and Random Number Generation in SRAM (FERNS) [14,15]. This technique involves the use of onboard RAM as the source of true randomness. FERNS works by repurposing blocks of RAM into physical fingerprints which, when run through a random number extractor (such as a hash function), produce random output. This technique is quite promising as any device, regardless of its constraints, will contain some amount of onboard memory from which randomness can be drawn. In addition to random number generation, FERNS was also shown to be capable of creating fingerprints with which RFID tags can be uniquely identified.

Unfortunately, practical considerations prevent the FERNS approach to random number generation from reaching its full theoretical potential. Since FERNS relies on preexisting memory circuitry as a source of entropy, it must compete with other system functionalities for use of this shared resource. Other code running on a RFID tag, such as the EPC protocol stack itself (that is, the implementation of the protocol in software), will likely be occupying the device's memory at any given point during execution. As such, the amount of uninitialized RAM available for utilization as a randomness generator may be severely restricted.

Furthermore, RAM is subject to a phenomenon known as *data remanence*. While it is still volatile in the traditional sense, due to properties of the underlying hardware such memory retains its contents while receiving power and for a duration of several seconds afterwards. Thus, there is a time period after losing power during which stored data remains intact in memory. This means that after a portion of memory has been used for entropy collection once, it will require a relatively extended period of time without power before it can again be used in this capacity. In a usable RFID based security application which requires multiple random numbers this may lead to unacceptably high delays.

1.1 Contributions

In this paper, the practical considerations that must be taken into account when using RAM as an entropy source are demonstrated. We discuss the implementation of a true random number generator on Intel's Wireless Identification and Sensing Platform (WISP), which is a passive, programmable RFID tag [21,25]. This is the first such implementation to the authors' best knowledge [13]. Using this as a basis, how many bits of randomness one can expect to derive from a RFID device's memory at a given time is demonstrated. The results of this indicate that at most 309 bits of randomness can be derived from a tag with 512 bytes of RAM, with this figure dropping sharply as tag memory capacity decreases.

The implications of data remanence on RFID tags and the rate at which random number generation can be performed are then analyzed. By relating this process to the requirements of some popular RFID authentication protocols, the practicality of utilizing memory based randomness techniques on resource constrained devices is assessed. As an example, the implications that RAM based randomness derivation would have on the usage model of a typical RFID enabled access card are discussed. In addition, potential attacks that could be launched on RFID system while this method is in use are introduced.

This is an extension of work presented at the The Fifth Workshop on RFID Security [23].

1.2 Paper Organization

The rest of this work is organized in the following fashion. Section 2 introduces the fundamentals of RFID systems and discusses related work. In Section 3, the performed experiments are explained in detail. Section 4 provides a discussion of the practicality of the studied approach based on these experiments. Finally, Section 5 summarizes the results of these tests.

2 Background

2.1 RFID Overview

RFID is an increasingly popular technology for computerized identification. An RFID infrastructure consists of tags and readers. Tags are small transponders that store data about their corresponding subject, such as an ID value. Readers are used to query and identify these tags over a wireless channel. In most cases, tags are passive or semi-passive, meaning they derive the power to transmit data to a reader from the electromagnetic field generated when a reader sends a query to a tag. Additionally, tags typically have memory only in the range of 32-128 bits, perhaps just enough to store a unique identifier [17].

These ultra-low memory, computational, and power constraints are necessitated by the fact that RFID tags are designed to be placed ubiquitously in consumer products, appliances, and even users themselves (in the case of implanted payment tokens, for example). The minimalist capabilities of these tags present unique privacy and security challenges, the issue of random number generation being foremost among them. How can a device with limited power, memory, computational capabilities, and user interfaces generate high quality random numbers?

2.2 WISP Tags

In order to investigate this question, a special type of RFID tag designed by Intel Research known as a Wireless Identification and Sensing Platform, or WISP [21, 25], was utilized. WISPs are passively powered RFID tags that are compliant with the EPC protocol. Specifically, the 4.1 version of WISP hardware was used, which partially implements Class 1 Generation 2 of the EPC standard. By following this standard and deriving power only from the transmissions of a commercial RFID reader, WISPs closely model the type of RFID tag one might expect to find in a typical contactless access token.

Where WISPs differ from standard tags, however, is in their inclusion of an onboard Texas Instruments MSP430F2132 microcontroller. This 16-bit MCU features an 8 MHz clock rate, 8 kilobytes of flash memory, and 512 bytes of RAM. WISPs are the first programmable passive RFID devices. Unlike standard RFID tags, which are fixed function and state machine based, the flexibility of WISP tags allowed for the implementation of a random number generator and the probing of memory on a live, passive RFID device.

2.3 Random Number Generation Based on RAM

A recent proposal to address the difficulty of generating random numbers on a passive RFID device is called FERNS [14,15]. Instead of treating uninitialized memory as a indeterminate blank slate, FERNS works by considering this unused memory to be a fingerprint. This fingerprint can be used in two complimentary ways. The first is as a means of identifying a given RFID tag through the underlying physical characteristics of memory. The second is as a potential source of entropy. The focus in this paper is on the latter application.

Each unpowered RAM cell starts in an unstable state, then moves to a stable state, representing either a '0' or a '1', once supplied with power. Which of the two bit states the cell enters upon first receiving power is dependent on the threshold voltage mismatch as well as the thermal and shot noise of the cell. A large threshold voltage mismatch will cause a RAM cell to reliably initialize to one bit value or the other. A small mismatch, on the other hand, will be overshadowed by the cell's noise, causing it to take on a value randomly at power up. The probability of every RAM cell showing no random behavior is statistically insignificant. It is the physical noise of these RAM cells that supply entropy in the FERNS method.

Due to physical impurities, the random, well threshold matched cells will be randomly scattered throughout the RAM. As these bits do not occur in convenient proximity to one another, an extractor is necessary to pull these desirable bits from the RAM sequence. A hash function can be used in this capacity. The PH universal hash function of [27] is recommended due to its suitability for implementation in resource-limited hardware [14,15]. This function is a variant of the NH hash function that was designed to be efficient in software in order to accelerate the UMAC message authentication algorithm [5]. PH is the result of a retooling of the NH function in order to remove the need to perform carry operations, which makes the function more suitable for a hardware implementation in terms of speed, space, and power consumption. PH is defined in Equation 1. Blocks of uninitialized memory are provided to the hash function as both key (k_i) and message (m_i) inputs; the output of the hash function can then be used as a random bitstream.

$$PH_k(m) = \sum_{i=1}^{8} (m_{2i-1} + k_{2i-1})(m_{2i} + k_{2i}) \tag{1}$$

2.4 Data Remanence

Since computer memory is volatile, it is a common belief that data stored in RAM is completely lost as soon as it ceases to be supplied with power. This is not entirely accurate, however. While the contents of unpowered RAM will certainly degrade over time, the decay process takes several seconds to begin and several more to complete [24,11]. This process is due to the low-level electrical components that comprise a RAM chip. In SRAM, for instance, data is stored by setting the state of a flip-flop. This state is maintained as long as the flip-flop continues to receive power [9].

This circuit does not lose its state immediately upon loss of power, however. The state will remain for a short interval of time. Thus, there is a brief time period after losing power during which stored data remains intact. If power is again supplied before the end of this window the decay process is halted. While the speed of data loss varies greatly between individual chips, the rate of RAM decay is largely a function of temperature. At high temperatures the degradation process is accelerated, while if cooled to a low enough temperature the decay process can effectively be halted indefinitely.

The phenomenon of data remanence has serious repercussions for computer security. Many times cryptographic data, such as keys, are stored in RAM. If an adversary can gain physical access to a RAM chip containing sensitive material, remove it, and read it on a different device before it fully decays, he or she can potentially recover a full image of the memory contents, including any stored secrets. Even if the decay process has already started to set in, statistical techniques can be used to recover lost bits [11]. While data remanence plays an important part in the work presented in this paper, it does not involve the recovery of data from memory. Instead, the implications of data remanence on the frequency of RAM initialization are explored.

2.5 RFID Authentication

One of the most important RFID security challenges is tag authentication. RFID tags are designed to respond promiscuously to any query from a compatible reader. This behavior makes the forging and duplication of RFID tags a relatively straightforward process. Since tags respond to any query, there is little preventing an adversary with the proper equipment from obtaining a tag's data, then creating a new tag containing the exact same value [17]. In many cases, traditional cryptographic solutions cannot be used by RFID tags due to their low computational and memory capabilities.

Several new solutions have been developed to address these problems; one of the best known is the HB+ protocol [17]. HB+ is a challenge-response scheme based on the HB human authentication protocol [16] that was designed with the computational and memory constraints of RFID tags in mind, requiring only bitwise logic operators for computation. The only other requisite of HB+ is for tags to be capable of generating high quality randomness, a property which today's RFID tags are ill equipped to meet. The HB+ protocol requires at least 80 rounds [10], in each of which the RFID tag is expected to generate a 224 bit random value, in order to attain an 80-bit security level. If these rounds are run in parallel [18], a RFID tag will be required to produce 17,920 random bits at once.

Since its inception, various variants of HB+ have been proposed including HB++ [7], Trusted-HB [6], PUF-HB [12], and HB# [10]. Protocols derived from HB are not the only RFID security mechanisms that require randomness to operate, though. Other protocols that are based on pseudorandom functions will also require cryptographic random numbers to be generated. For example, the tree based privacy preserving authentication protocols of Molnar and Wagner [20] use pseudorandom functions that require high quality randomness at each tree level.

3 Experiments

In this section, the experiments used to measure the amount of randomness that can be derived from uninitialized RAM, as well as the rate at which this process can be performed, are presented based on the practical limitations of RFID tags and their usage model.

3.1 Experimental Setup

The following configuration of equipment was utilized for these tests. Four WISP tags of the latest hardware version, 4.1, were employed. The WISPs are depicted in Figure 1, with a U.S. quarter placed nearby to provide a sense of scale. When these tags were required to interact with the RFID reader they were loaded with the 6.0 revision of WISP firmware. At times when a tag's memory contents were of interest, tags were loaded with a C file containing nothing but a blank main function; this was done in order to minimize the amount of RAM overwritten during program execution.

To program these WISP tags a Texas Instruments MSP-FET430UIF debugging interface [4] was used, which was interacted with through a desktop computer running the IAR Embedded Workbench IDE [2]. The debugger was connected to the desktop machine with a USB cable and to the WISP tag over a JTAG interface. An EPC compliant Impinj UHF Generation 2 Speedway RFID reader [3] running firmware version 3.2.1 was used. Commands were issued to the reader from a desktop machine through a custom application which communicated with the reader over the Low Level Reader Protocol (LLRP). A block diagram of this hardware configuration is shown in Figure 2.

3.2 WISP Implementation

As a first step towards assessing the viability of deriving randomness from RAM on a passive RFID device, the random number generation portion of FERNS was implemented on a WISP tag. While RAM based randomness has been implemented on a desktop computer, to the authors' knowledge this is the first such implementation on a passively-powered RFID device [13]. The implementation was done in C using the IAR Embedded Workbench IDE. This was a relatively straightforward implementation of the PH hash function (described in Section 2.3). The main change that had to be made was to alter the input block size of the PH hash function from 64 bits to 16 bits, which reduced the size of the hash function's output from 133 bits to 37 bits. This was done in order for the hash output to

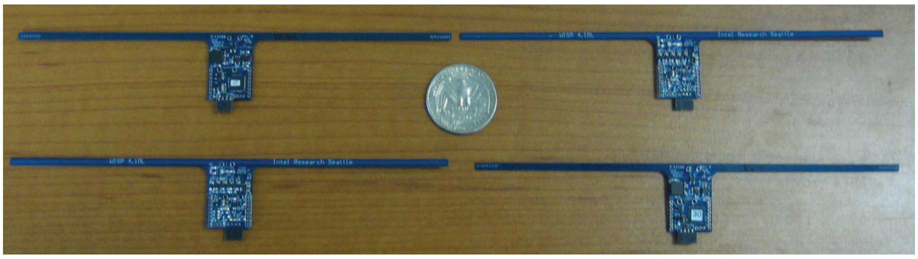

Fig. 1 Four WISP 4.1 Tags with a U.S. Quarter Included for Scale

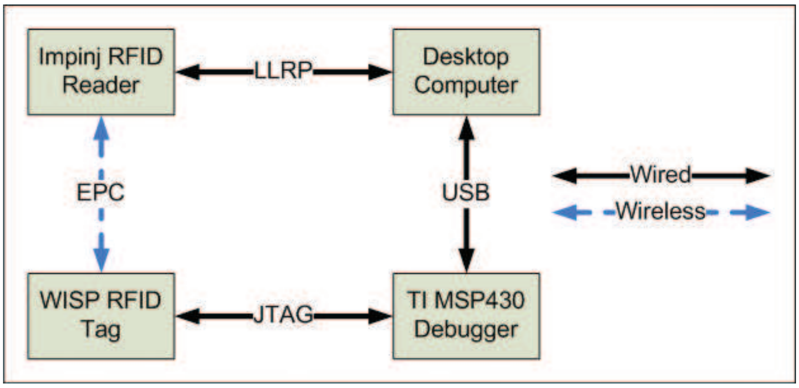

Fig. 2 A Block Diagram of the Experimental Setup

fit into the 64-bit long long data type of the C programming language. Had the original 133 bit output size been used, it would have necessitated the use of a multiple precision arithmetic library, which would require the dedication of the tag's scarce computational and memory resources.

Besides this practical consideration, reducing the block size of the PH function has theoretical advantages as well. When this function's block size is set to 64 bits, $64*32 = 2048$ bits of memory, half having been input as message blocks and the other half having been treated as key blocks, are consumed to produce a single 133 bit hash. When 16 bit blocks are used instead, only $16*32 = 512$ bits of RAM are needed to produce a 37 bit hash. While this decreases the amount of random bits output from one call to the hash function, the smaller block size allows PH to be called 3 more times on distinct blocks of the same amount of uninitialized RAM, yielding $37*4 = 148$ bits of randomness.

Thus, reducing the block size of the hash function allows more bits to be condensed from an equivalent portion of memory. This would be problematic if the larger bit amount exceeded the expected entropy of the values being hashed. Fortunately, this is not the case. Each bit of tag memory is capable of yielding 0.103 bits of entropy [14,15]. This means that the 148 bits output by multiple calls to PH with the smaller hash function is still well within the 210 bits of entropy that 2048 bits of raw memory would be expected to produce.

As a preliminary test of the random values generated by this "on tag" random number generator, 32 blocks of 16 bits each were read from an uninitialized area of the WISP's RAM. These values were hashed and written to a different memory address. The WISP was programmed to perform this operation once per query from the reader. The resultant 37 bit hash value was copied from memory into the WISP's EPC ID, which was then transmitted to the RFID reader in response to its queries. Surprisingly, the transmission of identical values was observed, indicating a clear lack of randomness. Since this random number generation technique is already known to work on traditional machines [14, 15], an investigation was launched into the source of the discrepancy found on this WISP implementation.

3.3 Measuring Data Remanence on WISPs

The WISP tag's programming was altered to transmit the contents of its memory to the reader. This was accomplished by programming the tag to break its RAM into blocks. These blocks were then transmitted through the tag's EPC ID in the same manner as was done with the hash values. While there were occasional changes in certain bytes, the contents of the memory seemed largely unchanged. This was being caused by the WISP tag's retention of values between queries. Recall that passive RFID tags derive power from reader queries. Thus, while continuously being polled for hash or memory values, the WISP tag was receiving a continuous supply of power, causing it to retain its RAM state rather than reinitializing its memory after each query.

A more thorough experiment to analyze the timing of data retention on the WISP's memory was arranged. The methodology of this experiment was similar to that employed in [11]. First, the WISP was connected to a desktop machine using the MSP-FET430UIF debugging interface. All 512 bytes of the WISP's memory were filled with a pseudorandom pattern generated on a desktop machine using the Mersenne Twister [19] implementation included in the random module of the Python programming language's standard library. This pattern was copied to the WISP's RAM through the Embedded Workbench IDE. The WISP tag was then disconnected from the debug interface, depriving it of power for a certain interval of time.

After this, the tag was reattached to the debugger. Rather than using the reader to supply power to the tag and reading the memory values through the tag's EPC ID, which is slow and prone to occasionally missing values, the tag's supply of power was resumed over the debugger. The contents of the WISP's memory were then read back. In order to calculate the tag's decay rate, the Hamming distance between the original pseudorandom pattern and the value read back from the RAM was computed. Two of the 512 bytes of RAM were always overwritten by the debugger, so these bytes were left out of the analysis. The fact that the original pattern was pseudorandom meant that it should contain an approximately equal amount of each bit. Therefore, RAM was considered to be fully decayed once the Hamming distance between the two strings was at or near 50%.

Samples were taken after removing the WISP from power for a duration of 0 to 60 seconds at 5 second intervals. This test was performed on a population of 4 WISP tags. The results of these tests are shown in Figure 3. This graph shows the per-tag decay rate as well as the average decay rate across all tags. A logistic curve has been fit to each data set. While there were some minor variations between samples, the decay rate observed on each tag was generally well matched to this curve, showing an initial 15 seconds with little ($<$ 1%) or no decay, then 15 seconds of very rapid decay, and concluding with the slow decay of whatever data remained. From this data it is clear that, depending on the particular tag, a powerless period of 25 to 30 seconds is required to allow the WISP's 512 bytes of memory to decay completely.

3.4 Temperature and WISP Data Remanence

As discussed in Section 2.4, one of the variables that plays a role in determining the life span of data stored in powerless memory is temperature. Cooling RAM tends to elongate its data retention period, while unpowered memory in a higher temperature environment loses its contents more rapidly. In order to quantify the effect heat has on data remanence in an RFID system, the experiments described in Section 3.3 were repeated. This time, however, additional steps were taken to control and monitor the temperature of the WISP tag while measurements were being taken.

First, a baseline room temperature test was performed. Here the temperature of the tag was left unaltered, but it was recorded using a digital thermometer. Next, a "cold" test was performed where heat was removed from the WISP tag. This was accomplished by filling a plastic freezer bag with ice and placing both the WISP and thermometer probe in contact with it. A visual depiction of this setup is provided in Figure 4. Finally, a "hot" test was conducted by using an electric space heater. The temperature of the tag was increased by placing it in front of the heater along with the thermometer sensor, as shown in Figure 5. In the case of both extreme temperature tests, care had to be taken to ensure a relatively stable temperature for the duration the experiment. For the cold test, this meant periodically replacing the ice as it melted to ensure the temperature did not rise too drastically. With the heat tests, on the other hand, a distance from the heater which sufficiently raised the tag's temperature while not exceeding the maximum temperature limit of the thermometer had to be determined.

During the cold experiments, the tag temperature varied between 2.0 °C and 2.8 °C. The WISP was consistently between 30.6 °C and 31.0 °C while room temperature measurements were being taken. For the heat tests, the tag went as low as 64.2 °C and as high as 69 °C. The results of these tests are provided in Figure 6. There was little difference between the decay rate observed at room temperature and when the tag was cooled with ice. In both cases, little decay occurred prior to the 10 second mark, followed by a slight increase between 10 and 20 seconds. After 20 seconds had passed, full data degradation had already set in. Contrary to this result, other publications have established that reducing the temperature of memory hardware tends to prolong or even suspend the process of data decay [11]. However, this effect was observed at a much lower temperature, approximately -50 °C, which was achieved using chemical coolant agents. A dramatic increase in the rate of decay was present when the tag

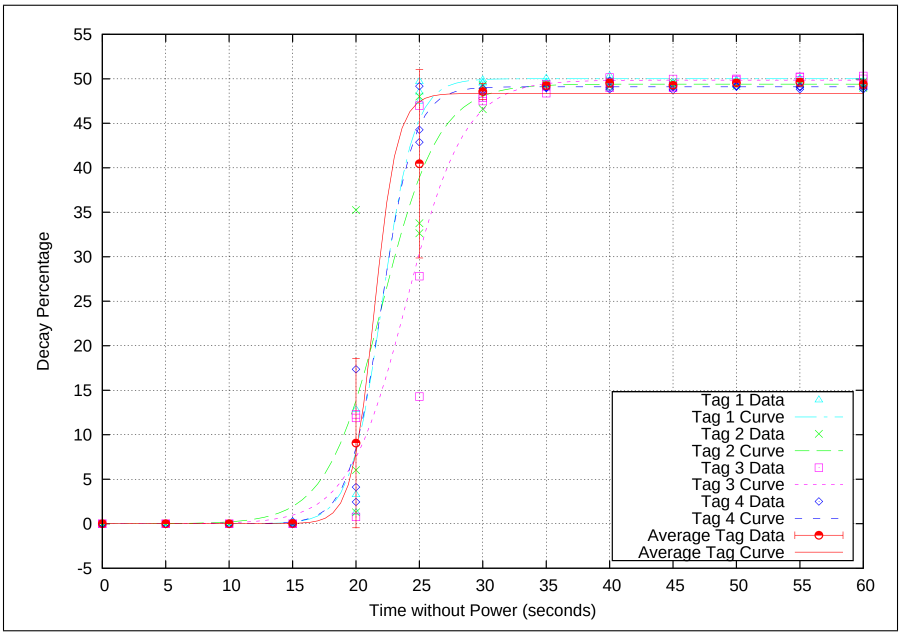

Fig. 3 Decay Rate for Each Tag

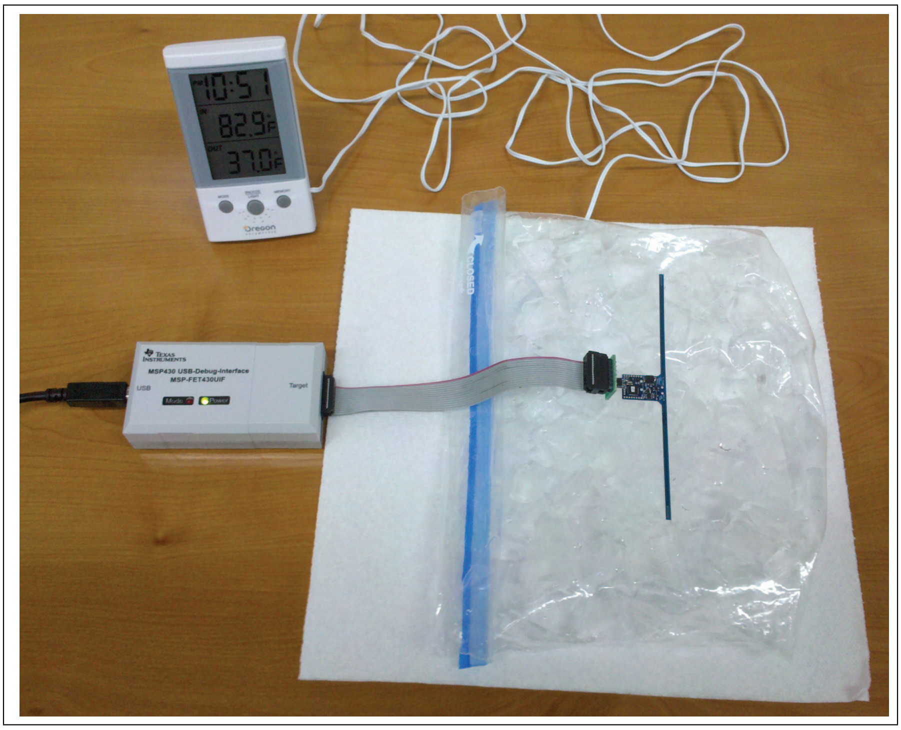

Fig. 4 WISP tag being cooled on a bag of ice while attached to MSP-FET430UIF debugging interface

was heated to approximately 65.6 °C, though. Rather than take 20 seconds to achieve the complete decay of the tag's memory contents, this occurred in only 10 seconds in the presence of this increased amount of heat. Thus, in the presence of temperature variations the WISP tag's onboard RAM exhibits data remanence behavior that largely conforms with observations of memory in other systems.

3.5 Available Memory

Having established how long it takes for a WISP's memory to return to an uninitialized state, the next open

Fig. 5 WISP tag being warmed by an electric heater while attached to MSP-FET430UIF debugging interface

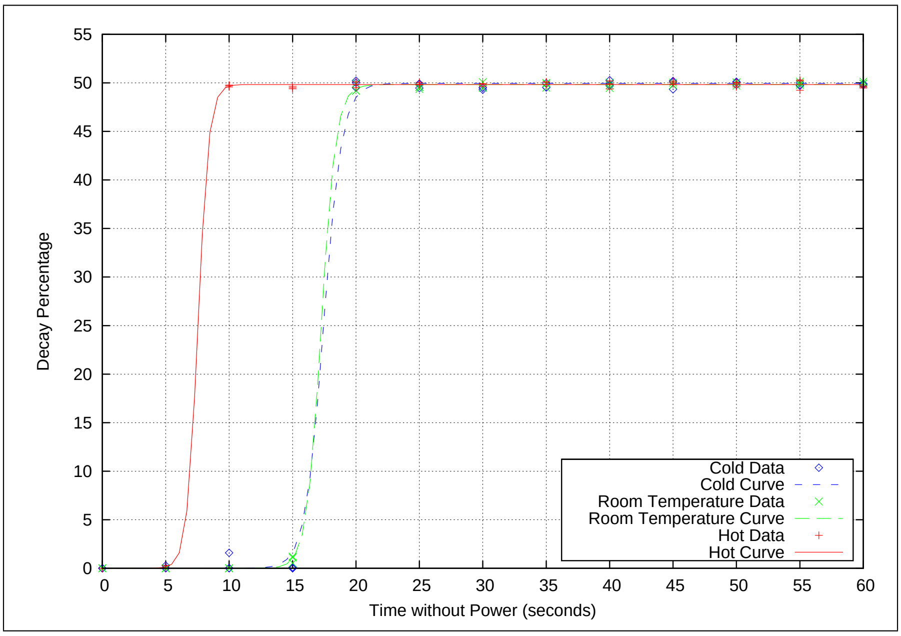

Fig. 6 Effect of Temperature on Decay Rate

question was how much uninitialized memory is available on a WISP at any given time. To determine the amount of unused RAM on the latest version (4.1) of WISP tags used in these experiments, the tags were first loaded with their default firmware. Next, the largest data structure allowed by the compiler was stored in the tag's RAM. It was observed that the WISP protocol occupied 136 bytes of this tag version's memory, leaving 376 bytes free for use as an entropy source. Note, however, that by default this firmware does not implement all mandatory aspects of the EPC standard. Enabling other features of the protocol in the WISP firmware, such as the ability to use multiple readers or read multiple tags simultaneously, takes up an additional 12 bytes of RAM, leaving 364 bytes available for random number generation.

For earlier versions (2.0 and 2.1) of WISP tags, [8] established that at any given time 112 bytes of WISP memory are occupied by the RFID protocol and stack. Assuming no additional memory is used in order to program the tag with increased functionality, this leaves a maximum of 144 uninitialized bytes for random number generation. This assumes that no other RAM space is occupied by the authentication protocol itself, which is unlikely to be true in practice.

4 Discussion

4.1 Practicality: Effects of Data Remanence and Available Memory

Taking the HB+ and HB# protocols as motivating examples, how feasible is the use of RAM based random number generation for RFID applications in need of random numbers? To provide 80 bit security, the HB+ protocol requires at least 80 rounds [10], in each of which the RFID tag is expected to generate a 224 bit random value. If these rounds are run in parallel, the WISP must be capable of generating 17,920 random bits at a time. The randomness requirements of the HB# protocol are more modest, requiring a single round where a 512 bit random value is generated by the tag, though this is at the cost of a higher memory overhead.

In the FERNS approach, as reported in [14,15], an entropy rate of 0.103 bits of entropy per bit of uninitialized memory was observed. Combining this with the maximum of 376 bytes of unused RAM on a 4.1 WISP tag yields an expected random number capacity of 309 bits. A 4.1 WISP tag would therefore require its available memory to be hashed 58 times in order to meet the randomness requirements of the HB+ protocol and 2 times in order to generate enough randomness for the HB# protocol. Since a “cool down” interval of about 30 seconds is required between memory hashes in order to allow a WISP tag’s RAM to return to its uninitialized state, this implies that 30 seconds of wait time would be required for this type of tag to generate enough randomness for a single HB# session and 28.5 minutes of wait time would be necessary to create enough random bits for one HB+ protocol instance.

Of course, these estimates only apply to the latest iteration of WISP hardware. RFID tags with lower capabilities would require even more time. On the earlier 2.0 or 2.1 versions of WISP tags, which featured 256 bytes of RAM in total, out of which 144 bytes are available for hashing, 118 random bits could be expected to be generated from each memory hash. This would necessitate 152 hashes for HB+ and 5 hashes for HB#, yielding uninitialization wait times of 76 minutes and 2.5 minutes for each respective protocol. These figures are specific to the specialized hardware found on WISP tags, which for the purposes of allowing programming flexibility have memory capacities well beyond those of commercial RFID tags. A typical fixed function RFID tag that costs 5 to 10 cents is expected to have a maximum of only 128 bits of RAM in total [17], making the prospect of deriving sufficient randomness from this source even dimmer.

The focus of the FERNS technique [14] is on True Random Number Generation and not on Pseudorandom Number Generation. It may be possible to use the random data derived from RAM as a seed for a Pseudorandom Number Generator (PRNG) that could generate a sequence of seemingly random bits that is much longer than the seed itself. There are several potential pitfalls in this approach, however. For one, a great deal of scrutiny is required to ensure that a PRNG algorithm generates randomness of sufficient quality for cryptographic use. Flaws in prominent PRNG implementations have gone unnoticed for months or even years [26], compromising the security of the systems on which they are based for that duration. Furthermore, physical attacks on an RFID tag could enable an adversary to compromise the randomness generated by such a procedure by deriving the PRNGs state. Finally, due to their hardware limitations, many RFID tags lack the resources necessary to support a PRNG. Those that do not have such hardware limitations may need to use these precious constrained resources for other computational tasks.

4.2 Effect on Usage Model

The issue of RAM data retention is complicated by the usage model expectations of an RFID system. For instance, consider the case of contactless access card usage. Since cost efficient tags are passively-powered, they power up when they come into range of a compatible RFID reader and do not power down until they leave the reader’s field of view. This would mean that a standard RFID enabled access card would have to be taken outside of the range of a reader in order to allow its memory to “cool down” and return to an uninitialized state in order to perform random number generation. Thus requiring multiple consecutive RAM hashes would significantly alter the RFID usage model. Instead of a user presenting his or her tag to a reader once, leaving it present momentarily, and returning the tag to a pocket or other storage, users would have to repeatedly bring the access card within the range of the reader and back out again, introducing a high user burden into the authentication process.

Further complicating the situation is the need for the user to determine when to remove the card from reader’s range and for how long. It is suspected that specialized hardware could be added to a RFID tag to address this problem by cutting power to memory after a random number generation was requested. This would add complexity and thus cost to the tag, however, contrary to their intended economic efficiency. Furthermore, a hardware based solution would also not address the underlying need to wait for several seconds between two consecutive RAM reads.

4.3 Potential Attacks

The need to move a tag outside of the range of a reader for a fresh random number generation also introduces the potential for new attacks. If an adversary were able to continuously supply power to a tag which made use of its RAM for randomness purposes, he or she would essentially force the tag to continuously reuse the same RAM values for hashing. This would make the values generated extremely predictable, undermining the security of any authentication scheme or cryptographic protocol built on top of the random number generator.

As mentioned above, hardware could be added to lock down a tag's memory until it has time to return to a decayed state. However, this would create the potential for a denial of service (DoS) attack where an attacker continuously powered a tag, preventing it from generating any random numbers and thus from being used at all. While DoS attacks on RFID systems are always possible by simply jamming the radio signals involved, this type of attack is worse in the sense that it does not involve any jamming in the traditional sense. All an attacker would need to do is repeatedly issue queries to the tag, rather than continuously jam an entire portion of the radio spectrum.

4.4 Fingerprint Extraction

In the FERNS technique, identifying a RFID tag through its memory is accomplished by locating bits that initialize to a consistent value. Because this process is not dependent on the availability of fresh mutable bits each time it is executed, RAM based fingerprint derivation is not as adversely impacted by the factors discussed above as the generation of random numbers. Nonetheless, there are still practical consideration that must be taken into account when performing memory fingerprinting on RFID tags. For one, the number of unique fingerprints that can be generated is clearly dependant on the size of a tag's RAM. The less memory that is available, the fewer possible fingerprints that may potentially be generated. This might result in problematic collisions for tags with only a few bits of memory.

Furthermore, while memory does not need to be recently uninitialized in order to be used for identification purposes, its original values still must not be overwritten prior to fingerprinting. Thus, if a segment of a tag's RAM has been filled, the RFID device must wait for it to decay fully before accessing its raw fingerprint. Alternatively, the RFID tag could work with the remainder of memory that is not in use, but this would result in a reduction of the size of the fingerprint, which may be problematically small to begin with. In summation, fingerprint extraction is also in contention with other RFID tag system operations for shared memory resources.

4.5 Impact of Temperature

Like memory found in other systems, the deterioration rate of information stored in an RFID tag's RAM changes when subjected to extreme temperature conditions. Tags that are expected to operate primarily in such environments will therefore experience more or less pronounced remanence effects. For example, a passive RFID sensor affixed to a heating furnace or vehicle engine would experience less of a remanence effect and thus a faster decay rate than one operating at room temperature. As a consequence, FERNS random number generation and identification is better suited for use on tags designed for such circumstances.

On the other hand, tags fashioned for use in ultra low temperature scenarios, such as those used to monitor manufacturing processes requiring coolants or frigid outdoor areas would experience prolonged or nonexistent memory data decay. FERNS is not a viable option on such devices for this reason. Additionally, any adversary with the capacity to remove a great deal of heat from an RFID tag could exploit this behavior by preventing the decay of stored data, causing the generated randomness to be based on stale, reused memory values. This would necessitate physical access to the tag, however.

4.6 Other Issues

In addition to data remanence, SRAM is subject to a less well known effect called "burn-in." If a memory cell is used to store the same value over an extended time interval, the value will "burn in" to that cell. That is, the cell will always enter the state which represents the stored value as soon as it is supplied with power. Like data remanence, "burn-in" has serious implications for cryptographic operations. If a key is stored in the same memory location for a long period of time, it may be possible to recover it even after power has been removed from the SRAM circuitry.

"Burn-in" is also problematic for memory based random number generation. An SRAM cell that becomes "burnt in" will be predictable and will not contribute any entropy even if the cell previously exhibited unpredictable behavior. On the other hand, "burnt in" cells are useful for fingerprinting operations because they reliably enter the same state upon the receipt of power. Since there is not a similar process by which predictable SRAM cells become unpredictable, this type of memory can be expected to lose random cells and gain deterministic cells over time.

We performed all of our experiments using the WISP's MSP430F2132 microcontroller. We therefore cannot claim that all types of SRAM will act in the same fashion. Some exotic forms of SRAM may be immune to the sources of physical noise that cause unpredictability in most SRAM cells. Clearly, any memory circuitry that is wholly deterministic cannot be used to generate random values but would be very applicable for fingerprinting purposes.

5 Conclusion

To conclude, several practical shortcomings of using general purpose memory as a source of randomness for low cost RFID devices have been presented. Since RAM is already in short supply on such resource constrained devices, much of it will likely be in use and thus unavailable as a source of randomness. Due to the phenomenon of data remanence, a longer than expected wait time is required between consecutive uses of RAM as an entropy source, making its repeated utilization impractical in the RFID usage model.

It should not be concluded, however, that RAM based randomness derivation should be discarded. This innovative technique remains attractive due to its repurposing of existing hardware, which is important for minimizing the costs of tag production. On its own, however, this method seems unlikely to be able to handle the randomness requirements of current RFID authentication protocols such as HB+, HB#, and related variants.

In practice, many services derive random numbers from environmental noise. Related work includes an investigation into the viability of alternative sources of randomness, such as onboard sensors, to collect ambient noise of different forms [22]. This approach is not subject to the time and space constraints faced when harvesting entropy from memory. As sensing platforms, WISP tags are well suited to exploring this area. For example, the current 4.1 iteration of WISP hardware features an onboard accelerometer, temperature sensor, voltage sensor, and capacitance sensor. Additionally, it is possible to add new sensors by wiring them to a WISP. Ways in which entropy sources such as these can be aggregated to efficiently produce the amount of randomness necessary to support various cryptographic protocols aimed at low cost tags will be subject to further analysis in the future.

Acknowledgements

The authors would like to thank Dan Holcomb, David Molnar, and the anonymous RFIDSec'09 reviewers for their helpful feedback on an earlier version of this paper. They would also like to give a special thanks to Dan Yeager both for his comments on this paper and his advice on WISP programming.

This work was partially supported by the United States Department of Education GAANN grant P200A090157.

References

1. EPC Global Class 1 Generation 2 UHF Air Interface Protocol Standard "Gen 2". Available at `http://www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_2_0-standard-20080511.pdf`, 2008.
2. IAR Embedded Workbench. Available at `http://www.iar.com/ew`, 2009.
3. Iminj's UHF Gen 2 RFID Speedway Reader. Available at `http://www.impinj.com/products/rfid-reader.aspx`, 2009.
4. MSP430 USB Debugging Interface - MSP-FET430UIF. Available at `http://focus.ti.com/docs/toolsw/folders/print/msp-fet430uif.html`, 2009.
5. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and Secure Message Authentication. In *CRYPTO*, 1999.
6. J. Bringer and H. Chabanne. Trusted-HB: A Low-Cost Version of HB+ Secure Against Man-in-the-Middle Attacks. In *IEEE Transactions on Information Theory*, 2008.
7. J. Bringer, H. Chabanne, and E. Dottax. HB++: a Lightweight Authentication Protocol Secure against Some Attacks. In *Security, Privacy and Trust in Pervasive and Ubiquitous Computing*, 2006.
8. A. Czeskis, K. Koscher, J. Smith, and T. Kohno. RFIDs and Secret Handshakes: Defending Against Ghost-and-Leech Attacks and Unauthorized Reads with Context-Aware Communications. In *ACM Conference on Computer and Communications Security*, 2008.
9. P. Gutmann. Data Remanence in Semiconductor Devices. In *USENIX Security Symposium*, 2001.
10. Y. S. H. Gilbert, M. Robshaw. HB#: Increasing the Security and Efficiency of HB+. In *EuroCrypt*, 2008.
11. J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Apelbaum, and E. Felten. Least We Remember: Cold Boot Attacks on Encryption Keys. In *USENIX Security Symposium*, 2008.
12. G. Hammouri and B. Sunar. PUF-HB: A Tamper-Resilient HB Based Authentication Protocol. In *ACNS*, 2008.
13. D. Holcomb. Personal Communication. April 2009.
14. D. Holcomb, W. Burleson, and K. Fu. Initial SRAM State as a Fingerprint and Source of True Random Numbers for RFID Tags. In *Conference on RFID Security*, 2007.

15. D. E. Holcomb, W. P. Burleson, and K. Fu. Power-up SRAM State as an Identifying Fingerprint and Source of True Random Numbers. *IEEE Transactions on Computers*, 2009. to appear.
16. N. Hopper and M. Blum. Secure Human Identification Protocols. In *Asiacrypt*, 2001.
17. A. Juels and S. Weis. Authenticating Pervasive Devices with Human Protocols. In *CRYPTO*, 2005.
18. J. Katz and J. Shin. Parallel and Concurrent Security of the HB and HB+ Protocols. In *EUROCRYPT*, 2006.
19. M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudorandom Number Generator. In *ACM Transactions on Modeling and Computer Simulation*, 1998.
20. D. Molnar and D. Wagner. Privacy and Security in Library RFID: Issues, Practices, and Architectures. In *ACM Computer and Communications Security*, 2004.
21. A. Sample, D. Yeager, P. Powledge, and J. Smith. Design of a Passively-Powered, Programmable Sensing Platform for UHF RFID Systems. In *IEEE International Conference on RFID*, 2007.
22. N. Saxena and J. Voris. Accelerometer Based Random Number Generation on RFID Tags. In *WISP Summit*, 2009.
23. N. Saxena and J. Voris. We Can Remember It for You Wholesale: Implications of Data Remanence on the Use of RAM for True Random Number Generation on RFID Tags. In *Workshop on RFID Security*, 2009.
24. S. Skorobogatov. Low Temperature Data Remanence in Static RAM. Available at `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.html`, 2002.
25. J. Smith, A. Sample, P. Powledge, A. Mamishev, and S. Roy. A Wirelessly-Powered Platform for Sensing and Computation. In *8th International Conference on Ubiquitous Computing*, 2006.
26. F. Weimer. New openssl packages fix predictable random number generator. Available at `http://lists.debian.org/debian-security-announce/2008/msg00152.html`, 2008.
27. K. Yuksel, J. Kaps, and B. Sunar. Universal Hash Functions for Emerging Ultra-Low-Power Networks. In *Communications Networks and Distributed Systems Modeling and Simulation Conference*, 2004.