EVALUATING A NEW MAC FOR CURRENT AND NEXT GENERATION RFID

A Thesis Presented

by

SERGE ZHILYAEV

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

February 2010

Department of Electrical and Computer Engineering

EVALUATING A NEW MAC FOR CURRENT AND NEXT GENERATION RFID

A Thesis Presented
by
SERGE ZHILYAEV

Approved as to style and content by:

____
Wayne P. Burleson, Chair

____
Kevin Fu, Member

____
Andras C. Moritz, Member

____
C.V. Hollot, Department Head
Department of Electrical and Computer Engineering

To my parents.

ACKNOWLEDGMENTS

First and foremost, I would like to thank Professor Wayne Burleson for providing me with an opportunity to experience this journey and for all of his advice and support. I would also like to thank Professors Wayne Burleson, Kevin Fu, and Andras Moritz for their service on my committee. Thanks to Professor Christof Paar for his invaluable advice and words of encouragement. I would like to thank my family and friends for their support throughout my academic career. I thank the entire VLSI Circuits and Systems group as well as the RFID-CUSP group for their advice and assistance. Special thanks to Jinwook Jang, Basab Datta, and Mike Todd.

ABSTRACT

EVALUATING A NEW MAC FOR CURRENT AND NEXT GENERATION RFID

FEBRUARY 2010

SERGE ZHILYAEV

B.S., UNIVERSITY OF MASSACHUSETTS AMHERST

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Wayne Burleson

Pervasive computing is an emerging technology in which inexpensive devices are used in a variety of applications which include authentication, identification, and micro payments. These systems need protection from malicious users but often are too constrained for traditional security algorithms. As a result, new algorithms dubbed lightweight have been proposed. We evaluate one such algorithm called SQUASH [1] and propose changes that reduce both area and latency. Our proposed changes take advantage of the underlying squaring operation in SQUASH. We also show that aggressive over minimization of the adder in [1] results in the addition of an auxiliary register that is used to store state: the cost of which is much greater than a more complex adder. By using a different adder architecture and taking advantage of the associative property of addition, we can eliminate the auxiliary register and decrease latency; resulting in an area reduction of 17 % and a latency reduction of 50 %.

To reduce area and latency further, we propose a new variant which we call permuted

SQUASH (PSQUASH). The new variant eliminates the duplicate NLFSR found in SQUASH by using a static permutation to reorder the NLFSR output before the squaring operation. PSQUASH retains the formal proof of security of SQUASH, but is significantly smaller. The smallest configuration of PSQUASH is synthesized in 1624 GE (gate equivalents). While our proposed changes to the hardware implementation of PSQUASH offer a sizable improvement over published work in both area and latency, the design may still be prohibitive for some applications mainly due to relatively high latency. The small size of PSQUASH still makes the hardware approach attractive when latency is not an issue; for example, PSQUASH can be a good fit for access control or authentic replacement parts.

Adi Shamir proposes that SQUASH may be best suited for next generation RFID tags because it can be scaled to arbitrary precision. We explore SQUASH on an embedded platform and show that the benefit of a wider ALU is negated by the inefficiency of NLFSR shifts in software. We provide a method to select a partial product ordering to minimize NLFSR shifts for any SQUASH configuration.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

1.1 Motivation

Radio frequency identification (RFID) tags are highly constrained devices capable of limited computation. As the name suggests, a common purpose of these devices is to provide identification. For example, Electronic Product Code (EPC) [2] Ultra High Frequency (UHF) RFID tags may replace the barcode or Universal Product Code (UPC). While the familiar barcode is still on consumer products, RFID integration into retail stores is evident with Wal-Mart's adoption of RFID systems for inventory tracking [3]. RFID tags are often embedded in credit cards and other payment methods, like micro payment cards. They are used in access cards, passports, and can be implanted into animals for tracking. RFID tags for medical supplies can save lives by reducing human error in hospitals, and RFID tags can be an effective tool in the fight against counterfeiting. Many exciting applications have been proposed for these devices and this technology continues to expand its impact on our lives.

Unit cost per tag is a major consideration for RFID tags because some applications need low cost tags to be feasible. Cost may be a secondary consideration in passports or credit cards because security is paramount and these devices may pass that cost on to the consumer without much concern. In an application like product tagging, cost is paramount, and the cost per tag needs to be low; otherwise, the benefits of RFID are outweighed by the cost. Securing RFID tags and providing privacy in consumer applications, while limiting cost per tag, has been the focus of much academic work. Due to the constraints on memory, power consumption, and amount of logic on RFID devices, standard cryptographic primitives are often unsuitable. This has led to a new class of cryptographic primitives and protocols that have been dubbed *lightweight crypto*, which are designed with resource constraints in mind.

Some of the first lightweight crypto work reduced the size and power consumption of standard algorithms like AES or DES by serialization or other techniques [4, 5]. Works like PRESENT [6] and the steam cipher competition eSTREAM [7] took lightweight cryptography further by designing new ciphers for resource constrained devices. Lightweight message authentication codes (MAC), or authentication schemes in general, have not received as much attention as symmetric ciphers; although, using symmetric ciphers for authentication has been proposed [4]. The HB protocol and its variants [8, 9, 10, 11], which are based on the *solving parity with noise problem*, are a lightweight authentication scheme but have several drawbacks [1]. A recent proposal called SQUASH [1] is a simple challenge response MAC for RFID. SQUASH focuses on providing authentication and does not preserve privacy; however, it does not require a source of randomness on the tag or a heavy computationally load on the backend.

1.2 SQUASH for RFID

SQUASH has several advantages and one big disadvantage over other authentication methods for RFID. Many lightweight authentication methods that have been proposed are only resistant against a passive adversary. Active attacks against the HB schemes and [12, 13] have exposed vulnerabilities. Many of the proposed methods shift computational workload to a backend server, which can expose them to a man in the middle attack [14]. Many RFID authentication schemes are tailored to the supply chain applications which can expect to have a centralized system with backend server support; however, the backend server requirement itself can be a disadvantage because it may not be practical or even available for all applications. While SQUASH requires a shared secret between the reader and key, it does not require a backend server to perform calculations or exhaustive searches.

SQUASH does not require the tag to generate any random bits. RFID tags generally do not have a true random number generator (TRNG) and many authentication protocols require a large number of high quality random bits. Random number generators on RFID tags are usually weak pseudorandom stream generators whose purpose is for collision avoidance in the communication protocol. This means that if an authentication scheme needs TRNG, it must be added to the tag.

SQUASH does have two major drawbacks which are higher area and potentially higher latency. Without going into details, the SQUASH architecture is much more complex than other lightweight RFID authentication schemes. Other authentication schemes shift the computational load onto a backend server, which minimizes the resources required by the tag. SQUASH can be deployed without backend server support but this requires greater tag resources. Unlike probabilistic authentication schemes such as HB, which require multiple queries to the tag, SQUASH only requires a single query; however the response takes thousands of cycles for the SQUASH designs explored in [15], which is a major drawback. This thesis evaluates and improves SQUASH for current and next generation RFID by reducing area and latency.

1.3 RFID Tags and Related Resource Constrained Devices

In this thesis we use RFID and *resource constrained device* interchangeably. This work can apply to any resource constrained device but sometimes we focus on RFID. We believe these devices are especially positioned to become the dominant resource constrained device deployed in the future. Some general classes of RFID tags and other resource constrained devices are briefly described in this section.

Low Frequency (LF) tags operate in the 125–134.2 kHz and 140–148.5 kHz range and work by inductive coupling.

High Frequency (HF) tags (13.56 MHz) also use inductive coupling.

Ultra-High Frequency (UHF) tags operate in the 868–928 MHz range; and unlike LF or HF tags, UHF tags use backscatter to communicate with the reader. They also have increased read range but can be unreliable in certain environments because the signal may be reflected by metal objects or absorbed by water.

Microwave tags further increase read range; however these tags are almost exclusively battery assisted. Automobile toll collection is a common application for these tags.

Passive tags do not have a battery; instead they harvest power from the reader's radio signal.

Active tags have a battery; which increases the read range and capability of these devices over their passive counterparts, but drives up costs.

Contactless Smartcards do not use a wireless channel for communication. Physical contact is required to communicate with the device.

1.4 Thesis Outline

A SQUASH implementation on FPGA was published in [15]; which concluded that while SQUASH consumed a large area, an ASIC implementation may be more competitive. In this thesis we synthesize SQUASH as an ASIC implementation. We find that in addition to its inherent high latency, SQUASH consumes more area than other lightweight primitives. In chapter two we introduce SQUASH and show that it can be improved by optimizing for squaring. In chapter three we propose a new variant we call PSQUASH, which achieves further reduction in area and latency. Chapter four explores SQUASH on embedded microprocessors. The final chapter draws conclusions and discusses the practical applications of SQUASH.

CHAPTER 2

A NEW MAC FOR RFID

2.1 Introduction

Some applications require the ability to authenticate a RFID tag. Anti-counterfeiting tag's main task is to authenticate a product; for example, a cell phone replacement battery can benefit from an anti-counterfeiting tag. If the cost of the tag is minimal, a cell phone manufacturer can embed a tag into the battery and be able to detect if a replacement battery is authentic. Since replacement parts are often a significant portion of revenue, the manufacturer can ensure its own battery is used by detecting competitors' batteries and draining them faster than its own product. Anti-counterfeiting for replacement parts or to authenticate brand names are one application, others include access cards, passports, and micro payment cards.

There are a multitude of existing applications, which require the ability to authenticate a tag, and new applications are likely to emerge; however there are few authentication schemes for RFID. A new proposal, SQUASH or "SQUare-hASH" by Adi Shamir [1] is a challenge response MAC for RFID. SQUASH does not require a source of randomness on the tag and is scalable to arbitrary word size. Unlike other authentication schemes SQUASH has a formal proof of security [1].

2.2 A New MAC for RFID

SQUASH [1] is a challenge-response MAC proposed by Adi Shamir. The idea behind SQUASH is to create a hash function that is not necessarily collision resistant, because collision resistance is costly. This makes SQUASH unsuitable for digital signatures but it is still an effective challenge-response MAC [1]. SQUASH does not require the tag to produce any random bits unlike the HB authentication scheme or its variants [8, 9, 10, 11], and has a formal proof of security based on Rabin's public key cryptography algorithm [16]. Given a shared secret *S* between the reader and tag, and a random challenge *R* from the reader; the tag returns $[m^2 \bmod n]_{\{i + l \,..\, i\}}$ where *n* is of the form 2^k - 1, *m* is a *k* - 1 bit value generated by a non-linear feedback shift register (NLFSR) seeded with (*R xor S*), *i* is the chosen lsb of the hash output, and *l* is the chosen hash length which can be 32, 64, or 128 bits.

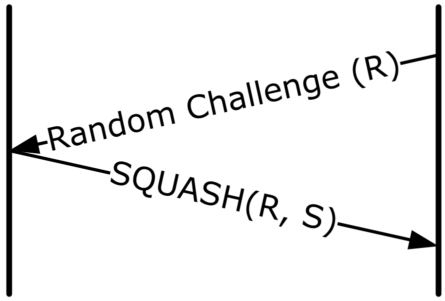


Figure 1: SQUASH is simple challenge-response.

At the heart of SQUASH is the well studied Rabin cryptosystem, which provides the one way property used to hide the secret *S*. This may seem unlikely at first since modular exponentiation is computationally expensive, and a poor choice for the resource constrained RFID. In addition to being computationally expensive, the recommended bit length for the modulus *n* is at least 1024 bits; this number alone is beyond the storage capacity of most RFID tags. To tailor SQUASH to RFID, several key adjustments are made which allow SQUASH to keep the formal proof of security of Rabin's cryptosystem while conforming to the constraints of RFID.

First, do not store the modulus *n*; instead, use a Mersenne number like $2^{1227} – 1$. [1] proposes several suitable moduli. Using a Mersenne number eliminates the need to store *n* and simplifies the modular operation because $2^k = 1 \bmod n$ when $n = 2^k -1$ *e.g.* when $n = 2^k -1$, $m^2 = m_1 * 2^k + m_2$ then $m^2 = m_1 + m_2 \bmod n$. Second, use a subset of $c = m^2 \bmod n$ as hash output from the center of *c*. This can be closely approximated by computing eight to sixteen guard bits [1]. The exact *carry in* would require the entire result of m^2 but [1] argues that approximating the carry with eight to sixteen guard bits is sufficient to be indistinguishable from the exact result. Finally generate *m* on the fly using a NLFSR. Summarizing the key points:

- Do not store modulus *n*, instead use a Mersenne number like $2^{1277} - 1$
- Output a subset of $m^2 \bmod n$ as hash of *m*
- Generate *m* on the fly using a reversible non-linear feedback register

SQUASH can be computed using successive convolutions. To compute thirty-two bit length SQUASH with $n = 2^{1277} - 1$ [1] proposes the following algorithm:

Algorithm 1: SQUASH using successive convolutions [1].

```
a. Set $j = 600$ and set $carry = 0$
b. Compute $carry = carry + m_v * m_{j-v \bmod k}$ for $v = 0,1,...,1276$
c. Set $c_j = \mathrm{lsb}(carry)$, $carry = \mathrm{rightshift}(carry)$
d. Repeat steps $b$ and $c$ 48 times, and output the 32 bits $c_{[647\ ..\ 616]}$
```

2.3 Sequence Generators

SQUASH uses a sequence generator to expand a seed into a larger sequence that is then used as *m* in $c = m^2 \bmod n$. One of the most simple sequence generators is a linear feedback shift register (LFSR) show in figure 2. A LFSR can be constructed using a maximal polynomial which will iterate through all $2^n - 1$ states (where *n* is the width of the LFSR) excluding the zero state.

The design of a LFSR is minimal and only consists of exclusive-or gates at the taps (which are the positions where the feedback polynomial coefficients are non zero.) While a LFSR meets the desired criteria of minimal hardware, it is not a good candidate for SQUASH because the sequence produced is linear. The authors in [17] show that SQUASH can be broken when a linear sequence generator is used; therefore, SQUASH uses a non-linear sequence generator.

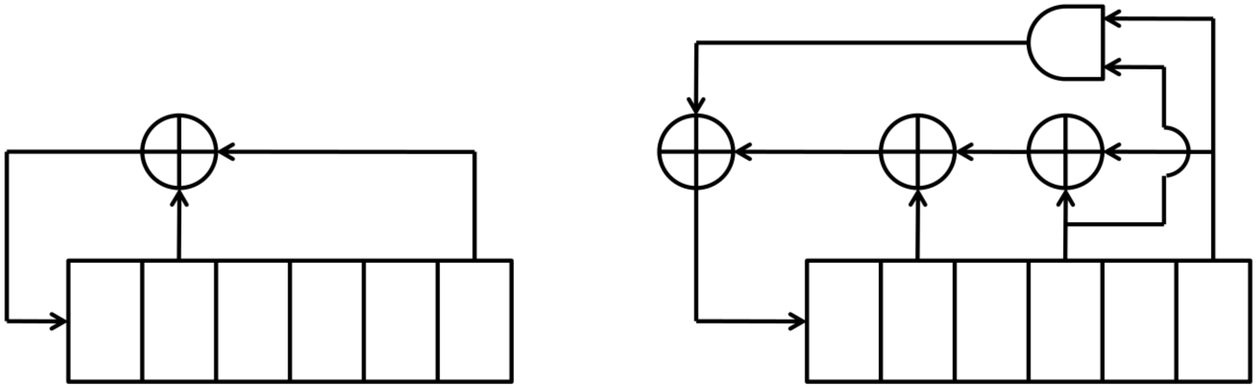


Figure 2: A linear (left) and non-linear (right) feedback shift register.

The suggested NLFSR for SQUASH is the non-linear portion of the Grain steam cipher [18]. The feedback function is a sum modulo two of a linear and a quadratic bent function with the following feedback polynomial *g(x)*:

$$g(x) = 1 + x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{65} + x^{71} + x^{80} + x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71}$$
$$+ x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} + x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43}$$
$$+ x^{47}x^{52}x^{59}x^{65}x^{71} + x^{28}x^{35}x^{43}x^{47}x^{52}x^{59}$$

2.4 SQUASH Architecture Using Successive Convolutions

A hardware SQUASH implementation using Algorithm (1) must address the modular operation on the *position* of the NLFSR. While a single copy of the NLFSR is sufficient to produce the sequence *m*: at least two NLFSR copies are required because the multiplier and multiplicand are – on average – *k*/2 shifts apart and *m* is not stored in memory. Using two copies of NLFSR state addresses the spatial difference between the multiplier and multiplicand, but does not solve the modular position issue. Figure 3 illustrates NLFSR position through several convolutions. NLFSR1 generates the multiplicand value which cycles from [0 .. *k* - 1] inclusive. NLFSR2 generates the multiplier which wraps around from the zero position to the k - 1 position. This wrap around, or modular operation on the NLFSR position, results in cycles dedicated to aligning the NLFSR to the proper position which we call *alignment shifts*.

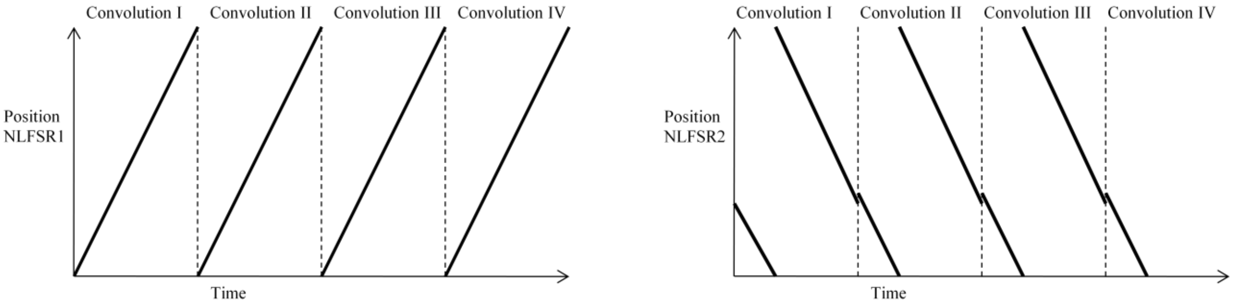


Figure 3: NLFSR position needed to generate *m*.

If only two copies of NLFSR state are maintained, alignment shifts will double latency, because every convolution requiring *k* cycles will also require *k* alignment shifts. The authors in [1, 15] address this issue by using an auxiliary register to store states that correspond to breaks in the otherwise continuous lines in figure 3. This approach adds significant area overhead by adding an eighty bit register, as well as multiplexors and exclusive-or gates to switch values between the auxiliary register and NLFSRs; however this scheme reduces alignment shifts to *k*. The *k* alignment shifts are incurred at the initialization step. After the initialization step, the auxiliary register holds the state needed by the each NLFSR to eliminate additional shifts. Figure 4 shows the proposed design in [1, 15].

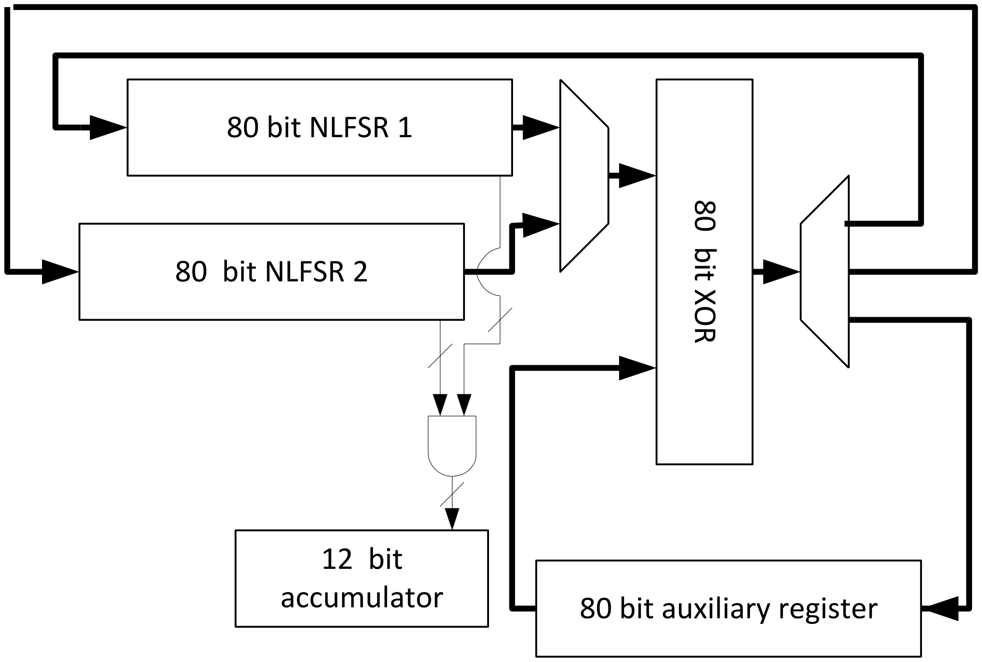


Figure 4: SQUASH architecture proposed in [1].

2.5 Optimizing SQUASH for Squaring

Using successive convolutions to compute SQUASH is essentially column wise or Comba method multiplication [19] with a column width of a single bit. This reduces carry propagation to a minimum and allows a twelve bit accumulator to suffice. Since squaring is the underlying operation, it would be beneficial to optimize for squaring. For each bit computed by Algorithm (1), k partial products are generated; however, only half are unique. The multiplier a and multiplicand b are identical; therefore $a_i * b_j = a_j * b_i$. Reducing partial product generation can be achieved by computing all unique partial products and shifting repeated partial products by one.

Optimizing for squaring has the potential to reduce the number of addition operations by a factor of two; however, it cannot be directly used with the architecture in [1, 15]. After the initialization step requiring k alignment shifts, the NLFSRs combined with the auxiliary register generate the bits needed to compute each partial product in order. These shifts also align the three state registers for the next convolution; therefore, grouping like terms using this architecture would reduce the number of additions per bit but proportionately increase the number of shifts needed to align the NLFSRs.

To optimize SQUASH for squaring recall that $m^2 = m_1 * 2^k + m_2$, then $m^2 = m_1 + m_2$ mod n when $n = 2^k$ -1. When computing SQUASH using successive convolutions, each bit of $m_1 + m_2$ is computed together. This does not allow any optimization for squaring because shifting the NLFSRs to the proper positions when moving between m_1 and m_2 negates the benefit of the optimization. The NLFSRs only need to be aligned when computation switches between m_1 and m_2; as a result, optimizing for squaring provides the most benefit if m_1 or m_2 is computed entirely before computing the other and no benefit if computation alternates between m_1 and m_2 for each bit of m^2.

A more complex adder is required to take advantage of the underlying squaring operation. The adder width can be less than the width of m; the tradeoff is more alignment shifts. We recommend an adder width of thirty-two, which gives up to a twenty bit window before overflow becomes a concern and the adder must be shifted (for single bit precision). For the most significant bits, overflow is not a concern and the adder length should shrink as computation approaches this point. The adder must be able to operate on any bit position and properly handle overflow. Figures 5 and 6 show two adder designs which fit these criteria.

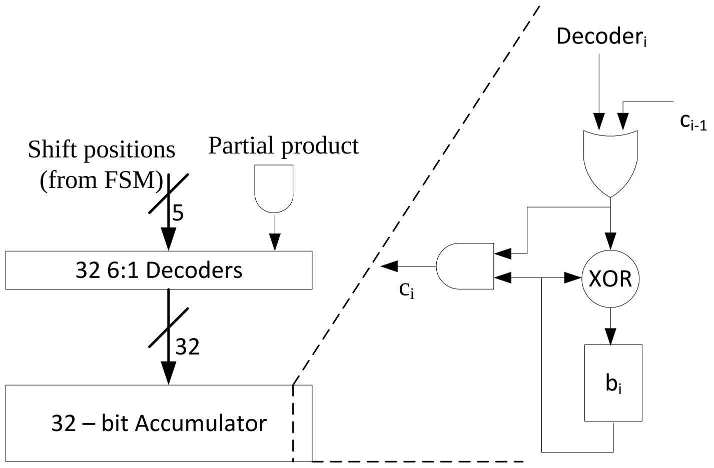


Figure 5: A modified adder using decoders to shift the partial product.

Figure 5 shows a suitable adder for single bit precision SQUASH. An individual adder cell is shown, which is a half adder with an additional *or* gate before the *carry in*. The decoders shift the partial product to the appropriate column: effectively varying the width of the adder from one to thirty-two bits. The control signals for the decoder are efficiently generated because the FSM already contains a counter that corresponds to the current column in the squaring operation. The adder is enabled by combining the enable signal with the partial product with an *and* gate. This is possible since an addition of zero does not change the stored value. The figure does not show the shift connections between adder registers.

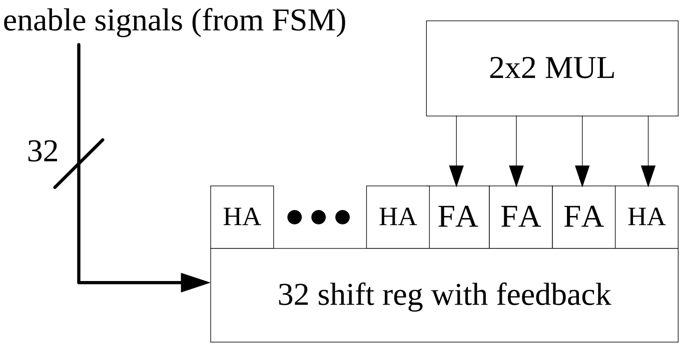


Figure 6: Adder width can be varied with individual register enable signals.

Full adders can be minimized for SQUASH with multiple bit precision as shown in figure 6. The partial products are aligned by shifting the register bits and the adder width can be adjusted with enable signals. This approach can be used for arbitrary precision. The architecture for squaring optimized SQUASH is shown in figure 7 and consists of two NLFSRs and the adder. For single precision: either adder can be used. Multiple bit precision requires the adder from figure 6. The adder complexity and FSM complexity is increased but the auxiliary register and circuitry to switch state between the auxiliary register and NLFSRs is eliminated.

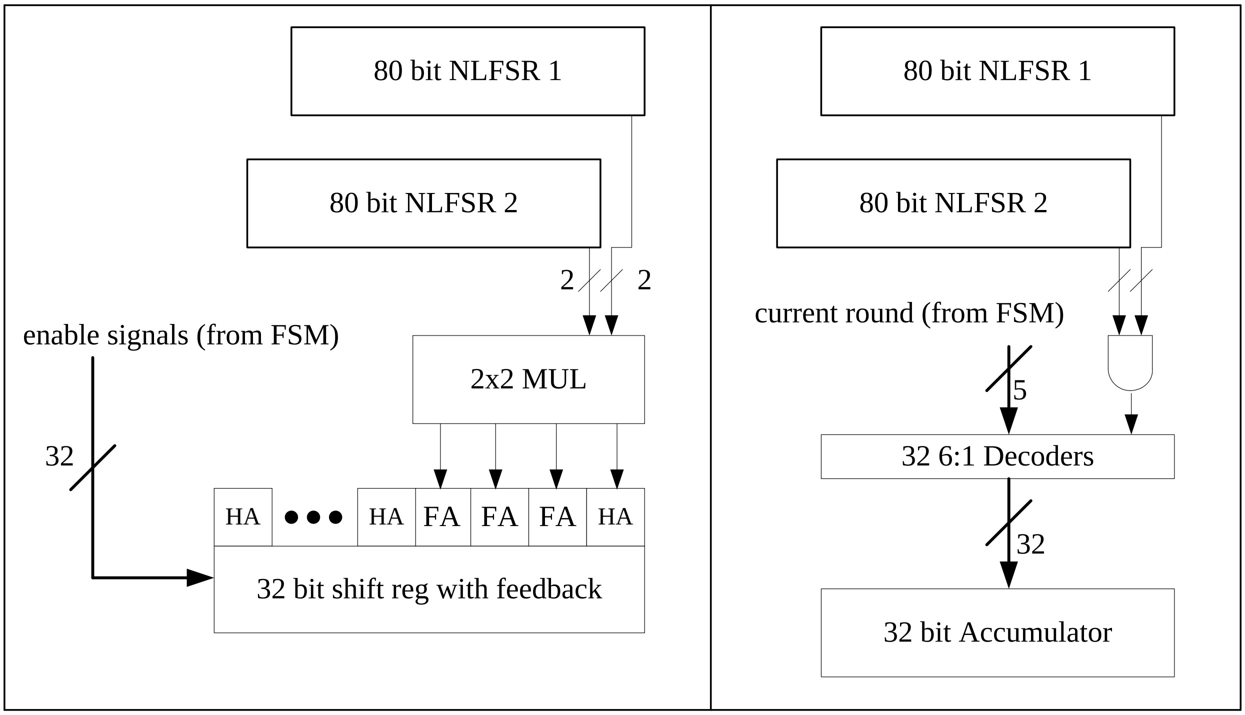


Figure 7: Dual bit (left) and single bit (right) SQUASH with the two proposed adders.

The squaring optimized SQUASH uses the same column method to compute SQUASH except partial products that are reoccurring are only computed once. The adder allows reoccurring partial products to be shifted left. The optimized SQUASH runs as follows. First both NLFSRs are seeded and run for 256 cycles to obfuscate the seed value. Next both NLFSRs run to the position corresponding to the non-reoccurring partial product in the first guard bit; this completes the initialization step. From this point the FSM cycles through all the partial products in alternating order for m_1, until a chosen point is reached: which is before risk of overflow (recall $m^2 = m_1 + m_2 \bmod n$). The NLFSRs are adjusted for m_2 and the same method is used to compute the partial result of m_2, up until the risk of overflow point. At this step the adder is shifted right, as in Algorithm (1), until the lsb of the adder corresponds to the current column. After the adder shift, computation of m_2 resumes until the new risk of overflow point: at which the NLFSRs are aligned again, and computation of m_1 resumes. This cycle is repeated until the entire result is computed.

2.6 Synthesis Results

Table 1 shows synthesis results for non-optimized SQUASH as well as squaring optimized SQUASH for single and dual bit precision. All designs are thirty-two bit output with sixteen guard bits. Synthesis and power simulation was done with Synopsys Design Vision. A virtual standard cell library was used based on UMC 130 nm process which was obtained from Faraday [20]. For area comparisons, we also synthesized Grain-80 which is one of the smallest cryptographic primitives. The Grain-80 description was VHDL and was obtained from OpenCores [21]. All our descriptions were written in Verilog. The clock was set to 100 kHz.

As shown in table 1, optimized SQUASH has a clear advantage over the non-optimized design. The twelve bit accumulator used in the non-optimized design is small but does not allow for squaring optimization. In addition, the added circuitry to reduce alignment shifts adds significant area pushing the total area higher than the optimized design with the much larger adder. The optimized design achieves nearly a 50 % reduction in latency; and the third entry, an optimized design with dual bit precision achieves an 86 % reduction in latency and is smaller than the single bit non-optimized SQUASH. The power numbers are also lower for the optimized designs.

Our comparison focuses on thirty-two bit output SQUASH with sixteen guard bits which uses an eighty bit NLFSR to generate *m*; but optimized SQUASH is preferable for all configurations. In the non-optimized design, increasing NLFSR width results in an increase in area dedicated to the auxiliary register and the supporting circuitry. This register and circuitry is eliminated in the optimized design and the adder complexity remains unchanged. The optimized design eliminates approximately half of the partial products; however alignment shifts increase when output length increases. The increase in latency due to extra alignment shifts is much less than the decrease due to elimination of repeating partial products; as a result, the optimized design will outperform the non-optimized design in terms of latency, in addition to having lower area and power consumption.

Table 1: Synthesis results.

Design	Area (GE)	Cycles	Dynamic Power	Leakage Power
Non-Optimized SQUASH	3169	62.8k	34.8 nW	13.5 nW
Optimized SQUASH	2646	31.8k	24.7 nW	11.0 nW
Dual-bit Optimized SQUASH	2825	8.6k	23.6 nW	11.8 nW
Grain-80	1391	N/A	22.2 nW	5.6 nW

CHAPTER 3

SQUEEZING SQUASH

3.1 Introduction

Squaring optimized SQUASH achieves a reduction in area but the duplicate NLFSR is still costly. Reducing SQUASH to one NLFSR would be ideal since the second NLFSR adds no additional state bits, only area overhead. Of course the reason two copies of state are used in squaring optimized SQUASH, and three in the original design, is due to NLFSR alignment shifts dominating latency without the added copies. In order to reduce SQUASH down to a single NLFSR, the NLFSR would need to produce both multiplicand and multiplier in the same cycle for each partial product.

In [1], a concrete SQUASH proposal is discussed to address this issue; however, the proposal glosses over some key points which add significant area overhead. We discuss this proposal and introduce our own SQUASH variant which we call permuted SQUASH (PSQUASH). Our proposed variant achieves the smallest area while greatly reducing alignment shifts.

3.2 SQUASH with a Single NLFSR

If only one NLFSR is used in the example designs from the last chapter, a significant amount of time would be dedicated to sliding the NLFSR back and forth to obtain the correct bits. In such a scheme, the designer would want to add additional registers in order to look at a larger window of bits during each shift. And, if the area·delay product is examined: it becomes clear that the most efficient architecture is another independent window (i.e. another NLFSR). The additional NLFSR does not enhance security because it uses the same state. Its purpose is simply to reduce alignment shifts. Figure 8 illustrates this issue.

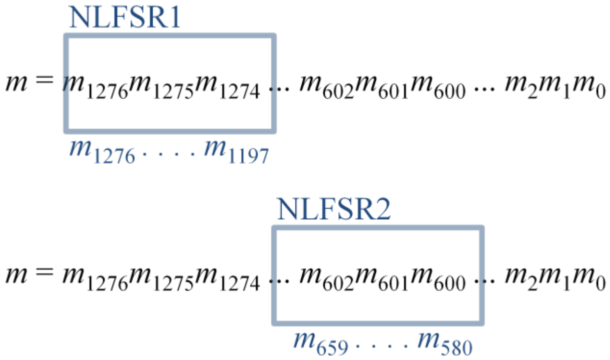

NLFSR1
$m = m_{1276}m_{1275}m_{1274} \ldots m_{602}m_{601}m_{600} \ldots m_2m_1m_0$
$m_{1276} \cdot \cdot \cdot \cdot m_{1197}$
NLFSR2
$m = m_{1276}m_{1275}m_{1274} \ldots m_{602}m_{601}m_{600} \ldots m_2m_1m_0$
$m_{659} \cdot \cdot \cdot \cdot m_{580}$

Figure 8: The NLFSR only contains a window of the entire sequence.

Removing the additional NLFSR without increasing latency due to alignment shifts significantly reduces the area and power consumption of SQUASH. Avoiding the penalty of alignment shifts with a single NLFSR requires that the NLFSR always contains the current multiplicand and multiplier bits within the current state; as well as the next multiplicand and multiplier bits contained in the next state, for all states.

3.3 A Single NLFSR SQUASH Proposal

A single NLFSR SQUASH design is proposed in [1] and shown in figure 9. Adi Shamir proposed this design for k = 128 which makes $n = 2^{128} - 1$. The NLFSR is seeded and run for 512 cycles to obfuscate the seed; at which point the NLFSR stops, containing all bits of m since m is now 128 bits long. The FSM then computes SQUASH, in the same manner as Algorithm (1), but adjusted to k = 128 and eight guard bits. Adi Shamir claims this design is smaller than Grain-128. The NLFSR is the non-linear portion of Grain-128 which like the Grain-80 NLFSR is a sum modulo two of a linear function and a quadratic bent function with the following feedback polynomial *g(x)*:

$$g(x) = 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{60}x^{44} + x^{67}x^{63} + x^{88}x^{80} + x^{101}x^{69}$$
$$+ x^{111}x^{110} + x^{117}x^{115} + x^{125}x^{61}$$

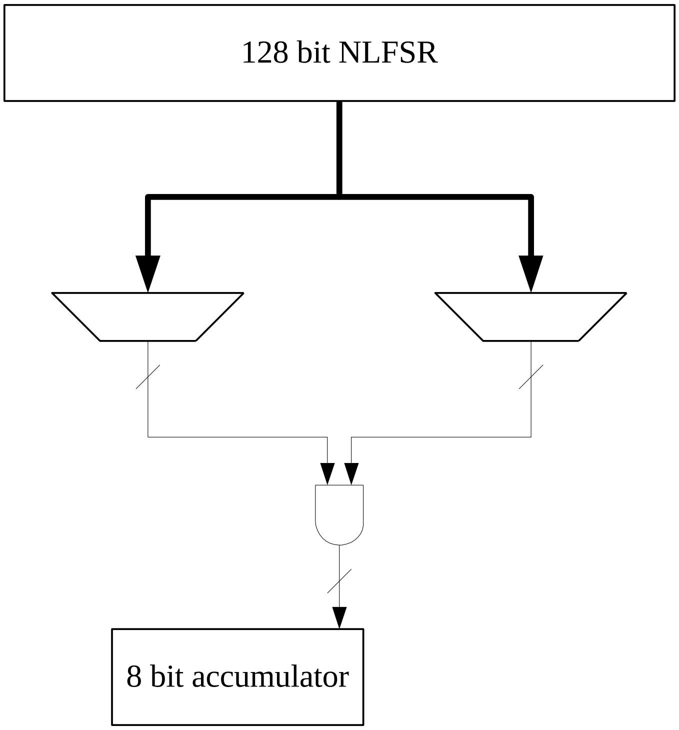


Figure 9: A single NLFSR design proposed in [1].

There are several issues with this proposed design; the first of which is the limited choice for the modulus *n*. This design stores *m* instead of generating it on the fly, which works well when for *k* = 128; but as *k* is increased, the cost of storing *m* quickly grows to thousands of gates. Using *k* = 128 also violates the assumption of a modulus with no known factors; therefore, the formal proof of security is not applicable. If an attack is devised on SQUASH with a factorable modulus, this design will not scale for a larger modulus. The second issue with this design is that it is not as small as [1] claims. The NLFSR is the biggest block in Grain and consumes half the total area. Selecting each bit from the NLFSR when computing SQUASH, without additional alignment shifts, requires two 128:1 multiplexors (as shown in figure 9). These structures and the corresponding control circuitry add significant area overhead. When combined with the counters and FSM complexity of SQUASH, this design is likely to be roughly the size of Grain-128.

3.4 Permuted SQUASH

Reducing squaring optimized SQUASH down to a single NLFSR can be achieved with a static permutation on the NLFSR sequence output: resulting in any pair of multiplicand and multiplier existing within the NLFSR state. As described above, SQUASH uses a NLFSR to generate sequence *m* and computes m^2 mod *n*; likewise, permuted SQUASH generates *m*` using the same NLFSR structure but then permutes *m*` into *m* using a static permutation and computes m^2 mod *n*.

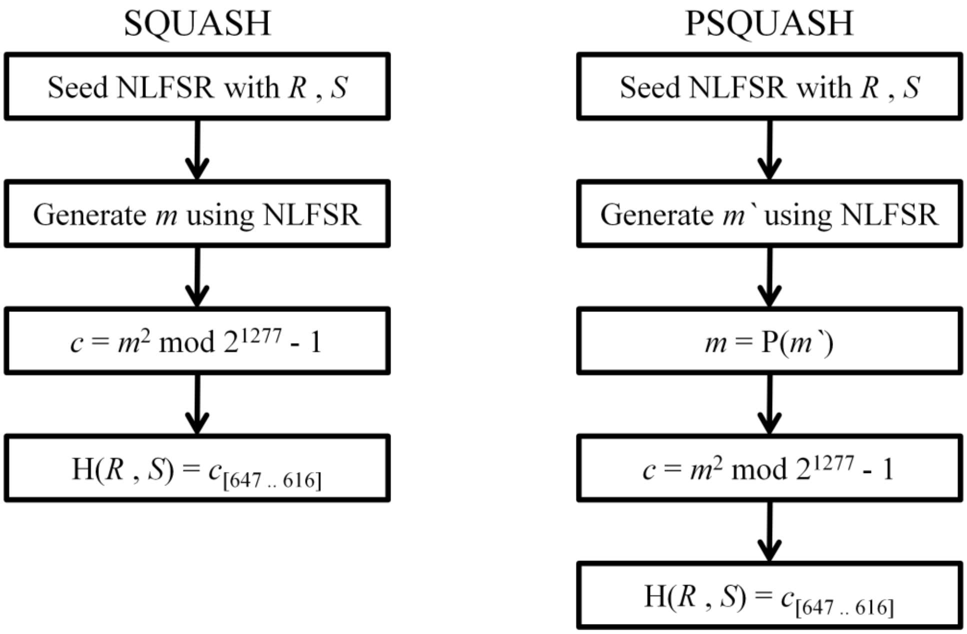

SQUASH
Seed NLFSR with R , S
Generate m using NLFSR
$c = m^2 \bmod 2^{1277} - 1$
$H(R , S) = c_{[647 .. 616]}$
PSQUASH
Seed NLFSR with R , S
Generate m` using NLFSR
$m = P(m`)$
$c = m^2 \bmod 2^{1277} - 1$
$H(R , S) = c_{[647 .. 616]}$

Figure 10: PSQUASH adds a permutation to the flow.

The added permutation does not undermine the security proof of SQUASH put forth in [1]. SQUASH is based on the strength of Rabin's PKC and this change does not reduce *n*. It is important to note that while Rabin's PCK is not affected by the choice of encrypted message (in our case NLFSR output *m*); care must be taken when designing the sequence generator for SQUASH. The authors in [17] show that SQUASH can be broken when a linear sequence generator is used to generate *m*. In this case, we use the same non-linear generator and perform a linear permutation on the final output which does not destroy non-linearity. This method of generating *m* (or in this case *m*`) does not make permuted SQUASH susceptible to the attack described in [17].

An efficient permutation for PSQUASH has two requirements; it must eliminate alignment shifts, and it should consume less area and power than a duplicate NLFSR. Several possible permutations were evaluated. The final choice eliminates alignment shifts after the initialization step, and uses a fraction of the gates required by a duplicate NLFSR. This permutation can be constructed for any value of *k*. The permutation is based on the msb of SQUASH output and is simply the column squaring ordering for the msb. Figure 11 shows the permutation, which starts with the middle of the m_2 column and ends with the middle of m_1 column. The resulting mapping has the multiplier and multiplicand bits in an order that is based on the current round and is fairly efficiently multiplexed. Only one large multiplexor is needed and the control signals can be generated using circuitry with a fraction of the area cost of a duplicate NLFSR.

NLFSR	0	1	2	3	…	646	647	648	649	650	651	…	1274	1275	1276
P(NLFSR)	324	323	325	322	…	647	0	648	1276	649	1275	…	961	963	962

Figure 11: An example permutation for *k* = 1277, 32 output bits, and 16 guard bits.

Figure 11 is color coded to show how the msb of SQUASH output is used to build the permutation. This permutation is for the SQUASH parameters in Algorithm (1). The green portion corresponds to the partial products of m_2 or as in Algorithm (1) the partial products computed by $m_v * m_{j\text{-}v \bmod k}$ for v = 0, 1, ..., 647. The blue portion corresponds to the partial products of m_1 or as in Algorithm (1) the partial products computed by $m_v * m_{j\text{-}v \bmod k}$ for v = 648, 649, ..., 1276. This approach can be used to map any set of SQUASH parameters but it is only efficient when the NLFSR width does not exceed 2 * (guard + output bits).

The maximum distance for any two operands is 2 * (guard + output bits); therefore, the NLFSR length must not be less than the maximum distance to eliminate alignment shifts. With our example parameters (32 output bits, 16 guard bits), 96 bits of NLFSR must be available in order to avoid alignment shifts. For the smallest SQUASH output (thirty-two) with only eight guard bits, an eighty bit NLFSR can be used. This restriction means that some combinations of NLFSR length and SQUASH output are not suitable for PSQUASH; however, PSQUASH is far more flexible than the single NLFSR proposal in [1] and places no restrictions on the size of the modulus. PSQUASH also requires smaller multiplexors than the single NLFSR proposal in [1]. Only one 80:1 and one 2:1 multiplexor is needed in PSQUASH for thirty-two bit output and eight guard bit configuration; the proposal in [1] requires two 128:1 multiplexors.

PSQUASH is run by seeding the NLFSR and running for 128 cycles. This number is cut in half from its SQUASH counterpart, because the NLFSR shifts two bits at a time. An additional twenty-four cycles are needed to move to the starting position for the lsb, since the starting position corresponds to the starting position for the msb bit. The number of shifts in this step is double the sum of output and guard bits. After these initialization steps, the NLFSR is shifted once per partial product. Each unique partial product is generated without repetition for each output bit: with the non-reoccurring term from the next column generated in the current column. This ordered generation of partial products without alignment shifts requires a simple accumulator and is optimized for squaring. This behavior is achieved by selecting the multiplicand and multiplier using multiplexors. The control signals are generated by an arithmetic relation with the counters values already required by SQUASH. Figure 12 shows the design for thirty-two output bits and sixteen guard bits. The NLFSR width and multiplexor width can be reduced to eighty bits if only eight guard bits are used.

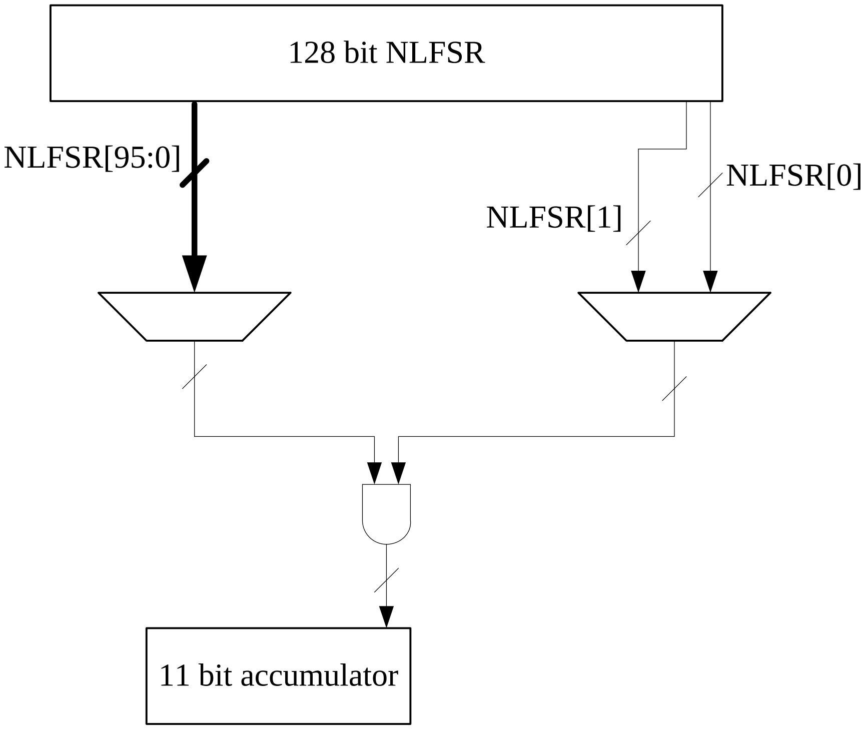


Figure 12: A PSQUASH design.

3.5 Synthesis Results

We synthesized PSQUASH using two NLFSR sizes, with the design flow described in the last chapter. The smallest PSQUASH configuration of thirty-two output bits and eight guard bits can use an eighty bit NLFSR and was synthesized with 1624 GE. A 128 bit NLFSR was used for thirty-two bit output with sixteen guard bits. Both designs are synthesized under 2k GE, which is a sizable improvement over the non-permuted SQUASH. PSQUASH also consumes less power than the optimized SQUASH designs and reduces latency by eliminating more alignment shifts. PSQUASH outperforms SQUASH in all categories. The only disadvantage of PSQUASH is the restrictions on minimum NLFSR size; however, unlike the SQUASH designs which need multiple copies of state, PSQUASH adds state bits. A wider NLFSR increases the security by increasing pre-image resistance. Even if the added state is unnecessary: PSQUASH outperforms SQUASH.

Table 2: Synthesis results with PSQUASH added.

Design	**Area (GE)**	**Cycles**	**Dynamic Power**	**Leakage Power**
Non-Optimized SQUASH	3169	62.8k	34.8 nW	13.5 nW
Optimized SQUASH	2646	31.8k	24.7 nW	11.0 nW
Dual-bit Optimized SQUASH	2825	8.6k	23.6 nW	11.8 nW
128 NLFSR PSQUASH	1918	30.2k	17.3 nW	8.9 nW
80 NLFSR PSQUASH	1624	25.1k	18.3 nW	7.7 nW
Grain-80	1391	N/A	22.2 nW	5.6 nW

CHAPTER 4

SQUASH ON COMPUTATIONAL RFID

4.1 Introduction

Computation RFID (CRFID) refers to contactless smartcards which operate like traditional finite state (FSM) machine driven RFID tags, but use an embedded microprocessor [22]. Electronic product tags and access cards are examples of traditional tags; these tags perform a set of tasks and are typically on custom silicon. The high non-recurring engineering cost (NRE) of these devices is absorbed by the incredibly large volumes. CRFID NRE costs can be significantly lower because development is in software. In addition to lower NRE costs, CRFID promise to provide the flexibility of general purpose processing to RFID.

Using a microprocessor on a resource constrained device is not a new idea. In fact, smartcards have used microprocessors like the Intel 8051 for years. While contact smartcards have used embedded microprocessors for years, contactless smartcards have been predominantly FSM driven. We believe that embedded microprocessors will become more prevalent in RFID. SQUASH benefits from these next generation CRFID tags because ALUs and memory address many of the issues that emerge in a hardware SQUASH implementation.

4.2 The WISP

The Wirelessly Powered Platform for Sensing and Computation (WISP) [23] is a prototype platform for CRFID (shown in figure 13). The WISP discussed here is the first generation model and is a MSP430F1232 microcontroller on a PCB board which adds circuitry that allows the device to harvest power and communicate like a typical UHF tag. The MSPF1232 is a sixteen bit microcontroller with 256 bytes of RAM and FLASH memory and 8k bytes of ROM. There is no hardware multiplier, which would be desirable for SQUASH, but it is still able to run SQUASH on harvested power.

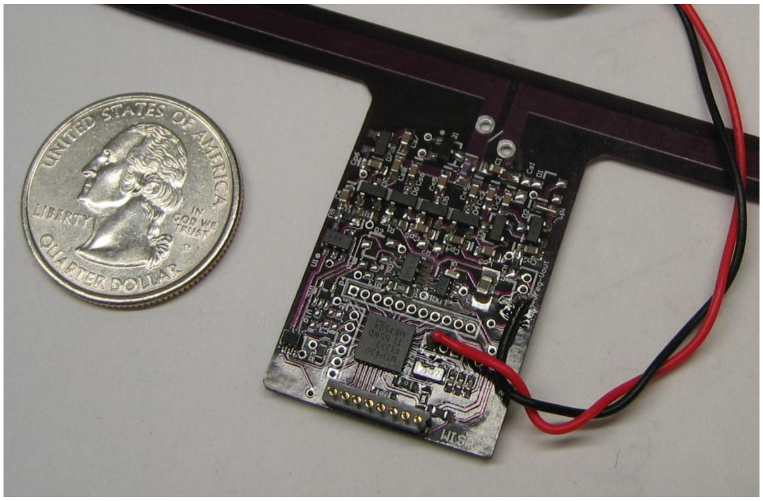


Figure 13: First Generation WISP.

Power harvested from the UHF reader is insufficient to run the microcontroller on the WISP. To address this issue, the WISP performs computation in a duty cycle. The microcontroller cycles between low power sleep mode and active mode, allowing charge to gather on the capacitor when in sleep mode. Figure 14 shows the voltage on the capacitor as the WISP charges up and performs computation. The blue trace shows the voltage on the capacitor, which drops when the microcontroller switches to the active cycle (yellow trace). The stored charge is regulated to a target voltage of 3.3 V. The microcontroller operates as long as the voltage on the capacitor does not drop below 3.3V during active cycle.

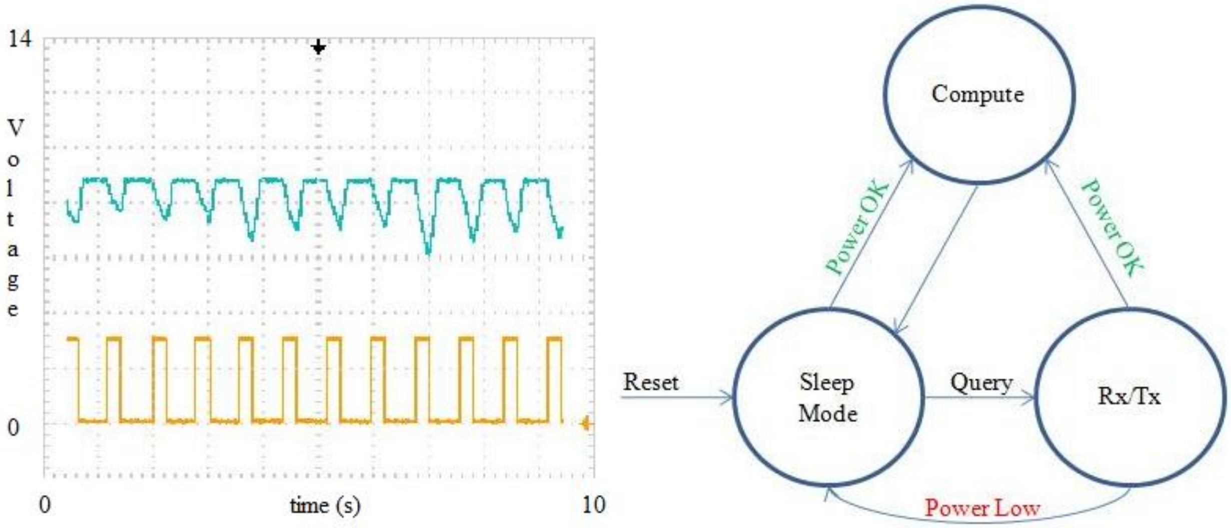


Figure 14: WISP power cycle. Charge is gathered during sleep mode and discharged during computation and communication.

4.3 SQUASH on the WISP

IAR Embedded Workbench [24] was used for development and the source code was written in C. The source code for communicating with the Alien UHF reader [25] and cycle the microcontroller through the duty cycle was provided by Intel Research Seattle [26] with the WISP. The microcontroller was programmed with a USB-FET430UIF [27]. The RFID reader was controlled using the One Wisp GUI which was developed by Intel Research Seattle.

We evaluated the potential for SQUASH on CRFID by programming the WISP to compute a thirty-two bit SQUASH output with sixteen guard bits and send the thirty-two bit output to the UHF reader. To reduce alignment shifts, two copies of the eighty bit NLFSR suggested in [1] are stored in memory. As mentioned previously, since SQUASH uses a Mersenne number for the modulus, the resulting hash is some chosen subset of $b^2_{[k\text{-}1\ ..\ 0]}$ (bottom half) + $b^2_{[2k\text{-}1\ ..\ k]}$ (top half): where b is the k bit sequence generated by the NLFSR. Each time computation switches between the bottom and top half: the NLFSR position must be adjusted. We compute the full bottom half before computing the top half to minimize alignment shifts.

Column squaring was used to perform the squaring operation similar to the approach used in hardware implementation. The general rule for choosing between column and row squaring is the cost of memory access versus cost of carry propagation. We chose the column method because we could not efficiently propagate carries in C, but from our results we will show that row method squaring is the better choice for SQUASH, especially if carries can be handled efficiently.

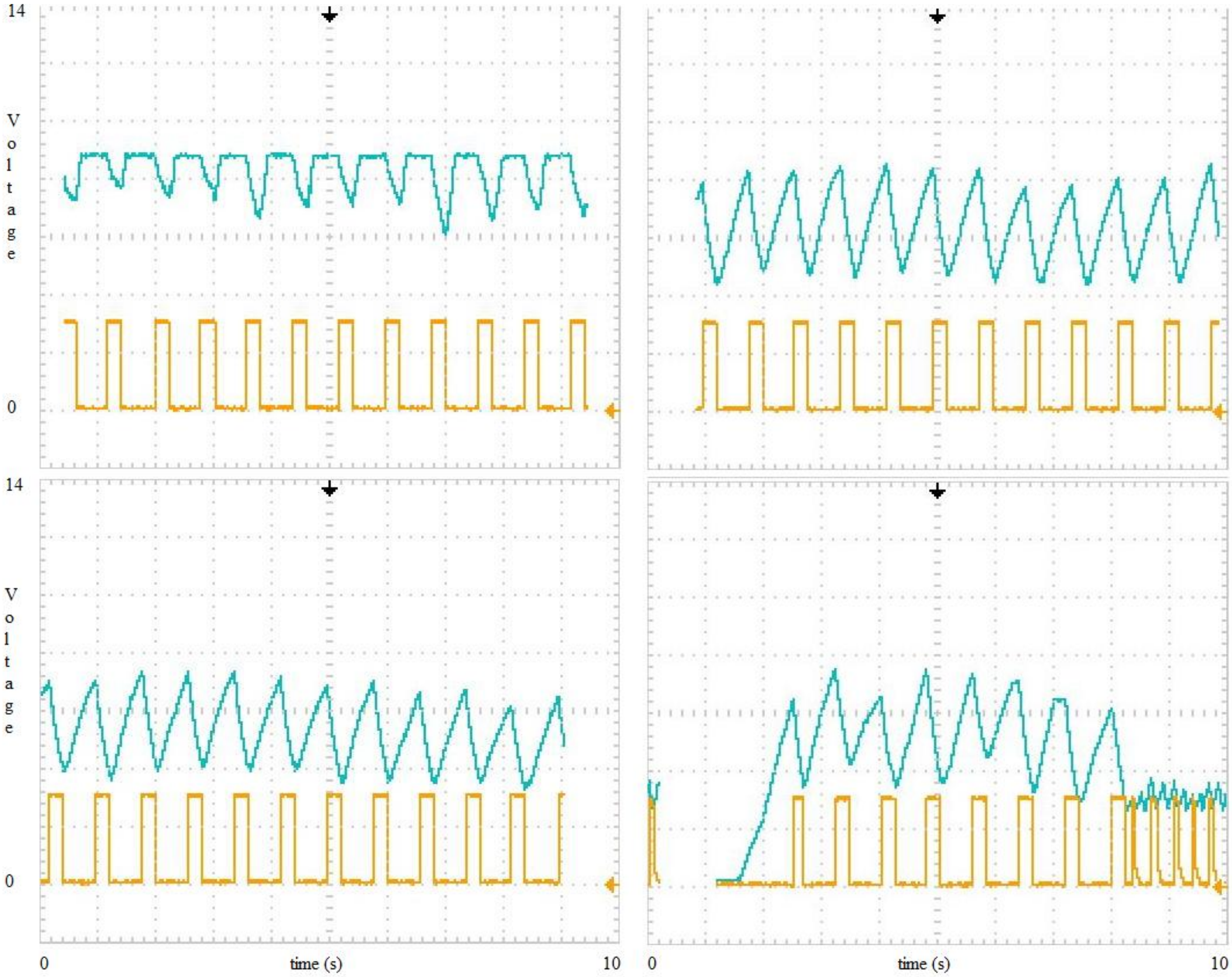


Figure 15: WISP Power trace when computing and transmitting SQUASH packet. Blue trace is voltage and yellow trace denotes computation. WISP distance from reader, from top left to bottom right: 6", 12", 24", 30".

Multiplication and NLFSR shifts were performed in eight bit precision. The WISP does not have a hardware multiplier, and multiplications are computed using Horner's method [28]. Computing thirty-two bit output SQUASH with sixteen guard bits requires 468 multiplications, and each multiplication requires both NLFSRs to shift for a total of 936 shifts. In addition to the NLFSR shifts during multiplication, 70 shifts are needed to initialize the NLFSRs and 78 shifts are needed to align the NLFSRs when computation switches from the bottom to the top half. A total of 468 multiplications and 1084 NLFSR shifts are needed to complete the algorithm.

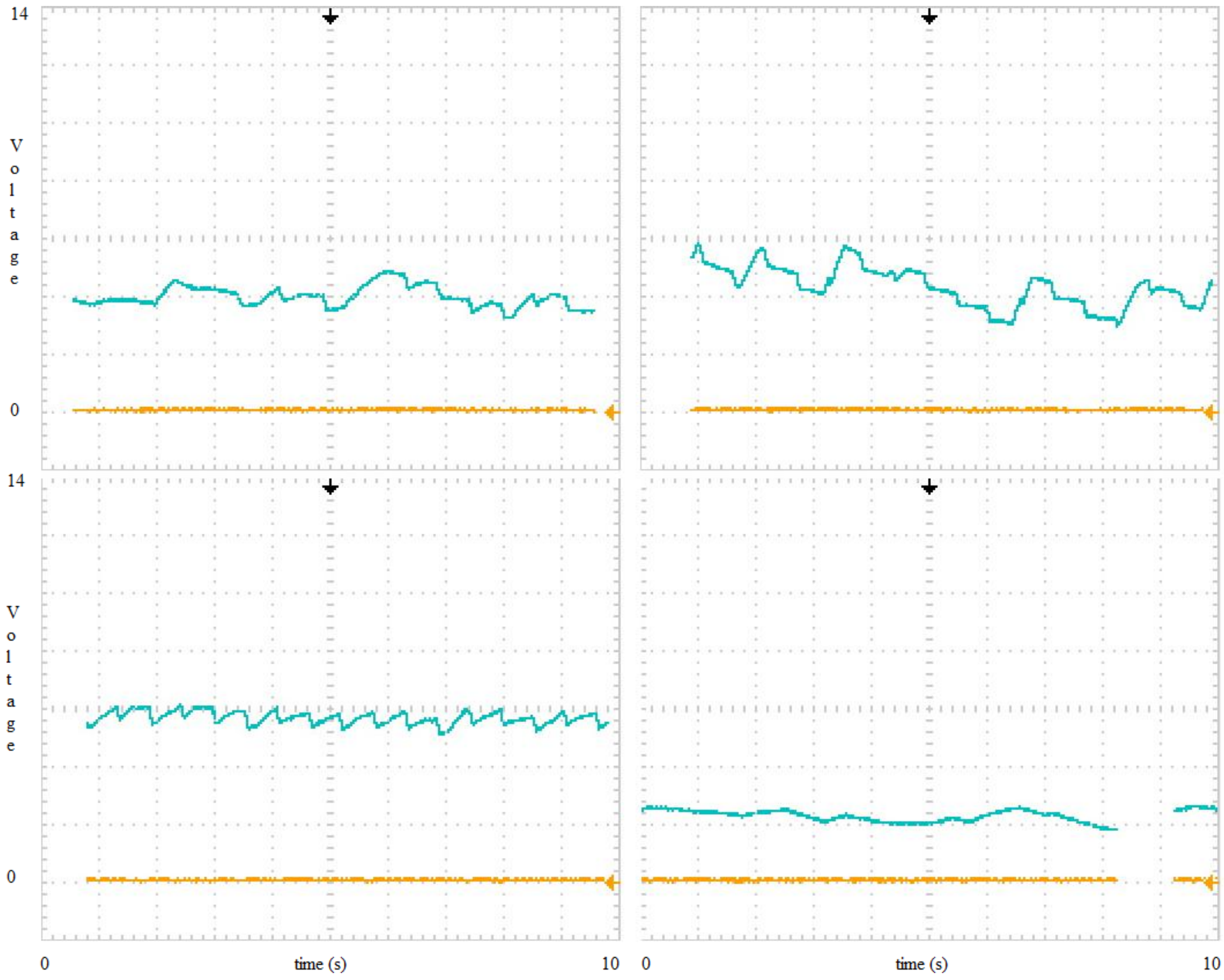


Figure 16: WISP power traces for static ID. Blue trace is capacitor voltage and yellow trace is computation of packet (static ID is nearly instant). Distance from reader from top left to bottom right: 2', 3', 4', 5'

The WISP clock speed was set to 3 Mhz and SQUASH takes approximately 240 ms to complete. The WISP was programmed to run the full algorithm before switching to low power mode which reduces the WISPs effective range. Figure 15 shows several WISP power traces while running SQUASH with successful transmissions up to two feet from the reader; however, power harvested is inadequate to operate when the distance from the reader exceeds thirty inches. Figure 16 shows power traces of a WISP programmed to transmit a static ID, which requires minimal computation. The read range of a static ID is between four and five feet and is the maximum effective read range of the device.

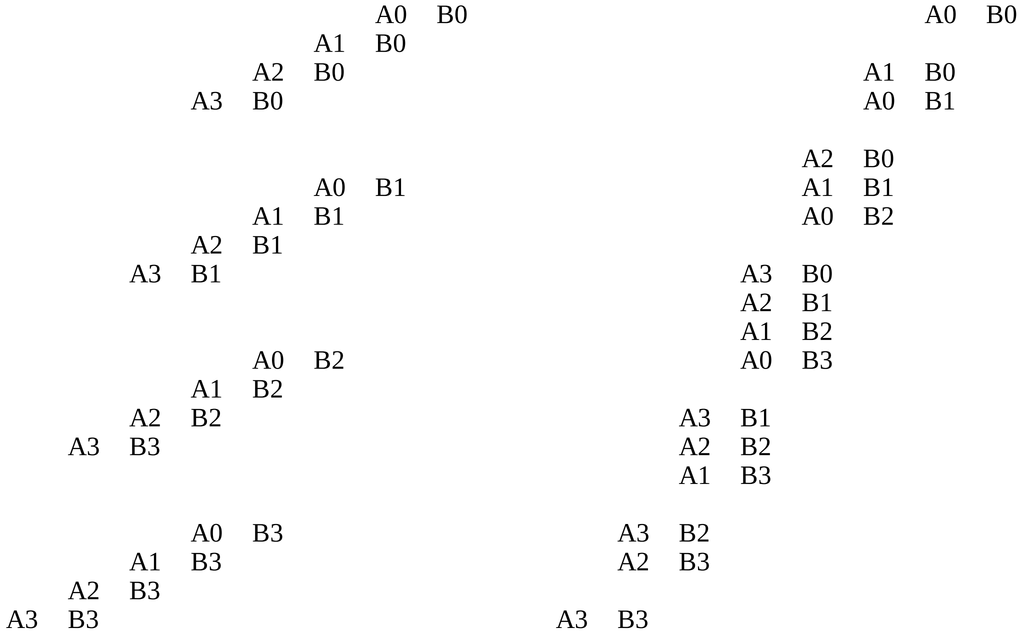
Table 3: SQUASH and RC5 specs on 3V WISP.

	MEM (bytes)	CODE (bytes)	Latency (ms)
SQUASH	29	1848	240
RC5	112	1518	9.4

The first generation WISP is a poor target for SQUASH primarily because it lacks a hardware multiplier. Table 3 shows the code size and latency of SQUASH and a RC5 implementation on the same device [29]. While RC5 is not a MAC but a block cipher, a MAC can be constructed using a block cipher [30]; therefore SQUASH is not well suited for the first generation WISP because a hardware multiplier is essential for fast SQUASH. The latency of SQUASH on the WISP can be decreased by using the method described in the next section; however, without a hardware multiplier, SQUASH will still perform poorly.

4.4 Efficient Software SQUASH

From a software standpoint, SQUASH is similar to a multi-precision arithmetic problem. While it is possible to use successive convolutions to compute SQUASH on a microprocessor, advanced squaring techniques offer a significant advantage. For example, using successive convolutions requires twice as many multiplications as column or row based squaring. There are a few factors to consider when choosing a multi-precision algorithm for an embedded microprocessor. These include the cost of memory access versus the cost of a multiply operation, the number of available registers, and the availability of an *add with carry* instruction.



```
                              A0   B0                                     A0   B0
                         A1   B0
                    A2   B0                                          A1   B0
               A3   B0                                               A0   B1

                                                                A2   B0
                         A0   B1                                A1   B1
                    A1   B1                                     A0   B2
               A2   B1
          A3   B1                                          A3   B0
                                                           A2   B1
                                                           A1   B2
                    A0   B2                                A0   B3
               A1   B2
          A2   B2                                     A3   B1
     A3   B3                                          A2   B2
                                                      A1   B3

               A0   B3                           A3   B2
          A1   B3                                A2   B3
     A2   B3
A3   B3                                     A3   B3
```

Figure 17: Row wise (left) and column wise (right) ordering.

The authors in [31] point out that while advanced multiplication techniques, like Karatsuba Ofman [32], reduce multiplications over column or row methods; they are not well suited for embedded applications due to their recursive nature. The reason being: recursive function calls can quickly exhaust the memory allocated for the stack. Column and row methods are not recursive and better suited for the limited memory of embedded devices. Each method has its own advantage and can be optimized for squaring. Row based multiplication reduces memory accesses because the multiplicand is reused, but this comes at a cost of increased additions due to carry propagation. This method also requires more available registers to be effective [31]. Column based multiplication requires minimal register space because the product is computed in columns starting with the least significant position; however the cost is increased memory accesses.

In SQUASH, the input to the squaring algorithm is not available in memory but must be generated with an NLFSR. This leads to key considerations when choosing a squaring method:

- NLFSR access is costly
- NLFSR is not random access

The first observation states that in SQUASH, the multiplier and multiplicand need to be generated by the NLFSR which is computationally expensive. This means that a row based method is always preferable since it reduces NLFSR shifts. The second observation is that the NLFSR produces an ordered output; that is, values are produced from *left to right* or *right to left* and all values between the current state and target state must be produced in order to move to the target state. This means ordering can have a significant impact on performance.

In an elliptic curve cryptography (ECC) on embedded microprocessors implementation, [31] proposes a hybrid method when there is insufficient register space which breaks the computation into columns but performs row wise multiplication within the columns. This is essentially what SQUASH forces, because the output is some subset of the full result and it can be computed with the hybrid method described in [31]. However, unlike other arithmetic operations in which the operands are expected to be in memory, SQUASH's operands must be generated on the fly. This means that the column width should be the full SQUASH length even if there are insufficient registers: because a memory load and store is preferable to NLFSR shifts. In addition to the reduction in NLFSR shifts from a static multiplier in each row, a further reduction in NLFSR shifts can be achieved because the NLFSR stores a subset of the multiplicand.

To see how the ordering of partial products has a big impact on NLFSR shifts, refer to figure 18. In the column method, each column requires *k*/2 partial products; but each partial product requires a NLFSR shift for the multiplier and multiplicand; for a total of *k* NLFSR shifts per column (recall *k* is the bit length of the modulus). The row method is expected to reduce NLFSR shifts with each row requiring a single shift for the multiplier and *l* shifts for the multiplicand (recall *l* is the combined length of output and guard bits). Notice that any subsequent rows are only separated by one shift and the total length *l* is typically shorter than the NLFSR width. As a result, the entire multiplicand can be contained within the NLFSR state, and only one shift is needed to move to the next row. This reduces the NLFSR shifts needed for row method to two shifts per row when NLFSR width is greater than *l*. If *k* = 1277, *l* = 48, and NLFSRs shift one bit per cycle and alignment shifts are not considered, the column method requires *k* * *l* = 61.3k shifts; on the other hand, the row method requires only 2 * *k*/2 = 1278 shifts.

Column 3	**Column 2**	**Column 1**	**Row 3**	**Row 2**	**Row 1**
		i = 0 .. 300 j = 600 ..300			i = 0 j = 647 .. 600
	i = 0 .. 300 j = 601 ..300			i = 1 j = 646 .. 599	
i = 0 .. 301 j = 602 ..301			i = 2 j = 645 .. 598		

Figure 18: First three columns and rows of a partial SQUASH computation.

The extra reduction in NLFSR shifts outlined above is possible when NLFSR width is greater than *l*. If this is not the case, there are two options for the software SQUASH designer. If there is available memory, a third copy on NLFSR state can be stored which doubles the length of multiplicand available in memory without NLFSR shifts. The second option is to reduce the column width to the length of the NLFSR. In this case, reducing the width of the column can be more efficient than the full output length column size. The rows should be computed from right to left to reduce alignment shifts for the NLFSRs.

4.5 Best Case SQUASH Performance on CRFID

For our test case, the WISP device was chosen for its unique characteristic of operating on passive power but it was not an ideal target because it lacked a hardware multiplier. In addition, we used column squaring instead of the optimal method outlined above which resulted in high latency. For the ideal case, a hardware multiplier is a must; also, an *add with carry* instruction and assembly coding is needed for efficient carry propagation. If these conditions are met, assuming an eight bit microprocessor with SQUASH parameters k = 1277 and thirty-two bit output with sixteen guard bits; 468 multiplications, 156 NLFSR shifts during squaring, and 112 NLFSR shifts for initialization and alignment are needed to complete the algorithm. It is clear that NLFSR shifts will dominate latency since a single shift takes approximately 230 cycles. Even if a very optimistic five cycles are needed to compute and add the result of each multiplication, the total clock cycles for additions, multiplications, and NLFSR shifts is approximately 61k cycles.

In conclusion, software SQUASH is a good fit for CRFID but requires a hardware multiplier. In [1], the author claims that SQUASH will benefit from ALUs which may be found on future RFID tags. We show that despite the larger precision of embedded ALU's, the inefficiency of the NLFSR in software prohibits SQUASH from distinguishing itself from software block ciphers, which could be used in the same way. The algorithm proposed in [1] performs poorly, requiring 2 * *k* * *l* NLFSR shifts; but this number can be reduced to *k* using the hybrid row method technique described above.

Software SQUASH may be better suited for RFID than the hardware counterpart. A more efficient software NLFSR would greatly reduce the latency of software SQUASH, making SQUASH attractive for CRFID. Latency is critical because we expect many CRFID applications to be battery powered. In that situation, longer execution time generally means more energy consumption. SQUASH has a fairly small memory footprint, and if a different NLFSR was used, SQUASH would be a good choice for CRFID.

CHAPTER 5

CONCLUSIONS

5.1 Practical Applications for PSQUASH

While PSQUASH is larger than most lightweight authentication methods and has considerably larger latency, it has several strengths over these other methods. The formal proof of security, a non-probabilistic method, and no random number generation requirement are the key advantages of PSQUASH. Our synthesized design for 128 bit NLFSR PSQUASH is just below the 2k GE mark, which is within the area budget for RFID tags. Authentication schemes that rely on a backend server to minimize tag resources may require significantly lower area, but the larger area of PSQUASH may be justified in applications where a backend server assisted approach is not practical. Another consideration is that no additional area needs to be dedicated to a TRNG, which may otherwise be unnecessary for a tag.

The area of PSQUASH is within reasonable constraints but a greater limiting factor is the high latency. Specifically, the supply chain application may not be a good choice for PSQUASH due to the high latency. Practical applications for SQUASH are those in which latency is not a critical factor. Examples of such applications may be access cards, some micro payment systems, and anti-counterfeiting tags. In the case of a supply chain application or inventory control, a retail shelf or conveyor belt may carry hundreds of tags; and even if these tags can be authorized concurrently, an automated system may be inefficient when individual responses from tags require a relatively large delay. The same delay may be considered relatively low in an access card application where the delay is perceived by a human operator passing an access point. The latency can be significantly reduced by using multi-bit precision, but this increases area consumption.

Tags for replacement parts are a good fit for PSQUASH. In the cell phone replacement battery application described earlier, the cell phone is a networked device capable of accessing a backend server, but using the limited bandwidth may be undesirable. A stronger case can be made for devices that are unlikely to have network access. Many products have a business model that relies heavily on replacement parts. These products could benefit from a PSQUASH tag. Some examples are automotive replacements parts, replacement parts for consumer electronics like cameras, and copier and printer toner cartridges. In these applications, a PSQUASH tag can increase revenue because parts can be authenticated.

5.2 Comparing SQUASH and PSQUASH to Hashes Constructed from Symmetric Ciphers

Adi Shamir points out that hash functions like SHA family owe a large amount of their complexity to collision resistance; therefore, SQUASH is better suited because collision resistance is not necessary for a challenge response MAC [1]. In other words, SQUASH is effective because it is one way; however this is also true of MACs constructed with symmetric ciphers. In [33] the authors make a case for using AES to construct hash functions for RFID due to lower area and power consumption over other hash functions including SHA-1, SHA-256, MD4, and MD5. CMAC can be constructed with symmetric ciphers [30]; furthermore, since the challenge response MACs do not require collision resistance, any symmetric cipher can be used.

A SHA-256 ASIC design is synthesized in [33] with 10,868 GE and [4] presents an AES design using 3,595 GE with a latency of 1016 cycles. All the SQUASH designs discussed in this thesis consume much less area than SHA-256. PSQUASH consumes significantly less area than the AES design, but has a latency that is an order of magnitude higher. While PSQUASH may be the better choice over AES when area is a primary concern and the additional latency can be tolerated, the advantage in area disappears when compared to lightweight ciphers like Grain or PRESENT. Both ciphers are comparable in size to the smallest implementation of PSQUASH with an eighty bit NLFSR, but have much lower latency. Full round PRESENT is two orders of magnitude faster than PSQUASH; also serialized PRESENT is an order of magnitude faster than SQUASH and requires only 1k GE [34].

The same is true for software implementations; however software SQUASH gives up less latency to software symmetric ciphers. Our work shows how NLFSR shifts can be greatly reduced over the original method of successive convolutions, but the inefficiency of NLFSR shifts in software still dominate latency. SQUASH can be a good choice over some steam ciphers from the eSTREAM project [35], but is still outperformed by a software implementation of PRESENT [36] and RC5 [29]. In addition, SQUASH requires a hardware multiplier, which is not typically required by symmetric ciphers. SQUASH may be improved if a faster software NLFSR is used in place of the proposed NLFSR. Even a marginal reduction in NLFSR complexity would greatly benefit software SQUASH, since the latency is heavily dominated by NLFSR shifts.

5.3 Our Contribution

In this thesis, we synthesized the generic SQUASH design proposed in [1] as an ASIC implementation. We showed that in addition to the large latency of SQUASH, area consumption is a concern. The SQUASH design proposed in [1] has a large area overhead due to the multiple copies of state, which are stored to reduce latency. We show that SQUASH can be optimized for squaring if the adder complexity is increased. This optimization reduces latency by eliminating repetition of reoccurring partial products. We also showed that the third copy of NLFSR state can be removed with a reordering of partial products. The removal of the third copy of state and the circuitry to switch values between the auxiliary register and NLFSRs, even with the addition of a more complex adder, reduces the area consumption of SQUASH.

The duplicate NLFSR in SQUASH contributes a significant area overhead. In his SQUASH proposal, Adi Shamir proposes a single NLFSR design but the area consumption of this design is understated and it is not scalable to a larger modulus. We propose a new variant called PSQUASH which adds a permutation to the NLFSR output. This permutation allows PSQUASH to use a simple accumulator and single NLFSR to compute m^2 mod n. The proposed change does not destroy non-linearity of the NLFSR and allows us to use the formal proof of security of SQUASH. The smallest PSQUASH configuration is synthesized with 1624 GE which is within the security budget of RFID tags. Although we significantly reduce the area and latency of SQUASH with our PSQUASH variant, the area consumption is slightly more than some notable symmetric ciphers; moreover, the latency of PSQUASH is much higher.

Adi Shamir suggests SQUASH is well positioned to take advantage of next generation RFID tags which may have ALUs because SQUASH is scalable to arbitrary word width. We tested SQUASH on a prototype CRFID. Our results show that while SQUASH benefits from ALUs in embedded microprocessors, the cost of NLFSR shifts in software negate the benefit of increased word width. A method to choose the optimal partial product ordering is introduced which greatly reduces NLFSR shifts. This reordering of partial products makes SQUASH comparable to software implementation of symmetric ciphers; however a software friendly NLFSR is needed for faster SQUASH on CRFID.

SQUASH is a novel idea but the large squaring operation leads to a large latency, and for this reason, it is unlikely to be deployed in many RFID applications. New lightweight symmetric ciphers have significantly lower latency, while consuming slightly less area: making them a better choice for constructing a challenge response MAC. These ciphers are relatively new, and have not been extensively studied as Rabin's cryptosystem, but they have received significant peer review. Still SQUASH has a potential to be used in RFID systems and its use has been proposed in [37, 38, 39]. For potential SQUASH applications, our contributions offer a significant reduction in area and/or latency.

BIBLIOGRAPHY

[1] Adi Shamir. SQUASH – A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags. FSE 2008, LNCS 5086, pp. 144–157, 2008.

[2] EPCGlobal http://www.epcglobalinc.org/

[3] RFID Journal, November 2003.

[4] Martin Feldhofer, Sandra Dominikus and Johannes Wolkerstorfer, Strong Authentication for RFID Systems Using the AES Algorithm, Proc. Workshop on Cryptographic Hardware and Embedded Systems - CHES 2004, Cambridge (Boston), Ma.,USA, Springer, vol. 3156, August 2004, pp. 357-370.

[5] Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann and Leif Uhsadel, A Survey of Lightweight-Cryptography Implementations, IEEE Design & Test of Computers, vol. 24, no. 6, November 2007, pp. 522-533.

[6] A. Bogdanov et al., PRESENT: An Ultra-Lightweight Block Cipher, Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES 07), LNCS 4727, Springer, 2007, pp. 450-466.

[7] eSTREAM project http://www.ecrypt.eu.org/stream/

[8] Hopper, N.J., Blum, M.: A Secure Human-Computer Authentication Scheme, CMU-CS-00-139 (2000)

[9] Juels, A., Weis, S.A.: Authenticating Pervasive Devices with Human Protocols. In Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 293–308. Springer, Heidelberg (2005)

[10] Bringer, J., Chabanne, H., Dottax, E.: HB++: a Lightweight Authentication Protocol Secure Against Some Attacks. In: Workshop on Security, Privacy and Trust in pervasive and Ubiquitous Computing - SecPerU (2006)

[11] Munilla, J., Peinado, A.: HB-MP: A further step in the HB-family of lightweight authentication protocols. Computer Networks 51, 2262–2267 (2007)

[12] Tassos Dimitriou. A lightweight RFID protocol to protect against traceability and cloning attacks. IEEE SECURECOMM, 2005.

[13] Tassos Dimitriou. A secure and efficient RFID protocol that can make big brother obsolete. International Conference on Pervasive Computing and Communications, PerCom, 2006.

[14] Chatmon, C., Le, T.v., and Burmester, M. (2006). Secure anonymous RFID authentication protocols. Technical Report TR-060112, Florida State University, Department of Computer Science, Tallahassee, Florida, USA, 2006.

[15] Francois Gosset, et al. FPGA Implementation of SQUASH. Twenty-ninth Symposium on Information Theory in the Benelux, May 2008.

[16] Rabin, M.O.: Digitalized Signatures and Public-Key Functions as Intractable as Factorization, MIT LCS/TR-212 (1979)

[17] K. Ouafi and S. Vaudenay. Smashing SQUASH-0. EUROCRYPT 09, volume 5479 of LNCS, pages 300-312. Springer, 2009.

[18] Hell, M., Johansson, T., Maximov, A., Meier, W. A Stream Cipher Proposal: Grain-128, http://www.it.lth.se/martin/Grain128.pdf

[19] P.G. Comba, Exponentiation on the IBM PC, IBM Systems Journal, vol. 29, no.4, pp. 526-538, 1990.

[20] Faraday UMC Free Library http://freelibrary.faraday-tech.com/

[21] OpenCores http://www.opencores.org/

[22] Benjamin Ransford, Shane Clark, Mastooreh Salajegheh, Kevin Fu, “Getting Things Done on Computational RFIDs with Energy-Aware Checkpointing and Voltage-Aware Scheduling.” In Proceedings of *USENIX Workshop on Power Aware Computing and Systems* (HotPower), December 2008.

[23] J. R. Smith, A. P. Sample, P. S. Powledge, S. Roy, and A. Mamishev. “A wirelessly-powered platform for sensing and computation." In *8th International Conference on Ubiquitous Computing* (Ubicomp 2006), pages 495–506, Orange Country, CA, USA, September 2006.

[24] IAR Embedded Workbench http://www.iar.com/ew, 2009.

[25] Alien UHF RFID readers http://www.alientechnology.com/readers/index.php

[26] Intel Research Seattle http://seattle.intel-research.net/

[27] MSP-FET430UIF http://focus.ti.com/docs/toolsw/folders/print/msp-fet430uif.html

[28] William George Horner. “A new method of solving numerical equations of all orders, by continuous approximation.” In *Philosophical Transactions of the Royal Society of London*, pp. 308–335, July 1819

[29] H.-J. Chae, D. J. Yeager, J. R. Smith, and K. Fu. Maximalist cryptography and computation on the WISP UHF RFID tag. In Proceedings of the Conference on RFID Security, July 2007.

[30] Morris Dworkin, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. NIST Special Publication
http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[31] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs." Boston, Massachusetts: 6th International Workshop on Cryptographic Hardware and Embedded Systems, August 2004.

[32] A. Karatsuba and Y. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. Doklady Akad. Nauk, (145):293{294, 1963. Translation in Physics-Doklady 7, 595-596.

[33] M. Feldhofer, C. Rechberger. A Case against Currently used Hash Functions in RFID Protocols. In On the Move to Meaningful Internet Systems 2006: OTM'06 Workshops, including the First International Workshop on Information Security (IS'06), Montpelier, France, Lecture Notes in Computer Science 4277, pp. 372–381, Springer-Verlag, 2006.

[34] Rolfes, C., Poschmann, A., Paar, C.: Security for 1000 Gate Equivalents. In: Secure Component and System Identification (SECSI) 2008 (2008)

[35] Meiser, G., Eisenbarth, T., Lemke-Rust, K., Paar, C.: Efficient implementation of estream ciphers on 8-bit avr microcontrollers. In Industrial Embedded systems, 2008. SIES 2008, pp. 58–66 (2008)

[36] M. Vogt, A. Poschmann, and C. Paar, "Cryptography is Feasible on 4-Bit Microcontrollers - A Proof of Concept", 2009 IEEE International Conference on RFID, pp. 241-248, 2009.

[37] Mastooreh Salajegheh, Shane Clark, Benjamin Ransford, Kevin Fu, and Ari Juels, CCCP: Secure Remote Storage for Computational RFIDs. In Proc. of the 18th USENIX Security Symposium, 2009

[38] Mohammad Shahriar Rahman, Masakazu Soshi, and Atsuko Miyaji, A Secure RFID Authentication Protocol with Low Communication Cost. International Conference on Complex, Intelligent and Software Intensive Systems, 2009.

[39] J. Wu and D. Stinson. How to Improve Security and Reduce Hardware Demands of the WIPR RFID Protocol. In IEEE International Conference on RFID. RFID 2009, Orlando, Florida, USA, April 2009.