SQUASH – A New MAC with Provable Security Properties for Highly Constrained Devices Such as RFID Tags

Adi Shamir

Computer Science department, The Weizmann Institute, Rehovot 76100, Israel
`Adi.Shamir@weizmann.ac.il`

Abstract. We describe a new function called *SQUASH* (which is short for *SQU*are-h*ASH*), which is ideally suited to challenge-response MAC applications in highly constrained devices such as RFID tags. It is exceptionally simple, requires no source of random bits, and can be efficiently implemented on processors with arbitrary word sizes. Unlike other ad-hoc proposals which have no security analysis, SQUASH is provably at least as secure as Rabin's public key encryption scheme in this application.

Keywords: Hash function, MAC, RFID, provable security, SQUASH.

1 Introduction

Passive RFID tags are very simple computational devices (costing a few cents each). They obtain their power from and communicate with a reader using a magnetic or electromagnetic field at a distance of several centimeters to several meters. They have many applications, including warehouse inventory control, supermarket checkout counters, public transportation passes, anti-counterfeiting tags for medicines, pet identification, secure passports, etc. They are already widely deployed, and many more applications are likely to be found in the near future.

The basic requirement in most of these applications is that a tag should be able to interactively authenticate itself securely to a reader. We assume that the tag contains some nonsecret identity I and some secret information S associated with it. When challenged by the reader, the tag sends I in the clear, and convinces the reader that it knows S, without enabling rogue eavesdroppers to extract S or to convince another reader that they know S when in fact they do not.

The classical solution for such a problem is to use zero knowledge interactive proofs, which prevent any leakage of information about S. However, such proofs are too complicated for RFID tags which have tiny memories and very limited computing power. In addition, in many applications the legitimate reader already knows the secret S, and thus we do not care about the potential leakage of information from the real prover to the real verifier. We can thus use the much simpler protocol of challenge-response authentication, in which the reader issues a random challenge R, and the tag responds with the value $H(S, R)$ where H

is some publicly known hash function. This value, which can be viewed as a *message authentication code* (MAC), is independently computed by the reader, which accepts the authentication if and only if the computed and received values are the same.

Most of the literature on the construction of MAC's (which deals with chaining and padding techniques for multiblock inputs) is irrelevant in our challenge-response application, since we always apply H to a single block input of fixed size. The main requirement from H is that it should protect the secrecy of S even after an eavesdropper or a rogue reader gets $H(S, R_i)$ for many (known or chosen) challenges R_i. In particular, it should make it difficult for the adversary to compute the correct response of the tag to a new random challenge which had not been seen before. The function H should thus be a one-way hash function, hiding all information about S, but not necessarily a collision-resistant hash function since the discovery of a collision is not a security threat in challenge-response authentication.

Unfortunately, standard hash functions (such as SHA-1) are primarily designed to be collision resistant in order to prevent forgery of digitally signed documents. This is a very difficult requirement, which adds a lot of unnecessary complexity to their design in our application, and makes them too complicated for RFID tags. This was recognized by the RFID research community, and over the last few years there was a major effort to develop dedicated one way hash functions which are not necessarily collision resistant, and which are more suitable for RFID applications.

The best known schemes of this type belong to the HB family of schemes originally proposed by Hopper and Blum in 2001, which now includes the schemes HB [6], HB+ [7], HB++ [3], and HB-MP [11]. The security of these schemes is based on the difficulty of solving the *parity with noise problem*, which is known to be NP-complete in general, and was investigated in several recent papers [4] [8]. These schemes are much simpler than SHA-1, but they suffer from several serious problems:

1. The tag needs an internal source of random bits. Real randomness is difficult to find and can be externally manipulated by the adversary, while pseudo-randomness requires a large nonvolatile memory and lots of computations.

2. Since the proof of authenticity in these schemes is probabilistic, there is a small chance that a tag will fail to convince a reader that it is valid even when both of them are honest and there are no adversaries.

3. There are several parameters involved (the length of the secret, the number of repetitions, the probability of the additive noise, etc) and there is considerable debate about which values of these parameters will make the scheme sufficiently secure.

4. Over the last few years, a large number of attacks were developed against most of these schemes, and the various members of the HB family were developed in response to these attacks. For example, HB is known to be insecure against active adversaries. HB+ was claimed to be secure against such adversaries, but it had been recently shown in [9] that it can be attacked by a man-in-the-middle

adversary who can modify the challenges and observe the reaction of the real reader to the modified responses. With each modification, the scheme became more complicated, requiring larger keys and more computations, and it is not clear that even the latest version is completely secure.

2 The New Approach

In this paper we introduce a new function called *SQUASH* (which is a squashed form of *SQU*are-h*ASH*), which is ideally suited to RFID-based challenge-response authentication. Unlike the HB schemes it is completely deterministic, and thus it does not need any internal source of randomness and there is no way in which a legitimate tag will fail to convince a legitimate reader that it is authentic. It is exceptionally simple, and yet it is provably at least as secure as the Rabin scheme (which had been extensively studied over the last 30 years) in this application.

The basic idea of SQUASH is to mimic the operation of the Rabin encryption scheme [12], in which a message m is encrypted under key n (where the publicly known modulus n is the product of at least two unknown prime factors) by computing the ciphertext $c = m^2 \ (mod\ n)$. This is an excellent one way function, but definitely not a collision resistant function, since m, $-m$, and $m+n$ all hash to the same c. To make the Rabin scheme secure, the binary length k of n must be at least 1000 bits long, the length of m should not be much smaller than k, and thus just to store n m and c we need at least 3000 bits. Clearly we can not perform a general modular squaring operation on a severely limited RFID tag which can store less than 300 bits. Our game plan in this paper is to use the Rabin scheme as a secure starting point, and to change it in multiple ways which make it much more efficient but with provably equivalent security properties.

There were several previous attempts to simplify the implementation of modular squaring on constrained devices. At Eurocrypt 1994 [13], I proposed to replace the modular squaring operation $m^2 \ (mod\ n)$ by a randomized squaring operation $m^2 + rn$ where r is a random number which is at least 100 bits longer than n. This scheme is provably as strong as the original Rabin scheme, and has the advantage that it can be computed with a very small memory since the successive bits of m^2 and rn can be computed on the fly from LSB to MSB. This scheme can be used in low end smart cards, but it requires a lot of time and power to compute all the bits of the output (which is twice as long as in the original Rabin scheme), and is not suitable for the weaker processors contained in RFID tag.

In any challenge-response application, the secret S (which is typically 64 to 128 bits long) and the challenge R (which is typically 32 to 64 bits long) should be securely mixed and extended into a message m in the same way that keys and IV's are mixed and extended into initial states in stream ciphers. In a Rabin-based scheme, the noninvertibility of the mapping should be primarily provided by the squaring operation, and we would like to use the simplest mixing function $M(S, R)$ which addresses the known weaknesses of modular squaring, such as its easy invertibility on small inputs, its multiplicativity, and its algebraic nature (which makes it easy, for example, to compute S from $(S + R_1)^2$ $(mod\ n)$ and $(S + R_2)^2$ $(mod\ n)$ when the challenge R is numerically added to the secret S).

Studying various choices of simple mixing functions $M(S, R)$ is likely to lead to many interesting attacks and countermeasures. For example, Serge Vaudenay (in a private communication) had already developed a very clever polynomial-time attack on the case in which the short mixed value $S \oplus R$ is expanded by a *linear* feedback shift register, and then squared modulo $n = pq$.

The best choice of M also leads to a delicate theoretical dilemma: if we make it too strong (e.g., use a provably secure pseudo-random function) there is no point in squaring its result, and if we make it too weak (e.g., use a constant function) we cannot prove the formal security of the combined construction. To address this difficulty, we proceed in the rest of this paper in two different directions.

In Section 3 we assume that the choice of M is not part of the generic SQUASH construction (just as the choice of hash function for long messages is not part of the generic RSA signature scheme). We prove a *relative security result* which shows that for any choice of M, the combination $SQUASH(M(S, R))$ is at least as secure as the combination $Rabin(M(S, R))$, even though SQUASH is much simpler and faster than Rabin. More formally, we claim:

Theorem 1. *Let $\psi(S)$ be any predicate of S, which can be computed with non-negligible advantage by using a known or chosen message attack on a MAC based on the mixing function M and the SQUASH function using modulus n. Then $\psi(S)$ can be computed with at least the same advantage by the same type of attack when SQUASH is replaced by the original Rabin function with the same modulus n.*

The security of the challenge-response authentication scheme can be viewed as a special case of this theorem, in which $\psi(S)$ is defined as the value of some bit in $H(S, R)$ for a new challenge R which had not been seen before by the attacker.

In this approach, it is the responsibility of each designer to pick a particular mixing function M that he would be happy with if it would be followed by the Rabin encryption scheme, and then we give him the assurance that he would not go wrong by combining the same M with SQUASH.

Since this approach makes it difficult to evaluate the precise security and efficiency of SQUASH and to compare it to other MAC's designed for RFID applications, we propose in Section 4 a particular choice of M. Since the combined scheme has no formal proof of security, we optimize it very aggressively but we still believe that in practice it provides a high level of security at very low cost. It uses the nonlinear part of GRAIN-128[5], which is a well studied stream cipher with an extremely small footprint. Our concrete proposal (which we call SQUASH-128) is even smaller than GRAIN-128, requiring only half the number of gates to implement both M and SQUASH.

3 The Generic SQUASH Proposal

We will now describe how to simplify and speed up the Rabin encryption scheme without affecting its well studied one-wayness. The basic idea of SQUASH is to compute an excellent numerical approximation for a short window of bits in the middle of the ciphertext produced by the Rabin encryption function which uses a modulus of a particular form. We will now describe how to gradually transform Rabin to SQUASH by a series of simple observations and modifications.

Our first observation is that in the challenge-response MAC application, no one has to invert the mapping in order to recover the plaintext from the ciphertext, since both the tag and the reader compute the hash function only in the forward direction. Since we do not need a trapdoor in this application, no participant in the protocol needs to know the factorization of n, and thus everyone can use the same universal modulus n as long as no one knows how to factor it.

Our second observation is that the Rabin scheme cannot be efficiently inverted (and many of its bits can be proven secure) for *any* modulus n with unknown factorization. If a universal n with unknown factorization can be compactly represented by a small number of bits, we can save a lot of storage on the RFID tag. In particular, we recommend using a composite Mersenne number of the form $n = 2^k - 1$, which can be stored very compactly since its binary representation is just a sequence of k 1's. Other recommended choices of n which have very compact representations, such as the Cunningham project numbers of the form $n = a * (b^c) \pm d$ for small values of a, b, c, and d, will be discussed later in the paper.

A lot of effort was devoted over the last decade to determine which Mersenne numbers are prime, and to factorize those Mersenne numbers which are composite. A table summarizing the current status of these efforts is maintained by Paul Leyland [10], and the most recent success in factorizing such numbers was the complete factorization of $2^{1039} - 1$ in 2007 by a large distributed computation [1]. Since such numbers are a little easier to factor (by the special number field sieve) than general numbers (which require the general number field sieve), we recommend using numbers of the form $n = 2^k - 1$ with $1200 < k < 1300$. The currently known factors of all the "interesting" numbers in this range are summarized below. For example, $2^{1279} - 1$ is a 386 digit prime number denoted by P386, whereas $2^{1201} - 1$ has four known prime factors which are relatively small, plus a 314 decimal digit cofactor denoted by C314 which is known to be composite but has no known factors.

```
1201: 57649.1967239.8510287.2830858618432184648159211485423. C314
1213: 327511. C360
1217: 1045741327. C358
1223: 2447.31799.439191833149903. P346
1229: 36871.46703.105431792806619161210 33. C339
1231: 531793.5684759.18207494497.6391907836312168 1207. C329
1237: C373
1249: 97423.52358081.2379005273.9345276045907272726364012481. C326
1259: 875965965904153. C365
1277: C385
1279: P386
```

```
1283: 4824675346114250541198242904214396192319. C347
1289: 15856636079.108817410937.827446666316953.9580889333063599
.16055826953448199975207. P314
1291: 998943080897.84051400422953302189544581841. C348
1297: 12097392013313.64873964199444497. C361
```

The most interesting number in this range (and the one we recommend as the universal modulus of SQUASH) is $n = 2^{1277} - 1$, which is a 385 digit composite number with a completely unknown factorization. Another number of this type is the slightly smaller $n = 2^{1237} - 1$. Both numbers are on the "most wanted" list of computational number theorists, and a lot of effort was devoted so far to their factorization, without any success. However, there is no guarantee that these numbers will remain unfactored forever, and thus we have to consider the potential impact of either a partial or a complete factorization of the recommended modulus. As will be shown later, SQUASH is surprisingly resilient to such future events: partial factorization of $n = 2^{1277} - 1$ will have no impact on the scheme or on its formal proof of security, and even full factorization of this n will only eliminate the formal proof of security but not necessarily the real security of SQUASH. In this sense, SQUASH is much better than the original Rabin scheme, whose security will be devastated by either a partial or a full factorization of its modulus.

Our third observation is that Mersenne moduli are not only easy to store, but they also make the computation of m^2 $(mod\, n = 2^k - 1)$ particularly simple: Since $2^k = 1$ $(mod\ n)$, we just compute the double sized m^2, and then numerically add the top half to the bottom half. More precisely, if $m^2 = m1 * 2^k + m2$, then $m^2 = m1 + m2$ $(mod\ n)$. Note that this sum could be bigger than n, creating a new wraparound carry of 1, but the effect of this carry will almost certainly be limited to a few LSB bits in the result.

Our fourth observation is that there is no need to send the full 1000+ bit ciphertext c in response to the challenge R. In general, when no information about the expected response c can be computed by the adversary, the probability that the reader will accept a random t-bit answer from an adversary is 2^{-t}. In most cases, a sufficiently secure authentication of an RFID tag will be achieved if it sends $t = 32$ bits (with a cheating probability of about one in 4 billion). Low security applications can even use $t = 16$, and high security applications can either use a larger t such as 64, or repeat a low security authentication procedure several times with different challenges. The tag can thus send only a small subset of t bits from c, and as we will see shortly, sending a window of consecutive bits makes the tag's computation particularly simple. Since arithmetic modulo $2^k - 1$ has cyclic symmetry (in which rotation of the bits is equivalent to multiplication by 2), the exact location of this window within c is not important, but for the sake of concreteness in the rest of this paper we place it close to the center of c. The crucial point is that the difficulty of computing some useful predicate of the secret S (such as computing one of the bits of its expected response to some new challenge R) is *monotonically decreasing* with t since any computational task can only become easier when more information is provided in the input. In particular, if we assume that it was difficult in the original Rabin scheme then it will certainly be difficult when only t out of the k bits from each Rabin ciphertext are made available by the tag to the adversary in each response.

Our fifth observation is that if we want to be sure that a particular bit we compute in m^2 is correct, we have to compute in the worst case all the earlier bits in order to be certain about the effect of the carry entering this bit position (addition carries propagate only from LSB to MSB, so we do not have to compute higher order bits in m^2). However, we can get an excellent numeric approximation of the carry into the t bits we would actually like to compute if we compute a longer window of $t+u$ bits with u additional low order bits (which we call *guard bits*), assuming that no carry entered into the LSB of this extended window, and providing only the top t out of the $t+u$ bits as an answer. For k between 1024 and 2048, it is easy to show that the carry into each bit position in the computation of m^2 can be at most 11 bits long, and thus if we add $u = 16$ guard bits to the computed window we have only a small probability of less than $2^{11}/2^{16} = 1/32$ of computing an incorrect carry into the 17-th bit we compute. If we add $u = 64$ guard bits, then this error probability becomes negligible. Note that we can easily determine when a mistake is possible (a necessary condition is that all the top $u - 11$ guard bits above the 11 LSB bits in the extended window are 1 so that the unknown carry can propagate through them). We can thus start the computation with a small u such as 16, and only in the small fraction of the cases in which all the $u - 11$ guard bits are 1, we can rerun the computation with a larger u such as 32 or 64. This can guarantee an extremely small error probability while keeping the average running time only slightly higher than always computing $t + 16$ bits.

With this relaxation, what we gain is the ability to compute the small number of relevant bits in m^2 in linear rather than quadratic time, which is in practice one to two orders of magnitude faster than a full computation of m^2. What we lose is that the value we produce is only an approximation of the real value produced by Rabin's encryption scheme, and thus it is conceivable that by using our protocol we will reveal more information about the secret S than by using Rabin's scheme. However, the two results differ only in a negligible fraction of executions, and thus neither the reader nor the adversary is ever expected to see an incorrectly computed answer, and thus the formal security proof (based on the assumption that the Rabin scheme is secure) remains unaffected.

Our sixth observation is that if the successive bits of $m = M(S, R)$ can be efficiently generated in both the forward and backward order, we can compute the successive bits in m^2 without storing the long m explicitly, by convolving these two streams of bits. When we want to compute bit j in the lower half of m^2, we compute it by summing all the products $m_v * m_{j-v}$ for $v = 0, 1, 2, ..., j$, and add to this sum the carry from the computation of the previous bit. When we want to compute bit $j + k$ in the upper half of m^2, we compute it by summing all the products $m_v * m_{j+k-v}$ for $v = j + 1, ..., k - 1$, and add to this sum the carry from the computation of the previous bit. When we want to compute m^2

$(mod\ n)$ for $n = 2^k - 1$, we want to sum the upper half and lower half of m^2, and thus the j-th bit c_j of $c = m^2\ (mod\ n)$ can be computed by adding bits j and $j+k$ in m^2, along with their carries. It is easy to verify that the sum of the two linear convolutions defining bits j and $j+k$ is exactly the circular convolution defined as the sum of all the products $m_v * m_{j-v(mod\ k)}$ for $v = 0, 1, 2, ..., k-1$. The final SQUASH algorithm is thus extremely simple:

1. Start with j which is the index at lower end of the desired extended window of $t+u$ bits, and set carry to 0.
2. Numerically add to the current carry (over the integers, not modulo 2) the k products of the form $m_v * m_{j-v(mod\ k)}$ for $v = 0, 1, 2, ..., k-1$.
3. Define bit c_j as the least significant bit of the carry, set the new carry to the current carry right-shifted by one bit position, and increment j by one.
4. Repeat steps 2 and 3 $t+u$ times, throw away the first u bits, and provide the last t bits as the response to the challenge.

To implement this algorithm, we can use a simple stream cipher such as a nonlinear feedback shift register (NFSR) with a reversible state transition function, initialize it with S and R, and run it back and forth to generate all the bits of m which are used in the convolution. This requires time proportional to $k^2 * (t+u)$ which is too high for $k = 1277$. A much faster implementation uses two copies of the stream cipher in order to compute the two sequences of bits we want to circularly convolve. However, whenever the state has to wrap around (e.g., to go from the first state to the last state) it has to do so in k clock cycles. The total running time is thus proportional to $2k(t+u)$. To save an extra factor of two in the running time, we can keep one additional state in an auxiliary register. We initially load both copies of the stream cipher with the initial state, clock the second copy to the desired middle state j, and load the auxiliary register with the last state $k-1$. We run the first copy upwards all the way from state 0 to state $k-1$, and the second copy downwards from state j. When it reaches state 0, we exchange its state with the auxiliary register, so that now the second copy will contain state $k-1$ and the auxiliary register will contain state 0. We continue to run the second copy downwards from state $k-1$ to state $j+1$. This completes the computation of the first c_j. We can now exchange the states of the first copy and the auxiliary register, and clock the second copy once, in order to bring all the components to the desired states for computing the next bit. Note that it is possible to exchange the values of two registers Y and Z without using additional storage by computing $Y = Y \oplus Z$, $Z = Y \oplus Z$, and $Y = Y \oplus Z$.

Note that due to the associativity of addition, we can compute the sum of products either upwards or downwards and get the same value, which makes it possible to run the algorithm in many different ways. For example, the first copy of the stream cipher can alternately run forwards and backwards through states $0, 1, ..., k-1$, the second copy can alternately run backwards and forwards in a cyclic order $(mod\ k)$, incrementing its state once after each round, and the auxiliary register can alternately keep states $k-1$ and 0 in order to help the

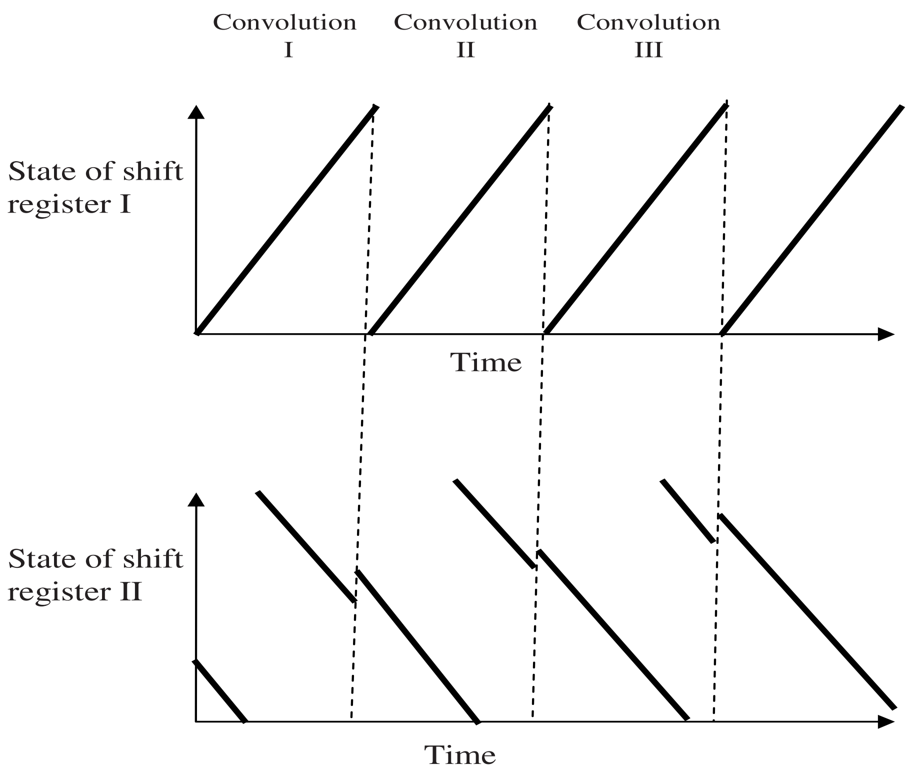

Fig. 1. The sequence of indices of the bits we have to multiply in the two streams to generate the successive output bits

second copy jump between these cyclically adjacent (but computationally wide apart) extreme states.

An important comment is that the SQUASH function is "one size fits all", and can be implemented efficiently on microprocessors with arbitrary word sizes. If the processor can multiply b-bit values in a single instruction, it can compute the same type of circular convolution b times faster by working with words rather than bits. Future RFID tags might contain simple 4-bit multipliers, which will speed up this algorithm by a factor of 4. In addition, the powerful microprocessors in the readers (which also have to carry out this computation to compare the expected and received responses) are likely to have 32-bit or even 64-bit multipliers, which will make the SQUASH algorithm extremely fast.

So far we described how to compute the SQUASH function when the underlying modulus is a composite Mersenne number of the form $2^k - 1$. It is very easy to modify the scheme to composite numbers of the form $2^k + 1$. The Rabin ciphertext in this case is defined by subtracting (instead of adding) the top half of m^2 from the bottom half. Consequently, when we compute the circular convolution we have to add to the carry all the products of the form $m_v * m_{j-v}$ for

$v = 0, 1, 2, ..., j$, and to subtract from the carry all the other products of the form $m_v * m_{j+k-v}$ for $v = j + 1, ..., k - 1$. Except for this minor change, everything else remains the same.

We can also consider more complicated moduli such as $n = a * 2^k - d$ where a and d are small positive integers. since $a * 2^k$ is congruent to d, we have to add to the bottom half of m^2 the top half divided by a and multiplied by d. To avoid the complicated division operation, we can change the definition of the output we are trying to compute to be a window of t consecutive bits in $a * m^2 \ (mod\ n)$. Note that the security of Rabin's encryption scheme cannot be changed if all its ciphertexts are multiplied by a known constant a, and thus we can not lose security by computing windows of bits in such modified Rabin ciphertexts instead of in the original ciphertexts. Since a multiplies both the top and the bottom parts of m^2, this implies that the algorithm now has to add to the carry all the products of the form $a * m_v * m_{j-v}$ for $v = 0, 1, 2, ..., j$, and then to add to the carry all the other products of the form $d * m_v * m_{j+k-v}$ for $v = j + 1, ..., k - 1$. If n is of the form $a * 2^n + d$, then the algorithm has to subtract (rather than add) from the carry all the products of the second type. When n is of the general form $a * b^c \pm d$ for small a, b, c and d, the algorithm can perform the same type of computations in base b instead of base 2, but this will probably make the scheme too complicated for a typical RFID.

Our seventh observation is that we can retain the formal proof of security even if n has some known small factors, provided that it has at least two large unknown factors. This can greatly extend the set of moduli which we can use, since most of the composite Mersenne numbers for $1000 < k < 2000$ have some small known factors. Consider, for example, the case of $n = 2^{1213} - 1$, which has a known prime factor of 327511 and a composite cofactor of 360 decimal digits whose factorization is completely unknown. If we use Rabin's encryption scheme with this n, the value of the ciphertexts modulo 327511 actually leaks the values of the plaintexts modulo this prime. We can completely stop this partial leakage of information by adding to each Rabin ciphertext a freshly selected random number between 0 and 327510, which randomizes the value of the ciphertexts modulo 327511. Since these added random values are small and we compute a window of bits near the middle of each Rabin ciphertext, we can *pretend* that such a randomizing value was indeed added to the ciphertext without changing anything in the definition of SQUASH - the only effect of such a randomization is that our numerical approximation of the middle windows in the Rabin ciphertexts will deteriorate in a negligible way[1]. An interesting corollary of this observation is that SQUASH will remain provably secure even if someone will *partially factorize* n in the future: Since we do not have to modify anything in the definition of SQUASH in order to use a modulus with a small known factor, we do not actually have to know its value when we use the scheme. Consequently, our formal proof of security will not be affected by a future discovery of some of the factors of the recommended modulus $n = 2^{1277} - 1$, provided that the

[1] This proof can be easily modified to deal with window locations which are closer to the low end of c.

factorization is partial and n has a sufficiently long cofactor whose factorization remains unknown.

Let us now assume that next year someone will find the *complete factorization* of $n = 2^{1277} - 1$. This will devastate the security of the Rabin encryption scheme which uses this modulus, since it will make it possible to decrypt all the previously produced ciphertexts. It will also eliminate the *formal proof of security* of SQUASH, but will not necessarily make it insecure in practice: Even when an attacker can extract arbitrary modular square roots *mod* n, it is not clear how to apply this operation when only a short window of bits in the middle of each Rabin ciphertext is available. In this sense, SQUASH is provably at least as secure as Rabin, but in practice it can be much more secure.

Our final observation deals with the relationship between SQUASH and some of the other proposed hash functions for RFID tags. The formal security of SQUASH is based on the difficulty of factoring the modulus n, but its implementation has the form of a cyclic convolution of a secret vector m with itself, which does not use n in an explicit way. It can thus be viewed as a scheme whose security is based on the difficulty of solving a system of quadratic equations of a very specific type. This is not entirely accurate, since the convolution is defined over the integers rather than over $GF(2)$, and the carries are defined by expressions with degrees higher than 2. In addition, the mixing function M can create complex dependencies between the bits of m. The QUAD scheme[2] is another attempt to construct a cryptographic primitive whose security is directly based on the NP-completeness of the general problem of solving systems of quadratic equations with k variables over $GF(2)$. However, the implementation complexity of QUAD is much higher than that of SQUASH since QUAD must use a dense system of quadratic equations with $O(k^2)$ randomly chosen coefficients per equation, whereas the convolution-based SQUASH has only $O(k)$ coefficients per equation defined in a very regular way. Consequently, SQUASH is much more suitable than QUAD for tiny RFID tags. Finally, HB+ also has the overall structure of convolving two vectors (S and R), but in this case R is known, and thus its security has to be based on the different problem of solving a large system of linear equations corrupted by noise.

These comparisons raise a number of interesting open problems about the security of other SQUASH-like functions. For example, SQUASH is typically implemented with two copies of the stream cipher M initialized with the same secret value and run in opposite directions. Can we initialize the two copies of M with different secret values? This can halve the number of state bits needed to get the same security against exhaustive search, but leads to bilinear rather than quadratic equations, and we have no formal argument which supports its security. Another modification is to run the two copies of M in the same direction rather than in opposite directions, and compute the dot product (with carries) of the generated sequence with various small shifts of itself. When M is implemented by a shift register, we can use only one copy of M, and get the $t + u$ shifted versions of its output by tapping various bits within this register. This can again halve the hardware complexity of the implementation, but there is no formal argument why the specific system of quadratic equations generated in such a way should be secure.

4 The Concrete SQUASH-128 Proposal

In this section we describe a fully specified MAC proposal, in order to make it possible to study its exact security and efficiency. It differs from the generic SQUASH construction in two important ways:

1. It uses a particular choice of mixing function $M_0(S, R)$, which is based on a single nonlinear feedback shift register. It shares this register with SQUASH, and thus the only additional hardware required (beyond the register itself) are a few gates to implement the feedback function and the carry adder.
2. Since the combined mapping $SQUASH(M_0(S, R))$ has no formal proof of security, we also simplify the SQUASH part in a very aggressive way by eliminating all the elements which were required by the security proof but which are not believed to contribute to the real security of the scheme. For example, we use only 8 guard bits instead of a variable number up to 64, which were needed only in order to claim that the windows of bits provided by SQUASH and Rabin are indistinguishable.

The most radical optimization step in our concrete proposal is to use a smaller modulus n. We can view the proof that SQUASH is at least as secure as Rabin as a *safety net* in order to show that the general structure of SQUASH can not be broken in polynomial time. This safety net is relatively weak (since the complexity of factoring is only subexponential in the size of n) and very erratic: it is applicable to $n = 2^{1277} - 1$ which has no known factors, but inapplicable to $n = 2^{1279} - 1$ which is a prime number. However, SQUASH seems to be much more secure than Rabin since there is no known attack on it even when the complete factorization of n is given. We believe that in fact the best attack on SQUASH requires exponential time and grows monotonically with the size of n, and thus we propose as a challenge to the reader to try to break an extremely reduced version of SQUASH which uses $n = 2^{128} - 1$ as the universal modulus, even though it is very easy to factor. We call this version SQUASH-128, and emphasize that its successful cryptanalysis will just indicate that we were too aggressive in our optimizations. The relationship between SQUASH-128 and the generic SQUASH construction is similar to the relationship between DES and the Luby-Rackoff theory of Feistel structures upon which it is loosely based.

The reduction in the size of n increases the speed of the scheme by a factor of 10, and makes it possible to halve its footprint: Since m is short, we can generate it with a single copy of M (instead of two copies which operate in opposite directions), store it in a single 128-bit register, and perform the convolutions directly on this register.

Our concrete proposal of SQUASH-128 uses a 64-bit key S and a 64-bit challenge R, and produces a 32-bit response. Our choice of M_0 is the nonlinear half of GRAIN-128 (we do not need the linear half since in this application we do not need any guaranteed lower bound on the cycle length of the generated sequence). It initializes a single 128-bit shift register denoted by $(b_0, \ldots, b_{127})$ by storing S in its low half and $S \oplus R$ in its high half. It mixes them by clocking the register 512 times (this is twice the number of initialization steps in GRAIN-128, which is still small compared to the time required by the convolutions), using no inputs and producing no outputs. The resultant 128-bit state is the value m which is squared modulo $2^{128} - 1$. The 32-bit response consists of bits 48 to 79 in the cyclically convolved result, using the 8 bits at positions 40 to 47 as guard bits. The clocking of the shift register uses the following nonlinear feedback function:

$$b_{i+128} = b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} + b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + b_{i+68}b_{i+84}$$

This function is the sum modulo 2 of a linear function and a quadratic bent function. It has the nice property that it can be applied up to 32 times faster by duplicating the feedback function and running these copies in parallel. Note that the zero state is a fixedpoint, and thus $S = 0$ should be excluded as a weak key.

Our choice of M_0 shares the same 128-bit shift register with SQUASH, and the only additional gates needed are an AND gate and an 8-bit carry register for the convolutions, a few AND and XOR gates for the feedback function, and two 7-bit counters for the indices v and j. Consequently, we expect the total number of gates needed by the complete SQUASH-128 scheme to be about half the number of gates in GRAIN-128, which is itself one of the smallest hardware-oriented cryptographic primitives.

This completes the description of SQUASH-128, and we encourage the reader to try to break the security of this scheme with a chosen challenge attack which requires less than 2^{64} time and space. As pointed out by Henri Gilbert and Helena Handschuh (in a private communication), this is the highest possible security level for any MAC which has a 128-bit internal state.

References

1. Aoki, K., Franke, J., Kleinjung, T., Lenstra, A.K., Osvik, D.A.: Research announcement, `http://actualites.epfl.ch/presseinfo-com?id=441`
2. Berbain, C., Gilbert, H., Patarin, J.: QUAD: A Practical Stream Cipher with Provable Security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 109–128. Springer, Heidelberg (2006)
3. 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)
4. Fossorier, M.P.C., Mihaljević, M.J., Imai, H., Cui, Y., Matsuura, K.: An Algorithm for Solving the LPN Problem and Its Application to Security Evaluation of the HB Protocols for RFID Authentication. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 48–62. Springer, Heidelberg (2006)
5. Hell, M., Johansson, T., Maximov, A., Meier, W.: A Stream Cipher Proposal: Grain-128, `http://www.it.lth.se/martin/Grain128.pdf`

6. Hopper, N.J., Blum, M.: A Secure Human-Computer Authentication Scheme, CMU-CS-00-139 (2000)
7. 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)
8. Levieil, E., Fouque, P.-A.: An Improved LPN Algorithm, Security and Cryptography for Networks (2006)
9. Gilbert, H., Robshaw, M., Silbert, H.: An active attack against HB+ – a provable secure lightweight authentication protocol, Cryptology ePrint Archive number 2005/237
10. Leyland, P., `http://www.leyland.vispa.com/numth/factorization/cunningham/2-.txt`
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. Rabin, M.O.: Digitalized Signatures and Public-Key Functions as Intractable as Factorization, MIT LCS/TR-212 (1979)
13. Shamir, A.: Memory Efficient Variants of Public-Key Schemes for Smart Card Applications. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 445–449. Springer, Heidelberg (1995)