Never trust a bunny*

Daniel J. Bernstein[1] and Tanja Lange[2]

[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7053, USA
djb@cr.yp.to
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands
tanja@hyperelliptic.org

Abstract. "Lapin" is a new RFID authentication protocol proposed at FSE 2012. "Ring-LPN" (Ring-Learning-Parity-with-Noise) is a new computational problem proposed in the same paper; there is a proof relating the security of Lapin to the difficulty of Ring-LPN. This paper presents an attack against Ring-LPN-512 and Lapin-512. The attack is not practical but nevertheless violates specific security claims in the FSE 2012 paper.

Keywords: Authentication, RFID, LPN, Ring-LPN, attacks, Lapin, bunnies.

1 Introduction

Lapin [15] is a lightweight "RFID authentication" system introduced at FSE 2012 a week before the submission of this paper. The system is claimed in [15, Section 1.1] to be "provably secure against active attacks". This claim is qualified elsewhere in [15]: attacking Lapin-n is provably as difficult as attacking an n-bit "Ring-LPN" problem introduced in the same paper [15].

This proof begs the question of how secure Ring-LPN-n is. Ring-LPN-512 is claimed in [15, Section 3.1] to "require 2^{77} memory (and thus at least that much time) to solve when given access to approximately as many samples" according to the "analysis" of [22].

A closer look shows that the attack in [22] begins with nearly 2^{76} 513-bit vectors consuming nearly 2^{82} bytes of memory, performs a series of 6 sorting steps to compute 2^{63} reduced 75-bit vectors, and concludes with a similarly expensive Walsh transform. Note that [22] is an analysis of *some* attacks; there is no proof that better attacks do not exist against Ring-LPN-512, or against Lapin-512.

* The Wolf: "I knew it! Never trust a bunny!" Twitchy: "Never trust a bunny!" [9]

** This work was supported by the National Science Foundation under grant 1018836, and by the European Commission under Contract ICT-2007-216676 ECRYPT II. Permanent ID of this document: 78fc70fd7aa0cbd24ffe73a426567577. Date: 2012.06.21.

Contributions of this paper. We present an attack that discovers the Ring-LPN-512 secret using $<2^{56}$ bytes of memory, $<2^{38}$ queries, and $<2^{98}$ easily vectorized bit operations. Running the attack twice discovers the Lapin-512 secret, allowing the attacker to clone a Lapin-512 RFID tag.

We do not claim that this Ring-LPN-512 attack is feasible. However, it is obviously much closer to feasibility than the attack in [22], and in particular breaks solidly through the "2^{77} memory" barrier claimed in [15] while remaining competitive in bit operations. It breaks even more solidly through the "approximately $[2^{77}]$ samples" barrier claimed in [15].

The concrete proposal in [15] is actually Lapin-532 rather than Lapin-512. Our attack scales naturally to any Ring-LPN-n, and allows many different tradeoffs between memory, queries, and bit operations.

Varying the error fraction. LPN, Ring-LPN, and Lapin have an implicit "error fraction" τ. The specific Lapin-n protocols and Ring-LPN-n problems discussed above have $\tau = 1/8$, but smaller values and larger values have appeared in other proposals: for example, the "reducible" Lapin variant in [15, Section 5.1] has $\tau = 1/6$, while [22, Section 5.2] recommends $\tau = 1/20$. We state and analyze our attack with τ as a variable.

It is stated in [19] that "the best algorithms [attacking LPN] take time $2^{\ell/\log \ell}$" for any "constant" $\tau > 0$. In Section 4 we consider the case $n = 1024$, $\tau = 1/20$, and give an attack on Ring-LPN taking far fewer than $2^{1024/\lg 1024} = 2^{102.4}$ bit operations, while using $<2^{21}$ bytes of memory and $<2^{64}$ queries. A variant of the attack uses more bit operations, but still fewer than $2^{102.4}$ bit operations, $<2^{21}$ bytes of memory, and just 10 Ring-LPN queries. This variant also works for LPN, using just 5120 queries.

We emphasize that τ plays an important role in the cost of these attacks, and that security estimates must take τ into account. Note that increasing τ slows down protocols.

Previous work. HB [16], HB$^+$ [18], HB^{++} [6], HB-MP [23], HB* [8], Trusted-HB [5], and HB$^\sharp$ [13] were each broken within one year of being proposed. See, e.g., [12].

Each of these RFID authentication protocols claimed security on the basis of the alleged difficulty of various LPN problems. The protocols were broken in two different ways. First, in many cases the protocol structure allowed attacks that were simpler than breaking those LPN problems. Second, in many cases the LPN problems turned out to be weaker than claimed.

Our attack follows the second line of work. We combine and refine LPN attack ideas previously published by Blum, Kalai, Wasserman, Levieil, Fouque, and Kirchner. See [3], [22], and [20].

We note that some of the previous attacks have the advantage of provability. Our algorithm — like the number-field sieve for attacking RSA, the rho method for attacking ECC, etc. — is only analyzed heuristically.

Notes on low-memory LPN attacks. It is often claimed that memory is the main bottleneck in breaking LPN: consider the "require 2^{77} memory" quote

above, or the quote "It takes 2^{46} bytes of memory to solve a LPN problem with $k = 256$ [and $\tau = 1/8$]" from [22, Section 5.2].

This claim is obviously violated by *extremely slow* attacks: a glance at the definition of LPN shows that a brute-force search for the LPN secret takes negligible memory. What is less obvious is that the claim is also violated by *fast* attacks, although perhaps not quite the fastest possible attacks. This was already visible from Kirchner's paper [20] (and not from earlier papers such as [22]); we introduce an improvement in the Walsh-transform step, but the critical LPN attack idea is due to Kirchner.

Notes on LPN vs. Ring LPN. [15, Section 3.1] says that the most efficient known attacks against "irreducible" cases of (q-query) Ring-LPN are simply attacks against (nq-query) LPN. However, one of the major steps in our Ring-LPN attack saves some time by taking advantage of the ring structure of Ring-LPN, suggesting that Ring-LPN is not as hard as LPN. This savings applies to both the "reducible" and "irreducible" cases; it is lost only when the number of queries is very small.

Consequences for RFID security. We do not mean to suggest that LPN and Ring-LPN are incapable of reaching a 2^{80} security level. LPN has always been able to dodge attacks by increasing its parameters, the same way that RSA has always been able to dodge improved factorization algorithms. However, we are skeptical of the idea that LPN, Ring-LPN, HB, Lapin, etc. can provide the same security as a block cipher within the same RFID cost constraints.

Lapin-532 is advertised in [15, Keywords] as being suitable for "RFID authentication" and in [15, Abstract] as having "10 times smaller code size" than AES. Code size is claimed in [15, Section 5] to be "the most precious resource once the run-time constraints are fulfilled ... For instance, the WISP, a computational RFID tag, has only 8 kBytes of program memory."

This comparison is puzzling for several reasons. First, the AES implementation cited in [15] is clearly very far from optimal. Second, Lapin-532 was designed for 2^{80} security, while AES was designed for 2^{128} security. Third, AES is certainly not the state of the art in lightweight block-cipher design. Fourth, Lapin-532 obviously needs much more RAM than a block cipher, and bytes of RAM are inherently more expensive than bytes of ROM; note that the MSP430F2132 in the current WISP4.1 has only 512 bytes of RAM.

What [15] actually reports for Lapin-532 is 195000 cycles using 459 bytes of code on an AVR ATmega163 smart-card CPU. For comparison, [10] reports

- 128-bit AES: 4600 AVR cycles, 1659 bytes of code, 32 bytes of RAM;
- 128-bit NOEKEON: 23500 AVR cycles, 364 bytes of code, 32 bytes of RAM;
- 128-bit HIGHT: 19500 AVR cycles, 402 bytes of code, 32 bytes of RAM;

and so on. Lapin uses more code than (e.g.) NOEKEON, uses more RAM, uses more cycles, uses more communication, and provides much less security against known attacks. The cost evaluation in [15, Section 5.3] also omits the cost of generating random bits, stating that this cost is "independent of the underlying cryptographic functions"; but cipher-based protocols actually require far fewer

random bits than Lapin. There is also no evidence that switching platforms (from AVR to MSP430 or to an ASIC) would make Lapin competitive.

Notes on provable security. The literature describes several ways for an RFID tag to securely authenticate itself using a secret AES key shared with the RFID reader; see, e.g., [11]. Attacking any of these AES-based authentication protocols is provably as difficult as attacking the standard "PRP" security notion for AES: distinguishing AES (with a secret key) from a uniform random permutation. Of course, AES can also be replaced with other block ciphers, as discussed above.

We see three reasons that the security proofs for these AES-based authentication systems are more satisfactory than the security proof for Lapin. First, the Lapin proof is limited to "active" attacks (obtaining information from the tag and then attempting to fool a reader) while the AES proofs cover a wider class of "man-in-the-middle" attacks (interacting with the tag and the reader in parallel); the "man-in-the-middle" attack in [15, Appendix] suggests that Lapin is actually less secure than Ring-LPN. Second, the AES proofs are based on a problem that seems difficult after extensive cryptanalysis, namely the AES PRP-distinguishing problem, while the Lapin proof is based on the obviously much less mature Ring-LPN problem. Third, even if there is no further progress in attacks, the quantitative security-performance tradeoff will be considerably worse for Ring-LPN than for AES.

We are puzzled by the claims in [15] that Lapin is "provably secure" while AES-based protocols are " 'merely' computationally secure with respect to known attacks". There are certainly reasons to avoid describing the AES-based protocols as "provably secure", but all of the same reasons also apply to Lapin. The AES proofs begin with an unproven hypothesis, namely that the AES PRP-distinguishing problem is difficult, but the Lapin proofs also begin with an unproven hypothesis, namely that the Ring-LPN problem is difficult. There could be better attacks against the AES PRP-distinguishing problem and the AES-based protocols, but there also could be better attacks against the Ring-LPN problem and the Lapin protocol. We are unaware of any reason to prefer Ring-LPN over AES as a foundation for security.

Acknowledgments. Thanks to the anonymous referees for their comments. We acknowledge one referee in particular for the random-bits comment above.

2 LPN, Ring-LPN, and Lapin

The learning parity with noise (LPN) problem was introduced as a basis of cryptographic authentication schemes by Hopper and Blum in [16]. Here is the basic protocol they describe: The parameters are $m, n \in \mathbf{Z}$ and $\tau, \tau' \in \mathbf{Q}$ with $0 < \tau \leq \tau' < 1/2$. The authentication tag and the reader share an n-bit secret $s \in \mathbf{F}_2^n$. To authenticate the tag to the reader the reader sends an n-bit challenge c to the tag; the tag computes the dot product $y = c \cdot s$, puts $e = 0$ with probability $1 - \tau$ and $e = 1$ with probability τ, and responds with $t = y + e$. The reader computes $e = t + c \cdot s$. The tag is accepted if after m repetitions of

the basic protocol the number of repetitions with $e \neq 0$ is no larger than $\tau' m$. Obviously one can merge the m executions into a single step by changing the challenge c to an $m \times n$ matrix C and requesting the tag to compute $Cs + e$ for some error vector $e \in \mathbf{F}_2^m$ in which each bit of e is set with probability τ.

LPN refers to learning the secret s under the noise e; the distinguishing version is given a sequence of pairs (c, b) and should distinguish whether it comes from $(c, c \cdot s + e)$ or (c, b) for random b. When HB was proposed, Håstad had shown in [14] that LPN is NP-hard and the best known attack at that time, due to Blum, Kalai and Wasserman [2], took $2^{\Omega(n/\log n)}$ challenge-response pairs and time $2^{\Omega(n/\log n)}$. However, in the above protocol an active attacker impersonating the reader can determine s bit-by-bit by setting c to be the ith unit vector several times until bit i of s is known with sufficiently high probability. Similarly, many subsequent HB variants have been broken because of the way that they use LPN, rather than because LPN is much easier than thought.

The past 10 years have seen various changes to the HB protocol which differ in the number of passes and in how the tags introduce extra randomness to avoid the basic attack; also some progress on solving the computational LPN problem has been made. At FSE 2012, Heyse et al. [15] presented a 2-pass authentication protocol based on the Eurocrypt 2011 2-pass protocol by Kiltz et al. [19] and some modifications for more efficient implementation. We first describe a matrix variant of their protocol and then state their changes to reduce communication and computation complexity.

The tag and the reader share 2 secret vectors $s, s' \in \mathbf{F}_2^n$. For each run of the protocol, the reader generates an $n \times n$ challenge matrix C, and the tag generates a random $n \times n$ matrix R. Challenged with C, the tag replies with $(R, R(Cs + s') + e)$. The reader deduces e and accepts the tag as authentic if R is invertible and the Hamming weight of e is smaller than $n\tau'$.

The scheme described so far is based on the LPN problem. The authors of [15] introduce a new problem called the "Ring-LPN" problem. Take the ring $\mathbf{F}_2[x]/f$ for some polynomial f of degree n and embed n-bit vectors $(s_0, s_1, \ldots, s_{n-1})$ as ring elements $\sum s_i x^i$. The Ring-LPN problem is to reconstruct s given pairs $(r, rs + e)$, where the computations take place modulo f and the bits of e are set with probability τ. In the decision version the distribution of $(r, rs + e)$ should be distinguished from (r, y) for random y's in $\mathbf{F}_2[x]/f$.

Using a ring has the advantage that the product of two vectors, interpreted as a ring element, gives another vector. Hence the $n \times n$ matrices C and R can be replaced by ring elements c and r, reducing the communication cost from $2n^2 + n$ to $3n$ and reducing the need for random bits. Furthermore, the ring operations can be implemented more efficiently. To reduce the communication complexity further, c is derived from a shorter bit string by some deterministic function $\pi : \{0,1\}^\lambda \to \mathbf{F}_2[x]/f$. In the Lapin-$n$ scheme the answers from the tag have the form $(r, z) = (r, r(s\pi(c) + s') + e)$. The reader verifies that r is invertible and that $z + r(s\pi(c) + s')$ has Hamming weight less than $n\tau'$.

To relate the Ring-LPN version to the LPN version note that R can be obtained from r as the matrix which for $0 \leq i \leq n - 1$ has the coefficients of

$rx^i \bmod f$ in column i, starting with the coefficient of x^0. This way the embedding of Rs into the ring gives $rs \bmod f$.

For the attacks we note that the reader controls c and thereby $\pi(c)$. If an attacker can solve the Ring-LPN problem, i.e. find s from observing pairs $(r_j, r_j s + e_j)$, then he can break the Lapin scheme by running the attack twice, once with c_1 and once with c_2 (for which $(\pi(c_1) + \pi(c_2))$ is invertible): The first attack reveals $s_1 = s\pi(c_1) + s'$ and the second one $s_2 = s\pi(c_2) + s'$ which give $s = (s_1 + s_2)/(\pi(c_1) + \pi(c_2))$ and $s' = (s_1\pi(c_2) + s_2\pi(c_1))/(\pi(c_1) + \pi(c_2))$.

The authors of [15] present 2 versions of Lapin-n which differ in whether or not f is irreducible; the attacks presented in the following section are independent of the properties of f, so we skip the review of these considerations.

3 The attack

This section states an attack against Ring-LPN. The attack has several parameters introduced in this section and optimized in the next section.

Initial queries. Query the Ring-LPN oracle repeatedly, obtaining a sequence $(r_1, r_1 s + e_1), (r_2, r_2 s + e_2), \ldots, (r_q, r_q s + e_q)$. Here q, the number of queries, is an attack parameter. As in the previous section, s is the oracle's secret; each r_j is a uniform random ring element; each bit of each e_j is set with probability τ; and all of the random choices made here are independent. Define $\delta = 1 - 2\tau$.

Targeting e_i. Choose $i \in \{1, \ldots, q\}$. The remaining steps of the attack hope that r_i is an invertible ring element (which is overwhelmingly likely) and that e_i does not have many bits set (which has a noticeable chance, depending on various parameters).

Compute $1/r_i$ in the ring; if this fails, stop. For each $j \neq i$, compute r_j/r_i and then $(r_j/r_i)(r_i s + e_i) + (r_j s + e_j) = (r_j/r_i)e_i + e_j$.

This computation transforms the original Ring-LPN oracle for s into a Ring-LPN oracle for e_i: the sequence of pairs $(r_j/r_i, (r_j/r_i)e_i + e_j)$ has exactly the same distribution as a sequence of Ring-LPN outputs for e_i. This transformation is useful because searching through the likely possibilities for e_i is much faster than searching through the possibilities for s, especially when τ is small.

The idea of this transformation was introduced by Kirchner in [20, Section 4.3.2] in the context of LPN. Starting from LPN outputs $(r_1, r_1 \cdot s + e_1), \ldots, (r_q, r_q \cdot s + e_q)$, Kirchner selects a target set T of n indices i, hoping that the n rows r_i for $i \in T$ form an invertible $n \times n$ matrix (probability about 30%) and that the n noise bits e_i for $i \in T$ do not have many bits set. Kirchner then applies the inverse of the matrix to replace the LPN oracle for s with an LPN oracle for these n bits e_i.

Our Ring-LPN variant is simpler and, more importantly, faster: ring multiplication is faster than matrix multiplication. We comment, however, that for small q the LPN transformation has an interesting feature not pointed out in [20]: it provides tremendous flexibility in the choice of n rows to combine into an invertible matrix. Exploiting the Ring-LPN structure means that our only

potential targets are $e_1, e_2, \ldots, e_q$, whereas the analogous LPN transformation is free to target any set of n positions. See Section 4 for an example.

Forgetting the ring structure. For each $j \neq i$, and for each $k \in \{0, \ldots, n-1\}$, define $v_{j,k} \in \mathbf{F}_2^{\{0,1,\ldots,n-1\}}$ as the vector whose ℓth bit is the coefficient of x^k in the ring element $(r_j/r_i)x^\ell$. Then the coefficient of x^k in the known quantity $(r_j/r_i)e_i + e_j$ is exactly $v_{j,k} \cdot e_i + e_{j,k}$, where $e_{j,k}$ means the coefficient of x^k in e_j. We now have $(q-1)n$ vectors $v_{j,k}$ and noisy dot products $v_{j,k} \cdot e_i + e_{j,k}$.

Clearing bits. Build a table of 2^b vectors as follows, where b is another algorithm parameter. For each of the above $(q-1)n$ vectors $v_{j,k}$ in turn, use the first b bits of $v_{j,k}$ as a table index, and store $v_{j,k}$, together with its noisy dot product, at the corresponding table position. If there is already a vector at that position in the table, do not overwrite that vector; instead xor it into $v_{j,k}$, and xor its noisy dot product into $v_{j,k} \cdot e_i + e_{j,k}$.

Now discard all the vectors in the table. There are at least $(q-1)n-2^b$ vectors *not* in the table, and each of them now has its first b bits all set to 0. Each noisy dot product has become more noisy: if two bits $e_{j,k}$ are each set independently with probability $(1-\delta)/2$ then their xor is set with probability $(1-\delta^2)/2$.

The table requires $2^b(n-b+2)$ bits of storage: each entry has n bits, minus b bits implicit from the table position, plus 1 bit for a noisy dot product, plus 1 bit to say whether the table entry is used. The list of vectors *not* in the table overwrites the original list of vectors without consuming any extra space.

We comment that, unless $(q-1)n$ is much larger than 2^b, one expects some table entries to be unused, so the $(q-1)n - 2^b$ bound is somewhat pessimistic. We also repeat a comment from Levieil and Fouque [22, Section 4]: starting from $t \approx (q-1)n/2^b$ different vectors sharing their first b bits, one can build $t(t-1)/2$ differences having their first b bits clear, rather than just $t-1$ differences with a single vector; this expands the pool of vectors beyond the original number of samples. For simplicity we avoid this option.

Clearing more bits. Repeat the above table-building process a total of a times, where a is another algorithm parameter. This produces a sequence of at least $(q-1)n-2^b a$ vectors v, each having its first ab bits clear and each accompanied by a noisy dot product $v \cdot e_i + \cdots$, where the noise is set with probability $(1-\delta^{2^a})/2$.

(In the extreme case $a = 0$, these vectors are the original $(q-1)n$ vectors with noise probability $(1-\delta)/2$. No storage is required for the table in this case, and the vectors can be compressed to the ring elements r_j/r_i.)

This bit-clearing procedure, without the initial transformation from s to e_i, was introduced by Blum, Kalai, and Wasserman in [3]. Blum, Kalai, and Wasserman cleared $n-1$ bits, obtaining noisy dot products $(0, \ldots, 0, 0) \cdot s + \cdots$ and $(0, \ldots, 0, 1) \cdot s + \cdots$, and then computed the last bit of s by a majority vote of the $(0, \ldots, 0, 1) \cdot s + \cdots$ dot products.

Guessing the remaining bits of e_i. The algorithm hopes at this point that e_i has Hamming weight $\leq W$ in its final $n - ab - \ell$ positions, where W and ℓ are two more algorithm parameters.

The first ab bits of e_i are not relevant to the noisy dot products $v \cdot e_i$ for the vectors v constructed above. We thus consider possible patterns for the remaining bits of e_i, i.e., patterns of n bits that have Hamming weight 0 in their first ab positions and Hamming weight $\leq W$ in their final $n - ab - \ell$ positions. There are exactly $2^\ell \sum_{w \leq W} \binom{n-ab-\ell}{w}$ such patterns.

For each pattern p, compute the number of vectors v such that the noisy dot product $v \cdot e_i + \cdots$ matches $v \cdot p$. If this number is large enough (see the next subsection) then p is overwhelmingly likely to match the final $n - ab$ positions of e_i. If this number is not large enough for any choice of p then the algorithm stops.

Carrying out this computation separately for each pattern would require counting a total of $V 2^\ell \sum_{w \leq W} \binom{n-ab-\ell}{w}$ dot products, where $V \geq (q-1)n - 2^b a$ is the number of vectors v. For $W \geq 2$ it is much better to share work between similar patterns. The idea is simple: if p, p' differ in only one bit, say $p_b \neq p'_b$, then the number of v with $v \cdot p = v \cdot e_i + \cdots$ is the sum of

- the number of v with $v \cdot p = v \cdot e_i + \cdots$ and $v_b = 1$ and
- the number of v with $v \cdot p = v \cdot e_i + \cdots$ and $v_b = 0$,

while the number of v with $v \cdot p' = v \cdot e_i + \cdots$ is the sum of

- the number of v with $v \cdot p \neq v \cdot e_i + \cdots$ and $v_b = 1$ (the complement of the first summand above) and
- the number of v with $v \cdot p = v \cdot e_i + \cdots$ and $v_b = 0$ (the same as the second summand above).

Repeating the same split ℓ times obtains 2^ℓ patterns as a "fast Walsh transform" of 2^ℓ summands, each involving $V/2^\ell$ vectors on average. The summands require counting only $V \sum_{w \leq W} \binom{n-ab-\ell}{w}$ dot products.

Levieil and Fouque in [22] used the extreme case $\ell = n - ab$ to search through all possibilities for $n - ab$ bits of s. Kirchner in [20] used smaller ℓ, and $W < n - ab - \ell$, but required e_i to have Hamming weight $\leq W$ in its final $n - ab$ positions; our variant has higher success probability.

Statistics: filtering out the noise. If $e_i = p$ then $v \cdot e_i + \cdots$ matches $v \cdot p$ exactly when the noise $\cdots$ equals 0. Recall that this occurs with probability $(1 + \delta^{2^a})/2$. In a pool of V vectors one expects an average of $V(1 + \delta^{2^a})/2$ matches, with a standard deviation proportional to $\sqrt{V}$.

If the final $n - ab$ bits of e_i do not match p then presumably $v \cdot e_i + \cdots + v \cdot p = v \cdot (e_i + p) + \cdots$ is set with probability $1/2$. (We do not claim that this analysis is provable.) In a pool of V vectors one then expects an average of $V/2$ matches, again with a standard deviation proportional to $\sqrt{V}$.

If the difference of averages, namely $V\delta^{2^a}/2$, is an order of magnitude larger than $\sqrt{V}$ then these two distributions have negligible overlap. The simplest strategy here is to define "large enough" as, e.g., $V/2 + 10\lceil\sqrt{V}\rceil$ (and require $\delta^{2^a}/2 \geq 20\lceil\sqrt{V}\rceil$) so that there is negligible chance of ever encountering a false positive; asymptotically 10 should be replaced by something growing (sublogarithmically) with the number of tests. A more complicated strategy is to define

"large enough" as, e.g., $V/2 + 2\lceil\sqrt{V}\rceil$, and to put more work into analyzing the occasional false positives.

Finishing up. At this point the final $n-ab$ positions of e_i are known. Eliminate those positions from the original vectors $v_{j,k}$, obtaining a new problem with n replaced by ab, and solve that problem recursively.

4 Analysis and optimization

This section analyzes the attack of the previous section, and gives several examples of reasonable parameter choices.

Success probability and cost. The chance that r_i is invertible depends on the selected ring but is indistinguishable from 1 for all Lapin proposals. The chance that e_i has Hamming weight $\leq W$ in its final $n - ab - \ell$ positions is exactly $\sum_{w \leq W} \binom{n-ab-\ell}{w} \tau^w (1-\tau)^{n-ab-\ell-w}$: for any particular weight w there are $\binom{n-ab-\ell}{w}$ choices of positions for w bits, chance τ of each of those bits being set, and chance $1-\tau$ of each of the other bits being clear. If $\sqrt{V}$ is sufficiently large compared to $2/\delta^{2^a}$ then any such e_i will in fact be recognized with overwhelming probability.

Each multiplication of r_j by $1/r_i$ costs $\leq 4n^2$ bit operations by standard techniques (and asymptotically $n^{1+o(1)}$ bit operations); the exact cost depends on the selected ring. Similar comments apply to the multiplication of r_j/r_i by $r_i s + e_i$, and to the computation of $v_{j,k}$, a "dual" multiplication. The initial computation of $1/r_i$ is easily amortized into $3 + o(1)$ multiplications. Overall there are $3q$ multiplications involved in computing the vectors $v_{j,k}$ and $\leq 2q$ additions, for a total of $\leq 12q(n^2 + n)$ bit operations.

Clearing bits involves at most $a(q-1)n^2$ bit xors. One of the factors n here is unnecessarily pessimistic: the first clearing xors only $n - b + 1$ bits, the second clearing xors only $n - 2b + 1$ bits, etc.

The dot products count at most $V \sum_{w \leq W} \binom{n-ab-\ell}{w} w$ bits. The fast Walsh transforms involve $\ell 2^\ell \sum_{w \leq W} \binom{n-ab-\ell}{w}$ additions of integers bounded by V. Each Walsh transform is performed separately, using a table of size 2^ℓ.

If e_i is not successful then the attack can try again, as many as q times, without any additional queries. If e_i is successful then the final recursion, determining the remaining bits of e_i, has negligible cost for any reasonable parameters.

An attack against $n = 512$ with $\tau = 1/8$. Take $n = 512$, $\tau = 1/8$, $q = 5 \cdot 2^{35} + 3 \cdot 2^{21} + 1$, $a = 5$, $b = 44$, $W = 4$, and $\ell = 24$. Here $n - ab - \ell = 268$, and the chance that e_i has Hamming weight ≤ 4 in its final 268 positions is $\sum_{w \leq 4} \binom{268}{w} (1/8)^w (7/8)^{268-w} \approx 2^{-35.055}$. (For comparison: Requiring e_i to have Hamming weight ≤ 4 in its final $n - ab = 292$ positions, as in [20], would have chance only about $2^{-39.194}$.)

The initial q oracle outputs $(r_i, r_i s + e_i)$ consume $1024q$ bits of memory, about $5 \cdot 2^{42}$ bytes. The $(q-1)n$ expanded vectors $v_{j,k}$, together with their noisy dot products, consume $513(q-1)n$ bits of memory, about $5 \cdot 2^{53}$ bytes. The table

adds fewer than 2^{50} bytes. The computation of these vectors costs approximately $2^{58.910}$ bit operations. (For comparison: If we did not exploit the Ring-LPN structure then we would have to perform approximately q multiplications of $n \times n$ matrices, costing approximately $2^{65.3}$ bit operations by schoolbook matrix multiplication or somewhat fewer bit operations by fast matrix multiplication.)

Clearing $ab = 220$ bits produces at least $(q-1)n - 2^b a = 3 \cdot 2^{30}$ vectors; we discard any extra vectors so that $V = 3 \cdot 2^{30}$. The bias decreases to $(3/4)^{32} \approx 2^{-13.28}$, but $3 \cdot 2^{30}$ vectors easily filter out this level of noise. This clearing involves at most $a(q-1)n^2 \approx 2^{57.644}$ bit operations.

The dot products count approximately $2^{61.248}$ bits, the main bottleneck in the computation. The Walsh transforms use $2^{56.254}$ additions.

This computation is repeated $2^{35.055}$ times on average, for a total of about $2^{97.5}$ bit operations. We comment that all of these bit operations are easily vectorized. Once the final 292 bits of e_i are known, the Ring-LPN outputs for e_i are converted into Ring-LPN outputs for the first 220 bits of e_i, which are found recursively at much less cost. Once all of e_i is known, computing s is trivial.

To summarize, this attack finds the Ring-LPN-512 secret using $<2^{56}$ bytes of memory, $<2^{38}$ queries, and $<2^{98}$ bit operations, as announced in Section 1.

We chose these parameters to emphasize memory at some cost in bit operations. Instead taking $q = 2^{62} + 2^{61} + 2^{51}$, $a = 6$, $b = 69$, $W = 5$, and $\ell = 40$ would use $<2^{78}$ bytes of memory, $<2^{63}$ queries, and $<2^{88}$ bit operations, beating the algorithm of [22] both in space and in time.

An attack against $n = 1024$ with $\tau = 1/20$. The following example illustrates the impact of τ. Take $n = 1024$, $\tau = 1/20$, $q = 10$, $a = 2$, $b = 12$, $W = 2$, and $\ell = 2$. Here $n - ab - \ell = 998$, and the chance that e_i has Hamming weight ≤ 2 in its final 998 positions is $\sum_{w \leq 2} \binom{998}{w} (1/20)^w (19/20)^{998-w} \approx 2^{-63.369}$.

The initial q oracle outputs consume 2560 bytes of memory, the $(q-1)n$ expanded vectors consume 1180800 bytes of memory, and the table consumes 519168 bytes of memory. The computation of these vectors costs approximately $2^{26.909}$ bit operations. Clearing $ab = 24$ bits produces at least $(q-1)n - 2^b a = 1024$ vectors, easily filtering out a bias of $0.9^4 = 0.6561$; this clearing involves at most $a(1-q)n^2 \approx 2^{24.170}$ bit operations. The dot products count approximately $2^{28.927}$ bits. The Walsh transforms use $2^{21.927}$ additions.

Overall one iteration of the attack uses $2^{29.373}$ bit operations. Running $2^{63.369}$ iterations of the attack uses $<2^{21}$ bytes of memory, $<2^{64}$ queries (with q iterations for each batch of q queries), and $<2^{93}$ bit operations.

For comparison, [22] says that $n = 1024$ and $\tau = 1/20$ take 2^{118} bytes of memory; the number of bit operations is even larger. We emphasize that the drastic reduction in memory consumption, and most of the reduction in bit operations, should be credited to Kirchner's paper [20].

A variant of the same attack is somewhat slower but reduces the total number of queries to just 10 and still fits into $<2^{21}$ bytes of memory. Instead of targeting 1 of 10 noisy vectors (producing 9 noisy vectors) and then forgetting the ring structure (producing 4608 noisy bits), first forget the ring structure (producing 5120 noisy bits) and then target 512 noisy bits (also producing 4608 noisy

bits). The remaining steps are exactly as before. This variant requires matrix multiplications instead of ring multiplications, but has the advantage of allowing $\binom{5120}{512} \approx 2^{2395}$ targets from the same 10 queries, far more than the $2^{63.369}$ targets needed on average. The same variant also works for LPN, using just 5120 queries.

References

[1] — (no editor), *Second international workshop on security, privacy and trust in pervasive and ubiquitous computing (SecPerU 2006), 29 June 2006, Lyon, France*, Institute of Electrical and Electronics Engineers, 2006. See [6].

[2] Avrim Blum, Adam Kalai, Hal Wasserman, *Noise-tolerant learning, the parity problem, and the statistical query model*, in STOC 2000 [**28**] (2000), 435–440; see also newer version [**3**]. Citations in this document: §2.

[3] Avrim Blum, Adam Kalai, Hal Wasserman, *Noise-tolerant learning, the parity problem, and the statistical query model*, Journal of the ACM **50** (2003), 506–519; see also older version [**2**]. Citations in this document: §1, §3.

[4] Colin Boyd (editor), *Advances in cryptology — ASIACRYPT 2001: proceedings of the 7th international conference on the theory and application of cryptology and information security held on the Gold Coast, December 9–13, 2010, proceedings*, Lecture Notes in Computer Science, 2248, Springer, 2001. ISBN 3-540-42987-5. See [16].

[5] Julien Bringer, Herve Chabanne, *Trusted-HB: a low-cost version of HB^+ secure against man-in-the-middle attacks*, IEEE Transactions on Information Theory (2008), 4339–4342. URL: http://eprint.iacr.org/2008/042. Citations in this document: §1.

[6] Julien Bringer, Herve Chabanne, Emanuelle Dottax, *HB^{++}: a lightweight authentication protocol secure against some attacks*, in [**1**] (2006), 28–33. Citations in this document: §1.

[7] Anne Canteaut (editor), *Fast software encryption*, 2012. See [15].

[8] Dang Nguyen Duc, Kwangjo Kim, *Securing HB^+ against GRS man-in-the-middle attack*, in Proceedings of SCIS 2007 (2007). Citations in this document: §1.

[9] Cory Edwards, Todd Edwards, Tony Leech, *Hoodwinked!* (2005). URL: http://www.imdb.com/title/tt0443536/. Citations in this document: §⋆.

[10] Thomas Eisenbarth, Zheng Gong, Tim Güneysu, Stefan Heyse, Sebastiaan Indesteege, Stéphanie Kerckhof, François Koeune, Tomislav Nad, Thomas Plos, Francesco Regazzoni, François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, *Compact implementation and performance evaluation of block ciphers in ATtiny devices*, Proceedings of the ECRYPT Workshop on Lightweight Cryptography, Louvain-la-Neuve, Belgium, November 2011; Africacrypt 2012, to appear (2011). URL: http://perso.uclouvain.be/fstandae/lightweight_ciphers/. Citations in this document: §1.

[11] Martin Feldhofer, Sandra Dominikus, Johannes Wolkerstorfer, *Strong authentication for RFID systems using the AES algorithm*, in CHES 2004 [**17**] (2004), 357–370. Citations in this document: §1.

[12] Dmitry Frumkin, Adi Shamir, *Un-Trusted-HB: security vulnerabilities of Trusted-HB* (2009). URL: http://eprint.iacr.org/2009/044. Citations in this document: §1.

[13] Henri Gilbert, Matthew J. B. Robshaw, Yannick Seurin, *HB$^\sharp$: increasing the security and efficiency of HB$^+$*, in EUROCRYPT 2008 [**27**] (2008), 361–378. Citations in this document: §1.

[14] Johan Håstad, *Some optimal inapproximability results*, in STOC 1997 [**21**] (1997), 1–10. Citations in this document: §2.

[15] Stefan Heyse, Eike Kiltz, Vadim Lyubashesvky, Christof Paar, Krzysztof Pietrzak, *Lapin: An efficient authentication protocol based on Ring-LPN*, in FSE 2012 [**7**] (2012), 326–340. Citations in this document: §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §1, §2, §2, §2.

[16] Nicholas J. Hopper, Manuel Blum, *Secure human identification protocols*, in ASIACRYPT 2001 [**4**] (2001), 52–66. Citations in this document: §1, §2.

[17] Marc Joye, Jean-Jacques Quisquater (editors), *Cryptographic hardware and embedded systems — CHES 2004: 6th international workshop, Cambridge, MA, USA, August 11–13, 2004, proceedings*, Lecture Notes in Computer Science, 3156, Springer, 2004. ISBN 3-540-22666-4. See [11].

[18] Ari Juels, Stephen A. Weis, *Authenticating pervasive devices with human protocols*, in CRYPTO 2005 [**26**] (2005), 293–308. Citations in this document: §1.

[19] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, Daniele Venturi, *Efficient authentication from hard learning problems*, in EUROCRYPT 2011 [**24**] (2011), 7–26. Citations in this document: §1, §2.

[20] Paul Kirchner, *Improved generalized birthday attack* (2011). URL: http://eprint.iacr.org/2011/377. Citations in this document: §1, §1, §3, §3, §3, §4, §4.

[21] Frank Thomson Leighton, Peter W. Shor (editors), *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, ACM, 1997. See [14].

[22] Éric Levieil, Pierre-Alain Fouque, *An improved LPN algorithm*, in SCN 2006 [**25**] (2006), 348–359. Citations in this document: §1, §1, §1, §1, §1, §1, §1, §1, §3, §3, §4, §4.

[23] Jorge Munilla, Alberto Peinado, *HB-MP: a further step in the HB-family of lightweight authentication protocols*, Computer Networks **51** (2007), 2262–2267. Citations in this document: §1.

[24] Kenneth G. Paterson (editor), *Advances in cryptology — EUROCRYPT 2011, 30th annual international conference on the theory and applications of cryptographic techniques, Tallinn, Estonia, May 15–19, 2011, proceedings*, Lecture Notes in Computer Science, 6632, Springer, 2011. ISBN 978-3-642-20464-7. See [19].

[25] Roberto De Prisco, Moti Yung (editors), *Security and cryptography for networks, 5th international conference, SCN 2006, Maiori, Italy, September 6–8, 2006, proceedings*, Lecture Notes in Computer Science, 4116, Springer, 2006. ISBN 3-540-38080-9. See [22].

[26] Victor Shoup (editor), *Advances in cryptology — CRYPTO 2005: 25th annual international cryptology conference, Santa Barbara, California, USA, August 14–18, 2005, proceedings*, Lecture Notes in Computer Science, 3621, Springer, 2005. ISBN 3-540-28114-2. See [18].

[27] Nigel P. Smart (editor), *Advances in cryptology — EUROCRYPT 2008, 27th annual international conference on the theory and applications of cryptographic techniques, Istanbul, Turkey, April 13–17, 2008, proceedings*, Lecture Notes in Computer Science, 4965, Springer, 2008. ISBN 978-3-540-78966-6. See [13].

[28] F. Frances Yao, Eugene M. Luks (editors), *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, ACM, 2000. ISBN 1-58113-184-4. See [2].