Smashing SQUASH-0

Khaled Ouafi[⋆] and Serge Vaudenay

EPFL
CH-1015 Lausanne, Switzerland
http://lasecwww.epfl.ch

Abstract. At the RFID Security Workshop 2007, Adi Shamir presented a new challenge-response protocol well suited for RFIDs, although based on the Rabin public-key cryptosystem. This protocol, which we call SQUASH-0, was using a linear mixing function which was subsequently withdrawn. Essentially, we mount an attack against SQUASH-0 with full window which could be used as a "known random coins attack" against Rabin-SAEP. We then extend it for SQUASH-0 with arbitrary window. We apply it with the proposed modulus $2^{1\,277} - 1$ to run a key recovery attack using 1 024 chosen challenges. Since the security arguments equally apply to the final version of SQUASH and to SQUASH-0, we challenge the blame-game argument for the security of SQUASH. Nevertheless, our attacks are inefficient when using non-linear mixing so the security of SQUASH remains open.

Key words: RFID, cryptanalysis, MAC

1 The SQUASH Algorithm

RFID tags use challenge-response protocols in which a reader sends a random challenge to a RFID tag to which this latter responds by computing the output of an algorithm (generally a MAC) fed with the challenge and a unique secret key. The reader then goes through a database containing a list of secrets associated with the identity of each tag to find the matching secret and thus the tag's identity. Due to computation and power constraints, most of the primitives proposed for RFID tags rely on symmetric key primitives since they offer competitive throughput, compared to public-key primitives which require much more transistors to be implemented as well as longer computation time.

At the RFID Security Workshop 2007, Adi Shamir [3] presented the SQUASH algorithm, a message authentication code (MAC) which, although based on the Rabin public-key cryptosystem, performs very well on benchmarks. In addition to this, it offers some kind of provable security based on the hardness of factoring large integers. This proof is used as a "safety net" as no attack using the modulus factors is known so far.

Essentially, SQUASH consists of a public hard-to-factor modulus N of ℓ bits and a r-bit length key which is also the length of the challenge. The algorithm is very simple as it only:

⋆ Supported by a grant of the Swiss National Science Foundation, 200021-119847/1.

– mixes the challenge and the secret key using a mixing function;
– converts it in a number;
– squares it modulo N;
– truncates the result to a specific window of bits.

There was essentially two versions of SQUASH. The first one considered a randomized version of the Rabin cryptosystem in which the square was hidden by adding a random multiple of N to avoid modulo N reduction, but which required a full-size window (i.e. no truncation at the end). The second one (preferred for efficiency reasons) considered a small window in the middle of the ℓ-bit result. Contrarily to the Rabin cryptosystem, SQUASH does not need to be invertible. So, the factorization of the modulus N does not need to be known by any participant. This motivated the recommendation of [4] to use Mersenne numbers (numbers of the form $N = 2^\ell - 1$) or the more general Cunningham project numbers (numbers of the form $N = a \cdot b^c \pm d$ for small a, b, c, d) whose factorization is unknown so far. Any other technical details regarding SQUASH are not relevant for our analysis.

Through this paper, we denote by R, K, C, and T the response, key, challenge, and truncation function of SQUASH respectively. The function SQUASH will simply consist of the following:

$$R = T\left(\left(\sum_{i=0}^{\ell-1} 2^i \times f_i(K,C)\right)^2 \bmod N\right),$$

where the f_i's are Boolean functions and the truncation function T is defined by

$$T(x) = \left\lfloor \frac{x \bmod 2^b}{2^a} \right\rfloor.$$

By expanding the square, we obtain:

$$R = T\left(\left(\sum_{i=0}^{\ell-1}\sum_{i'=0}^{\ell-1} 2^{i+i'} \times f_i(K,C) f_{i'}(K,C)\right) \bmod N\right) \tag{1}$$

The version of SQUASH presented in 2007, which we call SQUASH-0, uses a mixing function f expanding (using a linear feedback shift register) the XOR of the key and the challenge. It was subsequently updated in the version available on [3] following a private comment by Vaudenay. This comment was about a total break on the first version, i.e. the variant using no truncation. Nevertheless, the version published in [4] suggests to use a concrete non-linear mixing function. In this paper, we first present the attack by Vaudenay and apply it to any mixing function of form $g(K) \oplus L(C)$. Then, for L linear, we use discrete Fourrier transform to propose a variant of the attack and we generalize it to the second variant of SQUASH-0 using an arbitrary window.

Consequently, we restrict to the special case where there exists Boolean functions g_i and L_i such that

$$f_i(K,C) = g_i(K) \oplus L_i(C)$$

for all i, K, and C, where $\oplus$ denotes the exclusive or (XOR) operation. This is the case for the Rabin-SAEP encryption [1] where K plays the role of the plaintext and C plays the role of the random coins. In Sections 3 and 4, we further assume that the L_i's are linear in the sense that $L_i(a \oplus b) = L_i(a) \oplus L_i(b)$ for any a and b.

In what follows we first consider in Section 2 Vaudenay's passive attack against this algorithm with a full-size window. Namely, we assume the adversary gets the full Rabin encryption but no linearity in the L_i's. This translates into a "known random coins attack" against Rabin-SAEP. It works with complexity $O(\ell^2 r^6)$ and $O(r^2)$ known challenges. Next, we study an active variant of this attack in the linear case in Section 3 working in complexity $O(\ell r^2 \log r/\log\log r)$ and $O(r^2/\log\log r)$ chosen challenges. Finally, we apply the variant to the case of an arbitrary window in Section 4. Our final attack has a complexity of $O(\ell r^2 \log r)$ and uses $O(r^2)$ chosen challenges. It works when the window is large enough, namely $b - a > 4\log_2 r - 2$. However, in the case of Mersenne numbers, those figures boil down to a complexity of $O(r^2 \log r)$ and $O(r^2/\ell)$ chosen challenges, and the condition is relaxed to $b - a > 2\log_2 r - \log_2 \ell - 1$. When the window is smaller, there is still room for further improvements. We conclude that SQUASH-0 is dead broken and that the security proof in SQUASH is incorrect if factoring is hard. However, the security of SQUASH is still open.

2 Passive Attack with Full-Size Window

The goal of a passive total-break attack, is for an adversary to derive the secret key from challenge-response samples only. In what follows, we denote $k_i = (-1)^{g_i(K)}$ and $c_i = (-1)^{L_i(C)}$. We have

$$f_i(K,C) = \frac{1 - k_i c_i}{2}$$

which is linear (in the sense of $\mathbb{Z}$) in terms of k_i with coefficients known to the adversary. By expanding (1) we derive

$$R = \frac{1}{4}\sum_{i,i'} 2^{i+i'} c_i c_{i'} k_i k_{i'} - \frac{2^\ell - 1}{2}\sum_i 2^i c_i k_i + \frac{(2^\ell - 1)^2}{4} \bmod N \qquad (2)$$

when no truncation T is used.

A first attack consists of collecting enough equations of this form and solving them, e.g. by linearization or re-linearization [2]. Simple lineralization consists of expressing $k_i k_{i'}$ as a new unknown and solving linear equations. We get $\frac{r(r+1)}{2}$ unknowns and a solving algorithm of complexity $O(\ell^2 r^6)$ (as for $O(r^6)$ multiplications with complexity $O(\ell^2)$) after collection of $O(\ell r^2)$ bits (as for $O(r^2)$ samples of $O(\ell)$ bits). Since $k_i k_{i'} = \pm 1$ which is unexpectedly small, we can also consider algorithms based on lattice reduction using $O(\ell)$ samples only. The attack works even if the L_i's are not linear. In the Rabin-SAEP case, we obtain a "known random coins attack" in which an adversary can request many encryptions of the same plaintext and get the random coins with. His purpose is to recover the plaintext. However, for ℓ resp. r in the order of magnitude of 2^{10} resp. 2^6, complexities are still very high.

Interestingly, we note that when N is a Mersenne number then $N = 2^\ell - 1$ so Equation (2) simplifies by getting rid of r unknowns. Therefore, we have $\frac{r(r-1)}{2}$ unknowns instead of $\frac{r(r+1)}{2}$.

3 Active Attack with Full-Size Window

Let $C_1, \ldots, C_d$ be a set of d random challenges, given an integer d to be later discussed. Let U_i be the d-bit vector with coordinate $L_i(C_j)$, $j = 1, \ldots, d$. We consider an active attack making 2^d chosen challenges. Given a d-bit vector x, we define the challenge $C(x) = \bigoplus_j x_j C_j$ and $R(x)$, the response obtained after submitting this challenge. Given a real function $\varphi(x)$ and a d-bit vector V, we further recall the multidimentional discrete Fourrier transform of a function φ with group $\mathbb{Z}_2^d$:

$$\hat{\varphi}(V) = \sum_x (-1)^{x \cdot V} \varphi(x)$$

where $x \cdot V$ denotes the scalar dot product of vectors x and V. Thanks to the linearity of L_i, we have

$$c_i(x) = (-1)^{L_i(C(x))} = (-1)^{\bigoplus_j x_j L_i(C_j)} = (-1)^{x \cdot U_i}$$

Using the following:

- $\varphi(x) = 1 \Longrightarrow \hat{\varphi}(V) = 2^d \times 1_{V=0}$;
- $\varphi(x) = (-1)^{x_i U_i} \Longrightarrow \hat{\varphi}(V) = 2^d \times 1_{U_i = V}$;
- $\varphi(x) = (-1)^{x_i (U_i \oplus U_j)} \Longrightarrow \hat{\varphi}(V) = 2^d \times 1_{U_i \oplus U_j = V}$;

we deduce from Equation (2) that

$$\begin{aligned} \hat{R}(V) = & \frac{1}{4} \sum_{i,i'} 2^{i+i'} \left(\sum_x (-1)^{x \cdot (U_i \oplus U_{i'} \oplus V)} \right) k_i k_{i'} \\ & - \frac{2^\ell - 1}{2} \sum_i 2^i \left(\sum_x (-1)^{x \cdot (U_i \oplus V)} \right) k_i \\ & + \frac{(2^\ell - 1)^2}{4} \sum_x (-1)^{x \cdot V} \pmod{N}. \end{aligned} \tag{3}$$

3.1 First Method

Let I be an integer between 0 and $\ell - 1$. We assume that $C_1, \ldots, C_d$ are chosen such that

- for all j we have $L_I(C_j) = 0$;

– for every $i < i'$ there exists j such that $L_i(C_j) \neq L_{i'}(C_j)$.

In terms of U_i's, the hypotheses translate into observing that all the U_i's are pairwise different and that one of these vectors is the vector containing only 0's. Clearly, we can find these vectors by using an incremental algorithm to select C_j's in the hyperplane defined by $L_I(C) = 0$. If we generate d random vectors in the hyperplane, under heuristic assumptions, the probability that the condition is fulfilled is roughly $e^{-\ell^2 2^{-d-1}}$ which is constant for $d = 2\lceil \log_2 \ell \rceil$ and equal to $e^{-1/2}$.

By (3), thanks to the hypotheses we obtain

$$\hat{R}(0) = 2^{d-2} \sum_i 2^{2i} k_i^2 - 2^{d+I-1}(2^\ell - 1)k_I + \frac{2^d(2^\ell - 1)^2}{4} \pmod N$$

but since $k_i^2 = 1$ for all i we obtain

$$\hat{R}(0) = 2^{d-1}(2^\ell - 1)\left(\frac{2}{3}2^\ell - \frac{1}{3} - 2^I k_I\right) \pmod N$$

We can thus deduce k_I when N is not a Mersenne number. This means that recovering the key requires $O(r\ell^2)$ chosen challenges and complexity $O(r\ell^3)$. Clearly, we can trade data complexity against time complexity.

The Mersenne case. If N is a Mersenne number $N = 2^\ell - 1$, as suggested in [4], the above expression vanishes so we have to make a specific treatment. Actually, we have

$$R = \frac{1}{4} \sum_{i,i'} 2^{i+i'} c_i c_{i'} k_i k_{i'} \bmod N$$

By changing the assumption about the C_i's to

– there is a unique $I < J$ pair such that $U_I = U_J$

we obtain

$$\hat{R}(0) = 2^{I+J+d-1} k_I k_J \pmod N$$

so we can still adapt the attack. Other kinds of Cunningham numbers $N = a^b \pm c$, as suggested in [4], work like in the previous case.

3.2 Second Method

We now relax the assumption about the C_i's. By taking a value V such that

– $V \notin \{0, U_0, U_1, \ldots, U_{\ell-1}\}$
– there exists a unique $\{I, J\}$ pair such that $V = U_I \oplus U_J$

then (3) simplifies to

$$\hat{R}(V) = 2^{I+J-1+d} k_I k_J \pmod{N}$$

so we can deduce the value of $k_I k_J$.

The advantage of this method is that from the same set of challenges we can derive many equations of the form $k_I k_J = b$ (which are indeed linear equations) for all I and J such that $V = U_I \oplus U_J$ satisfies the above conditions. With random C_i's, the expected number of such equations is roughly $\frac{1}{2}\ell^2 e^{-\ell^2 2^{-d-1}}$ so for $d \approx 2\log_2 \ell$ we obtain enough equations to recover all bits of K using $O(\ell^2)$ chosen challenges and complexity $O(\ell^3 \log \ell)$.

3.3 Generalization

We can further generalize this attack by taking all values V which are either 0 or equal to some U_I or to some $U_I \oplus U_J$ but without requiring unicity of I or $\{I, J\}$. In general, we obtain an equation which may involve several k_I or $k_I k_J$ as Equation (3) simplifies to

$$\begin{aligned}\hat{R}(V) = & \sum_{\{I,J\}: U_I \oplus U_J = V} 2^{I+J-1+d} k_I k_J \\ & - \sum_{I: U_I = V} (2^\ell - 1) 2^{I+d-1} k_I + (2^\ell - 1)^2 2^{d-2} 1_{V=0} \pmod{N}.\end{aligned}$$

Provided that the number of monomials is not too large, the only correct ± 1 assignment of the monomials leading to an expression matching the $\hat{R}(V)$ value can be isolated.

Using $d = \log_2 \frac{r(r+1)}{2}$ we obtain only one unknown per equation on average so we can recover all key bits with complexity $O(\ell r^2 \log r)$ using $O(r^2)$ chosen challenges. We can still slightly improve those asymptotic figures.

Let ℓm be the complexity of getting the matching ± 1 assignments in one equation (i.e. m is the complexity in terms of modulo N additions). The complexity of the Fourier transform is $O(\ell d 2^d)$, so the complexity of the algorithm becomes $O(\ell(d + m)2^d)$. The average number of unknowns per equation is $r^2 2^{-d-1}$. By using an exhaustive search strategy to solve the equation we obtain $\log_2 m \approx r^2 2^{-d-1}$. With $d = 2\log_2 r - \log_2 \log_2 \log r - 1$ we have $m = \log r$ and we finally obtain a complexity of $O(\ell r^2 \log r / \log\log r)$ with $O(r^2/\log\log r)$ chosen challenges.

We could view the equation as a knapsack problem and use solving algorithms better than exhaustive search. For instance, we can split the equation in two halves and use a claw search algorithm. The effect of this strategy leads us to $\log_2 m \approx \frac{1}{2} r^2 2^{-d-1}$ and we reach the same asymptotic complexity. However the practical benefit may be visible as the following example shows.

Example 1. SQUASH with no truncation is trivially broken if we can factor the modulus N so it should be at least of 1 024 bits. As an example, for $\ell = 1\,024$ and

r arbitrary (up to ℓ) we can take $d = 14$ so roughly $2^d \approx 16\,000$ chosen challenges. We obtain at most $\frac{\ell(\ell+1)}{2}2^{-d} \approx 32$ unknowns per equation on average. Using claw search algorithm will work with 2^{16} numbers in memory and 2^{16} iterations to recover 32 bits of the key for each equation.

The Mersenne case. Finally, another nice thing with Mersenne numbers is that the equation further simplifies to

$$\hat{R}(V) = \sum_{n=0}^{\ell-1} 2^n \sum_{\substack{\{I,J\}:\\ U_I \oplus U_J = V, I+J-1+d \bmod \ell = n}} k_I k_J \pmod{N}. \tag{4}$$

So, if the set of $\{I, J\}$'s sparsely spread on $(U_I \oplus U_J, (I+J-1+d) \bmod \ell)$ pairs, the knapsack is nearly super-increasing and we can directly *read* all $k_I k_J$ bits in the table of all $\hat{R}(V)$'s. That is, m is constant. With $d = 2\log_2 r - \log_2 \ell - 1$ we roughly have ℓ unknowns per equation and we can expect this phenomenon. So, we obtain a complexity of $O(r^2 \log r)$ with $O(r^2/\ell)$ chosen challenges. For instance, with $N = 2^{1\,277} - 1$ and $r = 128$ we can take $d = 3$ so that 8 chosen challenges are enough to recover all bits. With $r = \ell$ we can take $d = 10$ so that 1 024 chosen challenges are enough.

Example 2. As a toy example we consider a SQUASH instance with $N = 13 \times 19 = 247$, $\ell = 8$, and $r = 4$ along with the function

$$f(K, C) = (K \oplus C) \| (K \oplus C)$$

with $d = 2$.

If $K = \mathsf{0x9}$, we have $k_0 = k_3 = k_4 = k_7 = -1$ and $k_1 = k_2 = k_5 = k_6 = +1$. We take 2 random values for the C_i's: $C_1 = \mathsf{0x2}$ and $C_2 = \mathsf{0xa}$. Here is a table for the $C(x)$ values:

x	$C(x)$	$f(K, C(x))$	$R(x)$
00	0x0	0x99 = 153	191
10	0x2	0xbb = 187	142
01	0xa	0x33 = 51	131
11	0x8	0x11 = 17	42

V	$\hat{R}(V)$
00	506
10	138
01	160
11	-40

The U_i's are

$$U_0 = U_4 = 00, \quad U_1 = U_5 = 11, \quad U_2 = U_6 = 00, \quad U_3 = U_7 = 01.$$

We sort monomials following the corresponding values of V as follows:

– $V = 00$: k_0, k_4, k_2, k_6, k_0k_4, k_1k_5, k_2k_6, k_3k_7, k_0k_2, k_0k_6, k_2k_4, k_4k_6,
– $V = 01$: k_3, k_7, k_0k_3, k_0k_7, k_3k_4, k_4k_7, k_2k_3, k_2k_7, k_3k_6, k_6k_7,

– $V = 10$: k_1k_3, k_1k_7, k_3k_5, k_5k_7,
– $V = 11$: k_1, k_5, k_0k_1, k_0k_5, k_1k_4, k_4k_5, k_1k_2, k_1k_6, k_2k_5, k_5k_6.

Due to the structure of f we know that $k_0 = k_4$, $k_1 = k_5$, $k_2 = k_6$ and $k_3 = k_7$ so the list simplifies (modulo $N = 247$) to:

$$\begin{aligned}
\hat{R}(00) &= \left(2^{0+4+1} + 2^{1+5+1} + 2^{2+6+1} + 2^{3+7+1}\right) \\
&\quad + \left(2^{0+2+1} + 2^{0+6+1} + 2^{2+4+1} + 2^{4+6+1}\right) k_0k_2 \\
&\quad -(2^8-1)\left(2^{0+1} + 2^{4+1}\right) k_0 - (2^8-1)\left(2^{2+1} + 2^{6+1}\right) k_2 \\
&\quad +(2^8-1)^2 + \frac{1}{3}(2^{16}-1) \\
\hat{R}(10) &= \left(2^{1+3+1} + 2^{1+7+1} + 2^{3+5+1} + 2^{5+7+1}\right) k_1k_3 \\
\hat{R}(01) &= -(2^8-1)\left(2^{3+1} + 2^{7+1}\right) k_3 \\
&\quad + \left(2^{0+3+1} + 2^{0+7+1} + 2^{3+4+1} + 2^{4+7+1}\right) k_0k_3 \\
&\quad + \left(2^{2+3+1} + 2^{2+7+1} + 2^{3+6+1} + 2^{6+7+1}\right) k_2k_3 \\
\hat{R}(11) &= \left(2^{0+1+1} + 2^{0+5+1} + 2^{1+4+1} + 2^{4+5+1}\right) k_0k_1 \\
&\quad + \left(2^{1+2+1} + 2^{1+6+1} + 2^{2+5+1} + 2^{5+6+1}\right) k_1k_2 \\
&\quad -(2^8-1)\left(2^{1+1} + 2^{5+1}\right) k_1
\end{aligned}$$

which yields

$$\begin{aligned}
\hat{R}(00) &= 176 + 89k_0k_2 - 25k_0 - 100k_2 \pmod{247} \\
\hat{R}(10) &= 109k_1k_3 \pmod{247} \\
\hat{R}(01) &= -200k_3 + 178k_0k_3 + 218k_2k_3 \pmod{247} \\
\hat{R}(11) &= 168k_0k_1 + 178k_1k_2 - 50k_1 \pmod{247}
\end{aligned}$$

The only values of k_0 and k_2 leading to $\hat{R}(00) = 506 \pmod{N}$ is $k_0 = -1$ and $k_2 = +1$. Similarly, k_1 and k_3 lead to $\hat{R}(10) = 138 \pmod{N}$ if and only if $k_1 = -k_3$. Using $\hat{R}(01) = 160 \pmod{N}$, we deduce $k_3 = -1$. From these values, we recover the key $K = 9$.

4 Application to Limited Windows

In what follows we let S denote the Rabin encryption. We assume that S is truncated to a window defined by

$$T(x) = \left\lfloor \frac{x \bmod 2^b}{2^a} \right\rfloor$$

so that $R = T(S)$. Our analysis from Section 3 still applies when R is replaced by S. However, S is not directly available to the adversary.

Since it is not clear how to break this variant of SQUASH even when N can be factored, ℓ could be much smaller than usual values for modulo bit-length. Indeed, [4] suggested an aggressive $\ell = 128$ with the Mersenne number $N = 2^{128} - 1$, $a = 48$, $b = 80$, and $r = 64$.

4.1 First Method

First, by observing that for any $e_1, \ldots, e_n \in \mathbb{Z}_N$ we have

$$\left(\sum_{i=1}^{n} e_i \bmod N\right) \bmod 2^b = \left(\left(\sum_{i=1}^{n} e_i \bmod 2^b\right) + \left(-\beta N \bmod 2^b\right)\right) \bmod 2^b \quad (5)$$

for some $0 \le \beta < n$ thus

$$T\left(\sum_{i=1}^{n} e_i \bmod N\right) = \left(\sum_{i=1}^{n} T(e_i) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}$$

for some $0 \le \alpha \le n$.

We now apply the previous attack (first method) with $n = 2^d$ and the list of all d-bit vectors x. We use $e_i = S(x)$ corresponding to the challenge $C(x)$. We deduce

$$T\left(\hat{S}(0) \bmod N\right) = \left(\hat{R}(0) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}.$$

Let

$$y_I^b = 2^{d-1}(2^\ell - 1)\left(\frac{2}{3}2^\ell - \frac{1}{3} - 2^I b\right) \bmod N$$

for $b = \pm 1$. Based on the previous attack we obtain

$$T(y_I^{k_I}) = \left(\hat{R}(0) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}$$

for some $\alpha \in [0, 2^d]$ and $\beta \in [0, 2^d - 1]$. The probability that there exists α and β such that $T(y_I^{-k_I})$ matches the right-hand side of the equation is at most 2^{2d-r}, so for $2d + 1 < r$ it is likely that we can deduce k_I.

The Mersenne case. When N is a Mersenne prime number, the $T(-\beta N)$ expression simplifies to $T(\beta)$. This is always 0 for $d \le a$ and it can be integrated in the α in $T(-\beta N) + \alpha$ in other cases: all $T(-\beta N) + \alpha$ values are numbers in the $\left[0, 2^d + \left\lfloor \frac{2^d - 1}{2^a} \right\rfloor\right]$ range. In what follows we assume that $d \le a$ for simplicity.

We now use $y_{I,J}^b = 2^{I+J+d-1} b \bmod N$ and, with the updated hypotheses on the C_j vectors, we obtain

$$T(y_{I,J}^{k_I k_J}) = \left(\hat{R}(0) + \alpha\right) \bmod 2^{b-a}$$

for some α in the $[0, 2^d]$ range. Note that $T(y_{I,J}^{-1}) = \overline{T(y_{I,J}^{+1})}$ and that

$$T(y_{I,J}^{+1}) = T\left(2^{(I+J+d-1) \bmod \ell}\right)$$
$$= \begin{cases} 2^{((I+J+d-1) \bmod \ell)-a} & \text{if } a \leq (I+J+d-1) \bmod \ell < b \\ 0 & \text{otherwise.} \end{cases}$$

This is enough to deduce $k_I k_J$ for (I, J) pairs such that there is no α for which $T(y_{I,J}^{-k_I k_J})$ matches the right-hand side. Thus we can recover $k_I k_J$.

4.2 Second Method

Similarly to (5), if $\varepsilon_i = \pm 1$ and $\varepsilon_1 + \cdots + \varepsilon_n = 0$ we have

$$\left(\sum_{i=1}^{n} \varepsilon_i e_i \bmod N\right) \bmod 2^b = \left(\left(\sum_{i=1}^{n} \varepsilon_i (e_i \bmod 2^b)\right) - \beta N \bmod 2^b\right) \bmod 2^b$$

for some $|\beta| < \frac{n}{2}$ thus

$$T\left(\sum_{i=1}^{n} \varepsilon_i e_i \quad \bmod N\right) = \left(\sum_{i=1}^{n} \varepsilon_i T(e_i) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}$$

for some $-\frac{n}{2} < \alpha \leq \frac{n}{2}$. We deduce that for each V there exist α and β verifying

$$T\left(\hat{S}(V) \bmod N\right) = \left(\hat{R}(V) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}.$$

With the appropriate conditions on the set of challenges we obtain

$$T\left((2^{I+J-1+d} k_I k_J\right) \bmod N) = \left(\hat{R}(V) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}$$

which can be used to recover $k_I k_J$.

In the Mersenne case with $d < a$, $T(-\beta N)$ is either 0 or $2^{b-a} - 1$ depending on the sign of β so we have a single additional value for $T(-\beta N) + \alpha$ which is $-\frac{n}{2}$. Since we will always take $d < a$ we omit the other cases.

4.3 Generalization

More generally, for all V there exists α and β such that

$$T\left(\left(\sum_{\{I,J\}: U_I \oplus U_J = V} 2^{I+J-1+d} k_I k_J - \sum_{I: U_I = V} (2^\ell - 1) 2^{I+d-1} k_I + (2^\ell - 1)^2 2^{d-2} 1_{V=0}\right) \bmod N\right)$$
$$= \left(\hat{R}(V) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}$$

with either $-\frac{n}{2} < \alpha \leq \frac{n}{2}$ and $|\beta| < \frac{n}{2}$ for $V \neq 0$ or $0 \leq \alpha \leq n$ and $0 \leq \beta < n$ for $V = 0$.

Our attack strategy can now be summarized as follows.

1. Take a value for d. Make a table of all $T(-\beta N) + \alpha$ values. This table has less than 2^{2d} terms ($2^d + 1$ in the Mersenne case) and can be compressed by dropping the d least significant bits (corresponding to the α part). In the Mersenne case, it can be compressed to nothing as numbers of form $T(-\beta N) + \alpha$ are all in the $[-2^{d-1}, 2^{d-1}]$ range modulo 2^{b-a}.
2. Pick d challenges at random and query all the 2^d combinations $C(x)$. Get the responses $R(x)$.
3. Compute the discrete Fourier transform $\hat{R}$ in $O(\ell d 2^d)$.
4. For each V, try all ± 1 assignments of occurring unknowns in $\hat{S}(V)$ and keep all those such that $T(\hat{S}(V) \bmod N) - \hat{R}(V)$ is of form $T(-\beta N) + \alpha$.

Again, this attack uses $O(2^d)$ chosen challenges and a complexity of $O(\ell(d + 2^{s2^{-d}})2^d)$ where s is the number of unknowns, i.e. $s = \frac{r(r+1)}{2}$ resp. $s = \frac{r(r-1)}{2}$ in the Mersenne case. The remaining question is whether all wrong assignments are discarded.

For a given equation, each of the $2^{s2^{-d}}$ wrong assignments is discarded with probability $2^{2d-(b-a)}$ resp. $2^{d-(b-a)}$. Thus, if $b - a > 2d + s2^{-d}$ resp. $b - a > d + s2^{-d}$ they can all be filtered out. The minimum of the right-hand side is $2\log_2 s + 2\log_2 \frac{e\ln 2}{2}$ resp. $\log_2 s + \log_2(e \ln 2)$ and reached by $d = \log_2 s + \log_2 \frac{\ln 2}{2}$ resp. $d = \log_2 s + \log_2 \ln 2$. By taking this respective value for d we have $O(r^2)$ chosen challenges and a complexity of $O(\ell r^2 \log r)$, and the condition becomes $b - a > 4\log_2 r + 2\log_2 \frac{e\ln 2}{2} - 2$ resp. $b - a > 2\log_2 r + \log_2(2e\ln 2)$. If $b - a > 4\log_2 r - 2$ resp. $b - a > 2\log_2 r$ this condition is always satisfied.

Example 3. We continue our toy example with $a = 1$ and $b = 7$ (i.e. we drop the least and most significant bits after the Rabin encryption). We still use 4 chosen challenges which are the linear combinations of $C_1 = 2$ and $C_2 = 10$ and compute the $\hat{R}(V)$ values (without any modular reduction). Here is a table for the $R(x)$ and $\hat{R}(V)$ values:

x	$C(x)$	$f(K, C(x))$	$R(x)$
00	0x0	0x99 = 153	$T(191) = 31$
10	0x2	0xbb = 187	$T(142) = 7$
01	0xa	0x33 = 51	$T(131) = 1$
11	0x8	0x11 = 17	$T(42) = 21$

V	$\hat{R}(V)$
00	60
10	4
01	16
11	44

The possible values of $T(-\beta N)$ for $|\beta| < \frac{n}{2}$ are in $\{0, 4, -5\}$. The possible values for $T(-\beta N) + \alpha$ are in $[-7, 5] - \{-3\}$. We take the equation $\hat{R}(10) = 4$. The possible values for $\hat{R}(10) + T(-\beta N) + \alpha$ modulo 64 are in $[-3, 9] - \{1\}$. On the other hand, $T\,(109k_1k_3 \bmod N)$ can only be 5 (for $k_1k_3 = -1$) or -10 (for $k_1k_3 = +1$) so we deduce $k_1k_3 = -1$. Similarly, for $V = 01$ we obtain

$$T\,(178k_0k_3 + 218k_2k_3 - 200k_3 \bmod N) \in [9, 21] - \{13\}$$

The 8 possible values for $T\,(178k_0k_3 + 218k_2k_3 - 200k_3 \bmod N)$ are

k_0k_3 k_2k_3 k_3	$+\,+\,+$	$+\,+\,-$	$+\,-\,+$	$+\,-\,-$	$-\,+\,+$	$-\,+\,-$	$-\,-\,+$	$-\,-\,-$
T	34	51	3	16	43	56	8	25

We deduce $k_3 = -1$, $k_0k_3 = +1$, $k_2k_3 = -1$ as the only possible assignment. Again, we recover $K = 9$.

The Mersenne case. Finally, the Mersenne case can simplify further using Equation (4). We take $d = 2\log_2 r - \log_2 \ell - 1$ and run the attack with $O(r^2/\ell)$ chosen challenges and complexity $O(r^2 \log r)$. Assuming that all unknowns k_Ik_J sparsely spread on $(U_I \oplus U_J, (I + J - 1 + d) \bmod \ell)$ pairs then $T(\hat{R}(V) \bmod N)$ yields $b-a-d$ useful bits with roughly one k_Ik_J per bit and ends with d garbage bits coming from $T(-\beta N) + \alpha$. So, we can directly *read* the bits through the window and it works assuming that $b - a > d$, which reads $b - a > 2\log_2 r - \log_2 \ell - 1$.

Example 4. We now use the parameters from [4]: $\ell = 128$, $N = 2^{128} - 1$, $a = 48$, $b = 80$. Although [4] suggested a 64-bit secret with non-linear mixing we assume here that the mixing is of form $f = g \oplus L$ with linear L but that g expands to $r = 128$ secret bits (possibly non-linearly). We have $s = 8\,128$ unknowns of form $k_ik_{i'}$. With $d = 10$ we obtain $1\,024$ vectors V so we can expect to find 8 unknowns in each equation. Equations are of form

$$T(\hat{S}(V) \bmod N) = \left(\hat{R}(V) + T(-\beta N) + \alpha\right) \bmod 2^{b-a}$$

where $(T(-\beta N) + \alpha) \bmod 2^{b-a}$ is in the range $[-2^9, 2^9]$ which gives a set of at most $2^{10} + 1$. Filtering the $2^8 - 1$ wrong assignments on the 8 unknowns we can expect 2^{-13} false acceptances in addition to the right one. Simple consistency checks can discard wrong assignments, if any, and recover all k_i's. Clearly, all computations are pretty simple and we only used 2^{10} chosen challenges.

Using the final trick in the Mersenne case we use $d = 6$ and thus 64 chosen challenges to get 64 equations which yield 26 bits each.

Example 5. With $N = 2^{1\,277} - 1$ and the worst case $\ell = r$ the attack works for $b - a \geq 21$ and we can take $d = 19$. We request for 2^{19} chosen challenges. We obtain 2^{19} equations with roughly 1.6 unknowns per equation.

By using the final trick we take $d = 10$. The $T(-\beta N) + \alpha$ part wastes 10 bits from the window and we can expect to have a single unknown per remaining bit so that we can simply read it through the window. Provided that the window has at least 32 bits we expect to read 22 bits in each of the $1\,024$ equations so we can recover all bits.

The narrow window case. When $b - a$ is too small for the previous attack to work, we can still work further on the analysis. To avoid wasting bits on the window, we shall decrease the value for d. Typically, our attack will get less equations and have more unknowns per equation. As a consequence it will list many assignments, including a single correct one. Several directions can be investigated:

- use the highly biased distribution of most significant garbage bits (since α has expected value 0 and standard deviation roughly $2^{\frac{d}{2}}/2\sqrt{3}$);
- use small d and better knapsack solving algorithms;
- analyze these assignments on redundant bits and check for consistency within an equation;
- analyze assignment lists in several equations and try to merge;
- use voting procedures and iterate.

This extension is left to further research.

5 Extending to Non-linear mappings

In case the mapping L is a (non-linear) *permutation*, we can adapt our attack strategy by choosing the challenges as follow:

- pick d challenges $C_1, \ldots, C_d$.
- compute the chosen challenges by $C^\star(x) = L^{-1}\left(\bigoplus_j x_j L(C_j)\right)$.

By using,

$$c_i^\star(x) = (-1)^{L_i(C^\star(x))} = (-1)^{\bigoplus_j x_j L_i(C_j)} = (-1)^{x \cdot U_i}$$

Equation (3) remains unchanged so that we can still apply all the attacks described through Sections 3 and 4.

More generally, we can extend these attacks to *any* mixing function of form $f(K, C) = g(K) \oplus L(C)$ as long as we can find vector spaces of dimension d in the range of L.

6 Conclusion

One argument for motivating the SQUASH algorithm consisted of playing the "blame game": if anyone can break SQUASH, then the Rabin cryptosystem is the one which should be blamed instead of the SQUASH design. Clearly, our attack demonstrates that this argument is not correct. There are instances of the SQUASH algorithm which can be broken although we still have no clue how to factor integers. Indeed, our method translates into a "known random coins attack" against Rabin-SAEP which leads to a plaintext recovery. Known random coins attacks are not relevant for public-key cryptosystems although they are in the way SQUASH is using it.

It is not clear how and if our attack can be adapted to the final version of SQUASH with non-linear mappings. So, although the "blame game" argument is not valid, the security of SQUASH is still an open problem.

References

1. Dan Boneh. Simplified OAEP for the RSA and Rabin functions. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 275–291. Springer, 2001.
2. Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.
3. Adi Shamir. SQUASH: A new one-way hash function with provable security properties for highly constrained devices such as RFID tags. Invited lecture to the RFID Security'07 Workshop. Slides available from http://mailman.few.vu.nl/pipermail/rfidsecuritylist/2007-August/000001.html.
4. Adi Shamir. SQUASH - a new MAC with provable security properties for highly constrained devices such as RFID tags. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 2008.