Key Recovery Attacks on Recent Authenticated Ciphers

Andrey Bogdanov[1], Christoph Dobraunig[2], Maria Eichlseder[2], Martin M. Lauridsen[1], Florian Mendel[2], Martin Schläffer[2], and Elmar Tischhauser[1]

[1] DTU Compute, Technical University of Denmark, Denmark
[2] IAIK, Graz University of Technology, Austria

Abstract. In this paper, we cryptanalyze three authenticated ciphers: AVALANCHE, Calico, and RBS. While the former two are contestants in the ongoing international CAESAR competition for authenticated encryption schemes, the latter has recently been proposed for lighweight applications such as RFID systems and wireless networks.
All these schemes use well-established and secure components such as the AES, Grain-like NFSRs, ChaCha and SipHash as their building blocks. However, we discover key recovery attacks for all three designs, featuring square-root complexities. Using a key collision technique, we can recover the secret key of AVALANCHE in $2^{n/2}$, where $n \in \{128, 192, 256\}$ is the key length. This technique also applies to the authentication part of Calico whose 128-bit key can be recovered in 2^{64} time. For RBS, we can recover its full 132-bit key in 2^{65} time with a guess-and-determine attack. All attacks also allow the adversary to mount universal forgeries.

Keywords: authenticated encryption, CAESAR, key collision, guess-and-determine, universal forgery, AVALANCHE, Calico, RBS

1 Introduction

An authenticated cipher is a symmetric-key cryptographic algorithm that aims to provide both the confidentiality and authenticity of data – the two most fundamental cryptographic functionalities. Authenticated encryption has been used extensively since decades by combining encryption algorithms with message authentication algorithms. However, it was not until the 2000s that the separate security goal of authenticated encryption (AE) has been formulated [6, 13].

Authenticated encryption schemes can basically be constructed either as a generic composition of an encryption algorithm (a block or stream cipher) and a message authentication code (MAC), or as a dedicated AE design. Doing both encryption and authentication in one cryptographic primitive also has the advantage of attaining a potentially higher peformance. Indeed, such combined schemes as OCB [14, 19, 20] and OTR [17] require only one block cipher call per block of data processed to produce both the ciphertext and the authentication tag.

Owing to its relatively recent origins, only very few AE designs have been included in international standards, the most prominent examples being CCM [9] and GCM [10, 15] which have been included in ANSI/ISO, IEEE and IETF standards. Recently, the CAESAR competition [1] has been initiated in order to establish a portfolio of recommended AE algorithms during the next years, making authenticated encryption a major focus of the cryptographic community. A large number of diverse designs has been submitted to this competition, ranging from block cipher modes of operation to dedicated designs.

A substantial fraction of the CAESAR submissions build upon proven components, such as the AES, the SHA-2 family or the Keccak [8] permutation. An important reason for such a design decision is the assumption that the AE scheme will inherit the good security properties of the building blocks. For some designs, this is backed up by a formal

security reduction. However this is not the case for all candidates, and even when security proofs for confidentiality and integrity are provided, they are often limited by the birthday bound, while at the same time any key recovery attack with complexity below 2^n would be highly undesirable.

We analyze three recent proposals for authenticated encryption: AVALANCHE [4] and Calico [21] are general-purpose AE schemes that have been submitted to CAESAR, and RBS [12] has recently been proposed for lightweight applications. All three algorithms are based on secure building blocks: AVALANCHE uses the AES, Calico builds upon ChaCha [7] and SipHash [5], and RBS a Grain-like [3] register-accumulator architecture [2].

Despite being based on these secure components, our analysis establishes that all three algorithms admit key recovery attacks with complexities significantly below exhaustive search. AVALANCHE and Calico lend themselves to attacks based on key collisions, leading to a key recovery with complexity $2^{n/2}$ for all versions of AVALANCHE ($n \in \{128, 192, 256\}$), and with complexity 2^{64} for the 128-bit authentication key of Calico. For RBS, recovering its full 132-bit key requires 2^{65} time with a guess-and-determine strategy. For all algorithms, the recovered key material enables the adversary to obtain universal forgeries. In the case of AVALANCHE and RBS, the adversary also obtains all necessary key material for decryption of arbitrary messages.

All these attacks are entirely structural and do not make use of any weaknesses of the building blocks themselves. We give an overview of the attacks and their complexities in Table 1.

Table 1: Overview of the attacks presented in this paper. For the attacks marked with (*), memoryless variants are possible [18], reducing the memory requirements to $\mathcal{O}(1)$, eliminating the offline computations, and increasing the time complexity by a factor of about 2.

Algorithm	Recovered key bits	Time		Memory	Data
		Offline	Online		
AVALANCHE-128	384	2^{64}	2^{64}	$2^{64}(^\star)$	2^{64}
AVALANCHE-192	448	2^{96}	2^{96}	$2^{96}(^\star)$	2^{96}
AVALANCHE-256	512	2^{128}	2^{128}	$2^{128}(^\star)$	2^{128}
Calico	128	2^{64}	2^{64}	$2^{64}(^\star)$	2^{64}
RBS	132	–	2^{65}	$\mathcal{O}(1)$	1

The remainder of the paper is organized as follows. We introduce some common notation in Sect. 2. Sect. 3 describes the key recovery attack on AVALANCHE. Our attack on Calico is presented in Sect. 4. In Sect. 5, we describe our guess-and-determine attack on RBS. We conclude our findings in Sect. 6.

2 Notation

In the following, we use N, A, M and C to denote nonce, associated data (data which is authenticated but not encrypted), a message (to be encrypted and authenticated) and a ciphertext, respectively. For binary strings x and y, we let $|x|$ denote the bit-length of x and $x\|y$ is the concatenation of x and y. We use ϵ for the empty string, i.e. $|\epsilon| = 0$. Subscript usually denotes the bit index of a string, so x_i is the ith bit of x, with the convention that x_0 is the least significant, rightmost bit. We use $\oplus$ to denote the XOR operation and use $\mathrm{hw}(x)$ to denote the Hamming weight of x. In the case where X is a

binary string of blocks, and where the block size is understood to be b, we let $X[i]$ denote the ith block of X, i.e. $X[i] = x_{bi-1} \| \cdots \| x_{b(i-1)}$ for $i \geq 1$.

3 AVALANCHE

The AVALANCHE scheme [4] is a submission to the ongoing CAESAR competition for authenticated encryption ciphers. We note that the specification of AVALANCHE leaves some room for interpretation. In the aspects relevant to our attacks, we assume the following:

- The nonce has sizes $|N| \in \{80, 160, 128\}$ for key lengths $n \in \{128, 256, 192\}$, respectively.
- The nonce N is randomly generated.
- The counter c is initialized to $c = 0$.
- The tag length is $|T| = 128$ as well as the security parameters k and p are 128-bit.
- The $(n + 256)$-bit key K consists of three independent parts, $K = (K_P, k, p)$.

AVALANCHE uses the AES to process a message M of m blocks and associated data A of arbitrary length to produce a ciphertext C of $m+1$ blocks and an authentication tag T. It does not support a public message number, instead a nonce N is generated by the encryption algorithm itself.

The input to AVALANCHE with a specified secret key $K = (K_P, k, p)$, is a 3-tuple (M, A, K) of message and associated data. The output is a 4-tuple (N, A, C, T) of nonce, associated data, ciphertext, and tag. The scheme uses two main algorithms described in the following; PCMAC for message processing and RMAC for processing associated data. The interfaces and outputs of the two algorithms are

$$(N, C, \tau_P) = PCMAC(M) \qquad \text{and} \qquad \tau_A = RMAC(A).$$

The final tag T is then computed as $T = \tau_P \oplus \tau_A$.

3.1 PCMAC

The encryption with PCMAC is illustrated in Figure 1. The padded message is denoted $M[1] \cdots M[m]$ and the ciphertext $C[0] \cdots C[m]$. The number r is generated at random.

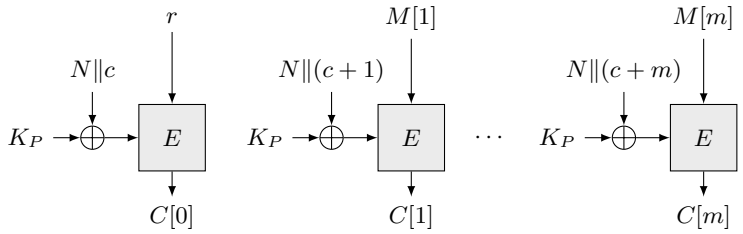


Fig. 1: Message processing with PCMAC

3.2 RMAC

The output of RMAC is an intermediate tag τ_A of 128 bits. RMAC uses the secrets k and p, p being a randomly chosen 128-bit prime and k chosen at random from $\{\lfloor p/2 \rfloor + 1, \ldots, p - 1\}$. The intermediate tag τ_A is determined as

$$\tau_A = (1 \| A) \cdot k \mod p. \tag{1}$$

3.3 Recovering the PCMAC key

The critical part of PCMAC is that the encryption key for E (see Figure 1) depends on the nonce and counter. This facilitates key collision attacks, similar to the one on McOE-X [16]. Our attack works in an *offline phase* and an *online phase* (see Algorithms 1 and 2). Both are called with the same, arbitrary single-message block M. The offline phase outputs a list L which is used in the online phase. We also note that this technique allows a free trade-off between time and memory by choosing the list size accordingly.

Algorithm 1: OFFLINE(M)

Data: Single-block M
$L \leftarrow \emptyset$
for $i = 1, \ldots, \ell$ **do**
  Choose new $K = (K_P, k, p) \in \{0,1\}^{n+256}$
  $(N, \epsilon, C, T) \leftarrow \mathsf{AVALANCHE}(M, \epsilon, K)$
  $L \leftarrow L \cup \{(C[1], K_P, N)\}$
end
return L

Algorithm 2: ONLINE(M, L)

Data: Single-block M, List L output from Algorithm 1
for $i = 1, \ldots, 2^n/\ell$ **do**
  Obtain (N, ϵ, C, T) for (M, ϵ) from AE
  if $\exists (x, y, z) \in L : x = C[1]$ **then**
    return $y \oplus ((N \oplus z) \| 0^{n-|N|})$
  end
end
return Failure

In the offline phase we build a table of size ℓ of AVALANCHE encryptions of the same message block, using different keys. In the online phase we request the encryption of the same single-block message M in total $2^n/\ell$ times. By the birthday paradox, we expect to see a collision in the oracle output $C[1]$ in the online phase and the list L from the offline phase. As the nonce N is public, we can then recover the secret key K_P by adding it to the stored nonce z and key y. We can verify candidate keys using an additional encryption. Obviously, choosing $\ell = 2^{n/2}$ gives the best overall complexity, using just $2 \cdot 2^{n/2}$ time and memory in the order of $2^{n/2}$ to store L. Memoryless versions of the meet-in-the-middle technique can be used here as well [18].

3.4 Recovering the RMAC secret parameters

To recover (k, p), we use the attack described above to recover the secret K_P. We furthermore ask for encryption and tag of some arbitrary message block; once with empty associated data, i.e. $A = \epsilon$, and once with $A = 0$, i.e. a single zero bit. Let the corresponding outputs of AVALANCHE be (N, ϵ, C, T) and $(N', 0, C', T')$, where $T = \tau_P \oplus \tau_A$ and $T' = \tau'_P \oplus \tau'_A$.

With K_P in hand, we can ourselves compute τ_P and τ'_P using PCMAC. Consequently, we obtain τ_A and τ'_A. Using the definition of RMAC of Eq. (1), we observe that for the case where $A = \epsilon$ we directly obtain $\tau_A \equiv k \mod p$, but since $k \in \{\lfloor p/2 \rfloor + 1, \ldots, p-1\}$ we have $k = \tau_A$. Now, for the case where $A = 0$, we find

$$\begin{aligned} \tau'_A &\equiv (1\|0) \cdot k \mod p \\ \Leftrightarrow \tau'_A &\equiv 2k \mod p \\ \Leftrightarrow p &= 2k - \tau'_A. \end{aligned}$$

We therefore obtain the secret parameters (k, p) of RMAC with a complexity of two one-block encryption queries.

In summary, we have recovered all $n + 256$ bits of secret key material in about $2^{n/2}$ time.

4 Calico

Calico [21] is an authenticated encryption design submitted to the CAESAR competition. For Calico in reference mode, ChaCha-14 [7] and SipHash-2-4 [5] work together in an Encrypt-then-MAC scheme [11]. The Calico design is depicted in Figure 2.

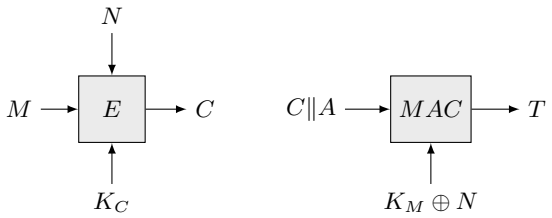


Fig. 2: The Calico scheme with encryption (left) and tag generation (right)

4.1 Specification

For the purpose of using ChaCha-14 and SipHash-2-4 in Calico, the 384-bit key K is split into two parts: a 256-bit encryption key K_C and a 128-bit authentication key K_M, s.t. $K = K_C \| K_M$. The plaintext is encrypted with ChaCha under K_C to obtain a ciphertext with the same length as the plaintext. Then, the tag is computed as the SipHash MAC of the concatenated ciphertext and associated data. The key used for SipHash is generated by XORing the nonce to the (lower, least significant part of the) MAC key K_M, so

$$(C, T) = \mathrm{Enc}_{\mathsf{Calico}}(K_C \| K_M, N, A, M),$$

where the ciphertext and tag, C and T respectively, are computed with

$$\begin{aligned} C &= \mathrm{Enc}_{\text{ChaCha-14}}(K_C, N, M) \\ T &= \mathrm{MAC}_{\text{SipHash-2-4}}(K_M \oplus N, C \| A). \end{aligned}$$

The tag T and nonce N are both 64 bits long.

4.2 MAC key recovery

In Calico, SipHash is modified by XORing a nonce to the lower-significance bits of the key. This modification of SipHash facilitates an attack similar to the one described by Mendel et al. [16]. The attack targets the tag generation to recover the MAC key, which in turn allows to forge tags for arbitrary associated data and ciphertexts.

We can split the attack into an offline phase and an online phase (see Algorithms 3 and 4), where the online phase requires access to an encryption oracle. Algorithm 4 does 2^{64} online queries for an overall complexity of 2^{64}. Tradeoffs are possible to reduce the number of online queries at the cost of the overall complexity.

Algorithm 3: OFFLINE

```
L ← ∅
for i = 0, ..., 2^64 − 1 do
    Compute tag T for A, M = ϵ under MAC key K_M = i‖0 and nonce N = 0
    L ← L ∪ {(T, K_M)}
end
```

Algorithm 4: ONLINE(L)

```
Data: List L output from Algorithm 3
for j = 0, ..., 2^64 − 1 do
    Request tag T for A, M = ϵ under nonce N = j from encryption oracle
    if ∃(x, y) ∈ L : x = T, y = i‖0 then
        i‖j is a candidate for K_M
    end
end
```

This produces at least one MAC key candidate; if necessary, remaining candidates can be filtered with additional offline computations, though their expected number is very small.

Since Calico preserves the plaintext length for the ciphertext, an empty plaintext and associated data will produce an empty input for the MAC, independent of the cipher key or nonce. Thus, all offline computations and online queries give tags calculated from the same MAC input, only with varying keys fed to SipHash. The SipHash keys used in the offline phase all have the lower 64 bits set to 0 and the upper 64 bits iterating through all possible values. In the online phase, the SipHash keys have the upper 64 bits set to the original bits of the secret K_M, while the lower bits iterate through all possibilities. Thus, there is exactly one match between the two key lists, which will also produce a colliding tag (though other tag pairs may collide as well). The matching key stored in the offline list gives the upper 64 bits of the correct key, the colliding nonce from the online phase the lower 64 bits.

For tradeoffs with online complexity $2^N < 2^{64}$, replace 2^{64} by 2^N in the online phase and by $2^{(128-N)}$ in the offline phase; the success probability remains 1. We note that memoryless versions of the meet-in-the-middle technique apply also here [18].

5 RBS

RBS is an authenticated encryption scheme by Jeddi et al. [12] proposed for use in RFID tags. The idea of RBS is to insert the bits of a MAC on the message among the message bits, in key-dependent positions, to produce the authenticated ciphertext.

5.1 Specification

The RBS scheme is depicted in Figure 3. It takes as input a 64-bit message M and a 132-bit key k to produce a 132-bit authenticated ciphertext C. Effectively, the key is split in two parts of sizes which we denote n and m respectively: the least n significant bits are used for clocking the MAC (which we described in detail later) while the most significant m bits are used for initializing the NFSR in the MAC. RBS uses $n = 64$ and $m = 68$, but we sometimes use n and m for generality in the following. Note that a requirement on the key k is that it has Hamming weight 68, and hence the size of the key space is $\binom{132}{68} \approx 2^{128.06}$.

The RBS MAC takes either a 64-bit or 68-bit input to be processed, along with the key k, and produces a 68-bit output. While RBS does not specify this, we assume (without influence on our attack) that the second MAC output is truncated by taking the least significant 64 bits to obtain the value S.

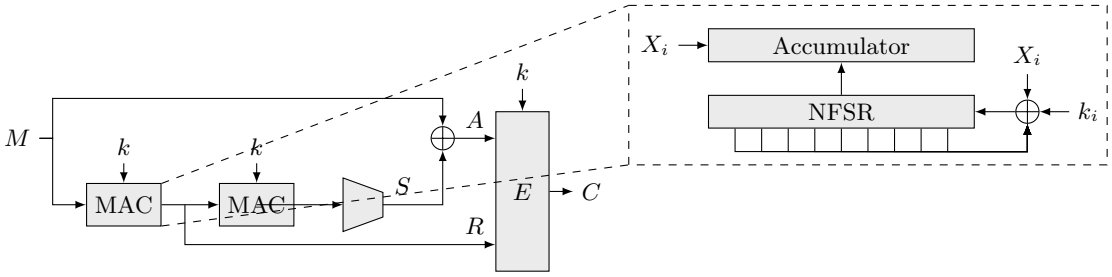


Fig. 3: The RBS scheme as an overview (left) and with the internals of the MAC (right)

Consider A and R of Figure 3 as registers of 64 bits and 68 bits, respectively. For the function E, the ith ciphertext bit, denoted C_i, is obtained as

$$C_i = \begin{cases} \text{least significant bit of } A & , k_i = 0 \\ \text{least significant bit of } R & , k_i = 1 \end{cases}.$$

Each time a bit is taken from A or R, to produce a ciphertext bit, the corresponding register is right-rotated one position. As 132 bits are produced for the ciphertext, E effectively obtains C by inserting the bits of R (the MAC of the message), in order, into A at key-dependent positions.

The RBS MAC. The MAC used in RBS which we denote $MAC(X, k)$, (depicted in the right side of Figure 3 where the input is denoted X) is a Grain-like design based on the MAC of Ågren et al. [2]. It is composed of a 68-bit NFSR and a 68-bit accumulator. In this work, we consider the NFSR with an arbitrary update function (and indeed the specification does not provide one). When a MAC is computed, the NFSR is loaded with the most significant 68 bits of the key, i.e. $k_{131}, \ldots, k_{64}$ and the accumulator is set to zero. To produce $MAC(X, k)$, the NFSR is clocked $|X|$ times, i.e. it is shifted left and the least significant bit is set to the feedback XORed with the input bit $X_i \oplus k_i$ where $i = 0, \ldots, |X|-1$. If and only if $X_i = 1$, the accumulator is updated by XORing the current NFSR state to it (we assume this is done prior to clocking the NFSR). When $|X| > 64$, which is the case for the second MAC call, we assume that one re-uses $k_{63}, \ldots, k_0$ for clocking, until all of X is processed, although this makes no difference to our attack.

5.2 Cryptanalysis of RBS

The attack on the RBS scheme we present in the following uses a single chosen plaintext and has *expected worst case* time complexity 2^{65} and negligible memory complexity. The attack is based on the following observations:

Observation 1. *When computing $R = MAC(M, k)$, if $M = 1$, then it immediately follows from the definition of the MAC that $R = k_{131} \| \cdots \| k_{64}$, i.e. the 68 most significant bits of the key.*

Observation 2. *Assuming one knows $k_{a-1} \| \cdots \| k_0$ for some a with $1 \leq a \leq 132$, then one can determine the first $\ell := hw(k_{a-1} \| \cdots \| k_0)$ bits of R, as the bits of R are directly mapped to C by the k_i where $k_i = 1$. These in turn correspond to the first ℓ bits of $k_{131} \| \cdots \| k_{64}$. These can in turn be used to determine more of R, and so on.*

Combined, these observations imply that for $M = 1$, we know that $R = k_{131}\| \cdots \|k_{64}$. When guessing any number of the least significant key bits, a number of bits of R and thus of $k_{131}\| \cdots \|k_{64}$, equal to the Hamming weight of the guess, can be directly obtained from C.

Definition 1 (Free bit iteration). *The ith* free bit iteration*, with $i \geq 0$, refers to the number of bits obtained "for free" in one such iteration.*

Thus, the 0th free bit iteration refers to the analysis of how many free bits are obtained from the initially guessed key bits; the 1st free bit iteration refers to how many free bits are obtained from the ones obtained from the 0th free bit iteration, and so on.

For $i \geq 0$, in the ith free bit iteration, we let ℓ_i denote the *expected* number of free bits obtained and let δ_i denote the *expected* density of 1-bits in the remaining unknown bits, after obtaining the ℓ_i free bits.

Lemma 1. *Let $k_{a-1}\| \cdots \|k_0$ be the initially guessed key bits and let $\ell_0 = hw(k_{a-1}\| \cdots \|k_0)$. Then*

$$\begin{aligned} \delta_i &= \frac{m - \sum_{j=0}^{i} \ell_j}{n + m - a - \sum_{j=0}^{i-1} \ell_j}, \qquad i \geq 0 \qquad \text{and} \\ \ell_i &= \ell_{i-1}\delta_{i-1}, \qquad i \geq 1. \end{aligned} \tag{2}$$

Proof. In the ith free bit iteration, $a + \sum_{j=0}^{i-1} \ell_j$ bits have already been guessed, so the denominator of δ_i is what remains unknown. The key bits guessed thus far have Hamming weight $\sum_{j=0}^{i} \ell_j$, so the 1-bits density among the last bits is δ_i.

The number of bits expected to obtained for free in iteration $i + 1$ is determined by the expected Hamming weight of the free bit portion just obtained in iteration i, which in turn is $\ell_i\delta_i$. □

We now derive a closed formula for the quantity ℓ_i by observing that the ratios ℓ_{i+1}/ℓ_i between consecutive elements of the sequence are actually constant, i.e. independent of i. We formally prove this in the following lemma.

Lemma 2. *Let a and ℓ_0 be such that $m - \ell_0 \neq n + m - a$ and $n + m - a \neq 0$. With the notations of Lemma 1, we have*

$$\ell_i = \left(\frac{m - \ell_0}{n + m - a}\right)^i \ell_0 \tag{3}$$

for $i \geq 1$.

Proof. We prove the claim by induction. For $i = 1$, Eq. (2) yields $\ell_1 = \frac{m-\ell_0}{n+m-a}\ell_0 = \left(\frac{m-\ell_0}{n+m-a}\right)^1 \ell_0$.

Assuming (3) holds for all $k \leq i$, we have

$$\begin{aligned} \ell_{i+1} &= \frac{m - \sum_{j=0}^{i} \ell_j}{n + m - a - \sum_{j=0}^{i-1} \ell_j} \cdot \ell_i \\ &= \frac{m - \ell_0 \sum_{j=0}^{i} \left(\frac{m-\ell_0}{n+m-a}\right)^j}{n + m - a - \ell_0 \sum_{j=0}^{i-1} \left(\frac{m-\ell_0}{n+m-a}\right)^j} \cdot \left(\frac{m - \ell_0}{n + m - a}\right)^i \ell_0. \end{aligned} \tag{4}$$

For $r \neq 1$, the geometric series $\sum_{i=0}^{N} r^i$ is equal to $\frac{r^{N+1}-1}{r-1}$. Instantiating this with $r = \frac{a}{b}$ yields $\sum_{i=0}^{N} \left(\frac{a}{b}\right)^i = \frac{\left(\frac{a}{b}\right)^N a - b}{a-b}$ and $\sum_{i=0}^{N-1} \left(\frac{a}{b}\right)^i = \frac{\left(\frac{a}{b}\right)^N b - b}{a-b}$. Since $\left(\frac{m-\ell_0}{n+m-a}\right) \neq 1$, we can apply this to the two sums in Eq. (4), yielding

$$\ell_{i+1} = \frac{m - \ell_0 \left(\frac{\left(\frac{m-\ell_0}{n+m-a}\right)^i (m-l_0) - (n+m-a)}{-\ell_0 - n + a} \right)}{n + m - a - \ell_0 \left(\frac{\left(\frac{m-\ell_0}{n+m-a}\right)^i (n+m-a) - (n+m-a)}{-\ell_0 - n + a} \right)} \cdot \left(\frac{m - \ell_0}{n + m - a} \right)^i \ell_0,$$

which can be reformulated to

$$= \frac{\left(\frac{m(-\ell_0 - n + a) - \ell_0 \left(\frac{m-\ell_0}{n+m-a}\right)^i (m-\ell_0) + \ell_0 (n+m-a)}{-\ell_0 - n + a} \right)}{\left(\frac{(n+m-a)(-\ell_0 - n + a) - \ell_0 \left(\frac{m-\ell_0}{n+m-a}\right)^i (n+m-a) + \ell_0 (n+m-a)}{-\ell_0 - n + a} \right)} \cdot \left(\frac{m - \ell_0}{n + m - a} \right)^i \ell_0,$$

and collecting common terms gives

$$\begin{aligned} &= \frac{(m-\ell_0)\left(\left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0 + n - a\right)}{\ell_0 + n - a} \cdot \frac{\ell_0 + n - a}{(n+m-a)\left(\left(\frac{m-\ell_0}{n+m-a}\right)^i \ell_0 + n - a\right)} \\ &\qquad \cdot \left(\frac{m - \ell_0}{n + m - a} \right)^i \ell_0 \\ &= \left(\frac{m - \ell_0}{n + m - a} \right) \cdot \left(\frac{m - \ell_0}{n + m - a} \right)^i \ell_0 \\ &= \left(\frac{m - \ell_0}{n + m - a} \right)^{i+1} \ell_0, \end{aligned}$$

as claimed. □

Note that the preconditions of the previous lemma are not imposing a limitation for the evaluation of the ℓ_i for relevant values of a. For instance, with $a = n$, the closed formula holds for any $1 \leq \ell_0 \leq m$, and the remaining case $\ell_0 = 0$ is trivial since all remaining unknown bits must be equal to one.

Optimal choice of a. The closed formula of Lemma 3 also yields an estimate for the optimal number of key bits a that should be guessed initially. Specifically, we should choose $a < n + m$ such that $\ell_0 \sum_{i=0}^{\infty} \left(\frac{m-\ell_0}{n+m-a}\right)^i$ reaches $n + m - a$, the number of still unknown bits. Since

$$\ell_0 \sum_{i=0}^{\infty} \left(\frac{m - \ell_0}{n + m - a} \right)^i = \frac{1}{1 - \left(\frac{m-\ell_0}{n+m-a}\right)} \ell_0 = \left(\frac{n + m - a}{\ell_0 + n - a} \right) \ell_0,$$

this means that the optimal choice of a should be such that

$$\begin{gathered}\binom{n+m-a}{\ell_0+n-a}\ell_0 = n+m-a \\ \Updownarrow \\ a = n \text{ or } a = n+m.\end{gathered}$$

Note however that this only holds asymptotically, and it is expected that slightly more than n bits will need to be guessed to determine the remaining part of the key.

For RBS this suggests that an initial guess of around $n = 64$ key bits should be sufficient to determine all remaining 68 key bits. In order to determine how many more bits than n we should guess, a more careful analysis of the progression of the ℓ_i's is needed. In the following, we develop a conservative estimate:

Lemma 3. *Let a and ℓ_i be as in Lemma 1. Let $L(a, \ell_0) = (\ell_0, \ldots, \ell_t)$ be the series of ℓ_i defined from a and ℓ_0 s.t. t is the largest integer s.t. $\ell_t \geq 1$. When guessing a initial key bits, the* expected number of extra free bits obtained *is determined as $\sum_{j=0}^{t-1} \ell_j$ and the* expected Hamming weight *of these bits is determined as $\sum_{j=0}^{t} \ell_j$.*

Proof. This follows directly from the definition of ℓ_i and $L(a, \ell_0)$. □

Theorem 1. *Let a, ℓ_i and $L(a, \ell_0)$ be as in Lemma 3. Let $w(a)$ denote the* worst case expected *complexity of key recovery when a is the number of key bits initially guessed. Then*

$$w(a) = \sum_{\ell_0=\max\{0,a-64\}}^{\min\{68,a\}} \binom{a}{\ell_0} \binom{\max\{0, \lfloor 132 - a - \sum_{j=0}^{t-1} \ell_j \rfloor\}}{\max\{0, \lfloor 68 - \sum_{j=0}^{t} \ell_j \rfloor\}} \tag{5}$$

Proof. When initially guessing $k_{a-1} \| \cdots \| k_0$, the Hamming weight of this guess, ℓ_0, is bounded below by $\max\{0, a - 64\}$, because when $a > 0$, the Hamming weight must be positive by the pigeon-hole principle. The Hamming weight ℓ_0 is bounded above by either a or 68.

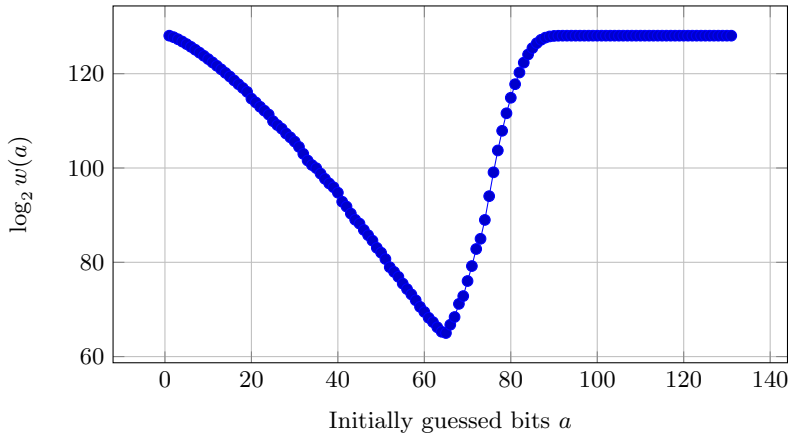
There are $\binom{a}{\ell_0}$ ways to distribute the ℓ_0 ones over $k_{a-1} \| \cdots \| k_0$. For each of these, the rightmost binomial coefficient of Eq. (5) gives the number of ways to place the remaining 1-bits among the unknown bits for this fixed combination of (a, ℓ_0). We take the sums of the ℓ_j as $\lfloor \sum_j \ell_j \rfloor$ for a conservative estimate of the complexity. Summing over all the possible ℓ_0 for a fixed a, the result follows. □

The key recovery attack. We summarize the resulting key recovery attack on RBS in Algorithm 5.

Algorithm 5: RBS-KEY-RECOVERY(a)

Data: Number of initial key bits to guess, a

$C \leftarrow \mathsf{RBS}(1)$
for $\ell_0 = \max\{0, 64 - a\}, \ldots, \min\{68, a\}$ **do**
  forall guesses *of* $k'_{a-1} \| \cdots \| k'_0$ with Hamming weight ℓ_0 **do**
    Let $L = (\ell_0, \ldots, \ell_t)$, where t is the largest integer s.t. $\ell_t \geq 1$
    $\Xi \leftarrow \max\{0, 132 - a - \sum_{j=0}^{t-1} \ell_j\}$; `/* # of bits yet unknown */`
    $\Phi \leftarrow \max\{0, 68 - \sum_{j=0}^{t} \ell_j\}$; `/* # of 1-bits remaining */`
    forall $\binom{\Xi}{\Phi}$ remaining candidates for $k'_{131} \| \cdots \| k'_{131-\Xi+1}$ **do**
      if $C = \mathsf{RBS}(1)$ under the key $k'_{131} \| \cdots \| k'_0$ **then**
        return $k'_{131} \| \cdots \| k'_0$ as the correct key k
      end
    end
  end
end



(a) Plot as a function of a

a	$\log_2 w(a)$
61	68.24
62	67.32
63	66.22
64	65.27
65	65.00
66	66.75
67	68.39
68	71.18
69	72.83
70	76.03

(b) Data points for the best values of a

Fig. 4: Expected worst time complexity for key recovery in RBS as a function of the number of bits initially guessed, denoted a

It remains to determine the number of key bits a that should be guessed initially. Figure 4 shows the base-2 logarithm of the expected worst case complexity $w(a)$. While Figure 4a shows a plot of $w(a)$ with $a \in \{1, \ldots, 131\}$, Figure 4b gives a numerical illustration of the best values for a giving the lowest complexity. From the data, we find that guessing $a = 65$ bits gives the lowest key recovery complexity of 2^{65}.

6 Conclusion

In this paper we presented key recovery attacks on three recent authenticated ciphers: AVALANCHE, Calico and RBS. The two former are submissions to the ongoing CAESAR competition for authenticated encryption schemes while the latter is a proposal for use in lightweight applications.

Common to all three designs is that they make use of solid primitives such as the AES, SipHash and ChaCha. We stress that the attacks presented here are purely structural, i.e. the weaknesses are present due to *the way the primitives are combined* and not the primitives themselves.

For AVALANCHE and Calico, the key recovery is possible due to the nonce being used as (part of) the key material, thus facilitating a key collision attack. For RBS, we used a guess-and-determine approach. In all cases, the key was recovered with a complexity of at most square root of the brute-force effort. Our attacks allows an adversary to perform universal forgeries in all three cases, and for AVALANCHE and RBS this extends to the ability to decrypt arbitrary ciphertexts.

Acknowledgments

The work has been supported in part by the Austrian government through the research program FIT-IT Trust in IT Systems (project 835919) and by the Austrian Science Fund (project P26494-N15).

References

1. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, March 2014. http://competitions.cr.yp.to/caesar.html.
2. Martin Ågren, Martin Hell, and Thomas Johansson. On hardware-oriented message authentication. volume 6, pages 329–336, 2012.
3. Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. volume 5, pages 48–59, 2011.
4. Basel Alomair. AVALANCHEv1. Submission to the CAESAR competition: http://competitions.cr.yp.to/round1/avalanchev1.pdf, 2014.
5. Jean-Philippe Aumasson and Daniel J. Bernstein. Siphash: A fast short-input PRF. In *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 489–508. Springer, 2012.
6. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, 2000.
7. Daniel J. Bernstein. ChaCha, a variant of Salsa20. Workshop Record of SASC 2008: The State of the Art of Stream Ciphers, 2008.
8. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The Keccak SHA-3 submission. Submission to NIST, 2011.
9. Morris J. Dworkin. SP 800-38C. recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. Technical report, Gaithersburg, MD, United States, 2004.
10. Morris J. Dworkin. SP 800-38D. recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. Technical report, Gaithersburg, MD, United States, 2007.
11. ISO 19772:2009. Information technology – Security techniques – Authenticated encryption, 2009.
12. Zahra Jeddi, Esmaeil Amini, and Magdy Bayoumi. A novel authenticated cipher for RFID systems. In *International Journal on Cryptography and Information Security*, volume 4, 2014.
13. Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *FSE 2000*, volume 1978 of *LNCS*, pages 284–299. Springer, 2000.
14. Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, 2011.
15. David A. McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, 2004.
16. Florian Mendel, Bart Mennink, Vincent Rijmen, and Elmar Tischhauser. A simple key-recovery attack on McOE-X. In *CANS 2012*, volume 7712, pages 23–31. Springer, 2012.
17. Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 275–292. Springer, 2014.
18. Jean-Jacques Quisquater and Jean-Paul Delescaille. How easy is collision search. new results and applications to DES, 1989.
19. Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, 2004.
20. Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *CCS 2001*, pages 196–205. ACM, 2001.
21. Christopher Taylor. The Calico family of authenticated ciphers version 8.