Cryptanalysis of the Counter mode of operation

Ferdinand Sibleyras

Internship Report – MPRI M2
Cryptanalysis of the counter mode of operation

Ferdinand Sibleyras, supervised by Gaëtan Leurent,
équipe SECRET, Inria

21 août 2017

General context

The counter mode of operation (CTR mode) is nowadays one of the most widely deployed modes of operation due to its simplicity, elegant design and performance. Therefore understanding more about the security of the CTR mode helps us understand the security of many applications used over the modern internet. On the security of the CTR mode, there is a well-known proof of indistinguishability from random outputs up to the birthday bound that is $\mathcal{O}(2^{n/2})$ encrypted n-bit blocks. This acts as a proof that no attack that can retrieve useful information about the plaintext exists with a lower complexity. In other words, any attack that breaks the confidentiality of the plaintext will require $\Omega(2^{n/2})$ blocks of ciphertext.

Research problem

While we have a lower bound on the complexity of a potential attack, it is not well understood how such attack would work and what would be its complexity not only in terms of data but also computationally (time and memory complexities). Most often the CTR mode is combined with the AES block cipher which acts on 128-bits blocks. In that setting, the birthday bound may appear sufficient to guarantee security in most if not all of today's internet applications as $2^{128/2} \times 128$ bits $= 256$ exbibytes, a comfortable margin. However if used alongside a 64-bits block cipher, like 3DES, the birthday bound stands at $2^{64/2} \times 64$ bits $= 32$ gibibytes, an amount of data quickly exchanged in today's internet. Moreover, the proof of indistinguishability says nothing at how quickly information on the plaintext is leaked when nearing the birthday bound. The goal of this internship is to devise efficient message recovery attacks under realistic assumptions and study their complexity to gain a better understanding of the security of the CTR mode.

Contribution

We give a concrete definition of the algorithmic problem naturally posed by the counter mode of operation, the *missing difference problem*, upon the resolution of which we can recover part of the unknown plaintext. Then we propose two algorithms to recover a block or more of secret plaintext in different settings motivated by real-life attacker models and compare the results with the work done by McGrew [McG12] on that same topic. We improve McGrew's results in two cases : the case where we know half of the secret plaintext, then we achieve time complexity of $\tilde{\mathcal{O}}(2^{n/2})$ compared to $\tilde{\mathcal{O}}(2^{3n/4})$ for McGrew's searching algorithm and the

case where we have no prior information on the secret where we achieve $\tilde{\mathcal{O}}(2^{2n/3})$ in time and query compared to the previous $\tilde{\mathcal{O}}(2^{n/2})$ queries and $\tilde{\mathcal{O}}(2^n)$ time.

This improvement allows better attack on the mode for a realistic attacker model than what had been described so far in the literature. In fact, we found out that the CTR mode does not offer much more security guarantees than the CBC mode as real attacks are of similar complexities. We described these attacks on the CTR mode and could even extend those to some message authentication code (MAC) schemes GMAC and Poly1305 based on the Wegman-Carter style construction.

Arguments supporting its validity

Not only do we provide some proofs for the asymptotic complexity but also implementations of these algorithms show that they are practical for blocks sizes $n \simeq 64$ bits and so are the associated attacks. These attacks rely on the repeated encryption of a secret under the same key so frequent rekeying will prevent those attacks from happening and thus we encourage any implementation of the CTR mode to force rekeying well before the birthday bound.

Summary and future work

We formalized an algorithmic problem that is naturally encountered in some cryptographic schemes, we called it the *missing difference problem*, and developed tools to solve it efficiently. These tools then help the cryptanalysis of different modes of operation and thus help understanding the security of popular real-world protocols.

Now we hope to publish these results and make users aware that using CTR is not much more secure than CBC (though CTR still offers other advantages). Especially when coupled with 64-bits block ciphers, it may not offer enough guarantee for most modern uses as 64-bits CBC mode was shown to be insecure in a recent work by Bhargavan and Leurent [BL16].

1 Introduction to block cipher modes of operation

Block ciphers. An n-bits block cipher can be understood as a family of permutations depending on a shared secret key k. It therefore describes a bijective function E with secret parameter k :

$$E_k : \{0,1\}^n \to \{0,1\}^n$$

Examples of popular block ciphers are DES ($n = 64$ bits) and AES ($n = 128$ bits).

Modes of operation. A mode of operation is an algorithm that specifies how to use a block cipher to achieve some cryptographic goals. This work focuses first on confidentiality modes of operation, such as CTR mode, that aim at hiding information about the plaintext but we will also discuss authenticity modes, such as CBC-MAC, that aim at providing a secure tag usually called *message authentication code* or MAC, and authenticated encryption modes, such as CCM or GCM, that are widely used over internet as it provides both unforgeability and confidentiality.

$$\begin{aligned} \text{Confidentiality mode} &: \{0,1\}^* \to \{0,1\}^* \\ \text{Authenticity mode (MAC)} &: \{0,1\}^* \to \{0,1\}^n \end{aligned}$$

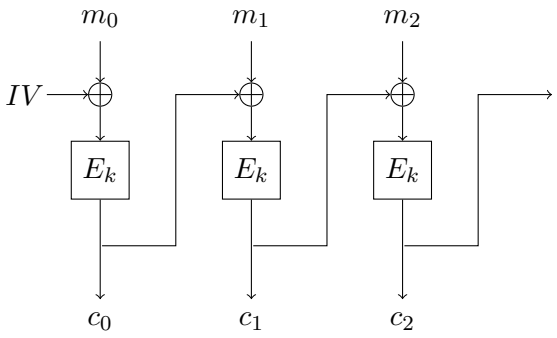


FIGURE 1 – Cipher Block Chaining mode CBC

Description. Historically CBC, CFB and OFB are three of the oldest standardized confidentiality modes of operation still in use today, especially CBC mode. They are all based on an initialization vector, IV, that is a public shared n-bits value that has some properties (unpredictable, nonce, random) according to the specifications of each mode. Those modes are algorithms and they are often described in the form of a diagram, see Figure 1 for the CBC mode, which shows the chaining rules between the IV, the blocks of plaintext m_i and blocks of ciphertext c_i. The blocks m_i are obtained by first padding the plaintext so that it reaches a size multiple of n and then slicing it in n-bits parts.

Collisions and birthday paradox. A mode of operation is usually described independently of the block cipher $E_k(\cdot)$ and in particular of its length n. However this n is actually an important security parameter that determines for how long one can use the same permutation $E_k(\cdot)$. For example many schemes are known to be secure only up to the encryption of a number of blocks less than the so-called birthday bound : $2^{n/2}$ blocks. For those schemes it is strongly recommended to renew the key k, therefore changing the permutation, well before the encryption of $2^{n/2}$ blocks of ciphertexts. It is typically the case for the aforementioned confidentiality modes. For example the CBC mode will leak information on the plaintext as soon as we observe a collision in the ciphertext blocks. Indeed if one observes that $c_{i+1} = c_{j+1}$ for some i, j then, as $E_k(\cdot)$ is bijective, we have $c_i \oplus m_i = c_j \oplus m_j$ that leaks the XOR of two blocks of plaintexts : $m_i \oplus m_j = c_i \oplus c_j$. And a collision is typically bound to occur with probability $\Omega(1)$ after observing $\mathcal{O}(2^{n/2})$ blocks of ciphertexts, that is the birthday paradox.

This work. In this report we first introduce the confidentiality mode CTR and its security proof using distinguishers. Then we study message recovery attacks for CTR mode formalizing the *missing difference problem* that need to be solved and present the state of the art on the topic. We propose efficient algorithms to solve this problem and then go back to devising real attacks on CTR mode and extend the attacks on authenticated modes of operation GMAC and Poly1305. At last we give implementations' results and proofs to support our claims.

2 The counter mode

The security of block ciphers is studied with cryptanalysis, with classical techniques such as differential [BS91] and linear [Mat94] cryptanalysis, dedicated techniques like the SQUARE

attack [DKR97], and ad-hoc improvements for a specific target. This allows to evaluate the security margin of a block cipher, and today we have a high confidence that AES or Blowfish are as secure as a family of pseudo-random permutations with the same parameters (key size and block size).

On the other hand, modes of operation are mostly studied with security proofs, in order to determine conditions where using a particular mode of operation is safe. However, exceeding those conditions doesn't imply that there is an attack, and even when there is one, it can range from a weak distinguisher to a devastating key recovery. In order to get a better understanding of the security of modes of operations, we must combine lower bound on the security from security proofs, and upper bounds from attacks.

More precisely, the counter mode is defined as $c_i = E(i) \oplus m_i$ (see diagram Figure 2). There are no collisions in the inputs/outputs of E, but this can actually be used by a distinguisher. Indeed, if an adversary has access to $2^{n/2}$ known plaintext/ciphertext pairs, she can recover $E(i) = c_i \oplus m_i$ and detect that the values are unique (because E is a permutation), while collisions would be expected with a random ciphertext. Both attacks have the same complexity, and show that the corresponding proofs are tight. However, the loss of security is quite different : an attacker can recover some bits of information from the attack against CBC (as shown in practice in [BL16]), but the attack against the counter mode hardly reveals any useful information.

In general, there is a folklore belief that the leakage of the CTR mode is not as bad as the leakage of the CBC mode. For instance, Ferguson, Schneier and Kohno wrote [FSK11, Section 4.8.2] :

> CTR leaks very little data. [...] It would be reasonable to limit the cipher mode to 2^{60} blocks, which allows you to encrypt 2^{64} bytes but restricts the leakage to a small fraction of a bit.
>
> When using CBC mode you should be a bit more restrictive. [...] We suggest limiting CBC encryption to 2^{32} blocks or so.

The main goal of this report is to study attacks against the counter mode beyond the simple distinguisher given above. We consider generic attacks that work for any instance of the block cipher E, and assume that E behaves as a pseudo-random permutation. We will focus on the asymptotic complexity of attacks, using the Big-O notation $\mathcal{O}()$, and the Soft-O notation $\tilde{\mathcal{O}}()$ (ignoring logarithmic factors).

Related works. While there are few generic attacks known against encryption modes, many interesting attacks have been found against authentication modes. In 1995, Preneel and van Oorschot [Pv95] gave a generic collision attack against all iterated message authentication codes (MACs), leading to existential forgeries with complexity $\mathcal{O}(2^{n/2})$. Later, a number of more advanced generic attacks have been described, with stronger outcomes than existential forgeries, starting with a key-recovery attack against the envelop MAC by the same authors [Pv96]. In particular, a series of attack against hash-based MAC [LPW13, PW14, GPSW14, DL14] led to universal forgery attacks against long challenges, and key-recovery attacks when the hash function has an internal checksum (like the GOST family). Against PMAC, Lee *et al.* showed a universal forgery attack in 2006 [LKS+06]. Later, Fuhr, Leurent and Suder gave a key-recovery attack against the PMAC variant used in AEZv3 [FLS15]. Issues with GCM authentication with truncated tags were also pointed out by Ferguson [Fer05].

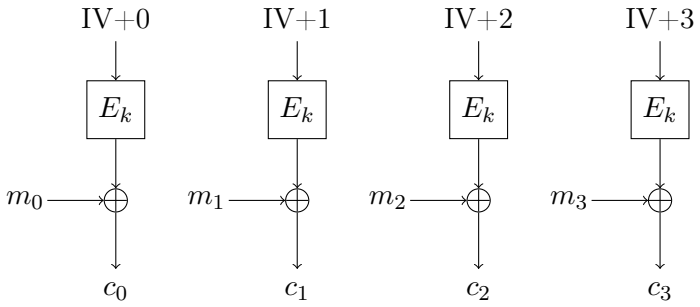


FIGURE 2 – CTR mode

3 Impossible Plaintext Attack on CTR mode

Even though it is not one of the four most classical modes of operation (CBC, CFB, ECB, OFB), the Counter mode of operation, or CTR, proposed by Diffie and Hellman, is just as old and since 2001 is also part of NIST's recommendations. CTR's parallelizability, speed and simple design makes it a widely popular mode of operation nowadays used for example as a basis for the Galois/Counter mode (GCM). This led Phillip Rogaway to write in an evaluation of different privacy modes of operation talking about CTR : "Overall, usually the best and most modern way to achieve privacy-only encryption" [Rog11]. Indeed CTR only offers privacy protection and has no guarantee regarding authentication or non-malleability when used by itself but is a most useful building block when designing schemes that, like GCM, achieve all these properties. A well known fact about CTR is that it is indistinguishable from random output up to the birthday bound in a chosen plaintext attack model *i.e.* coupled with a secret permutation the CTR mode of operation is IND-CPA as long as we restrict the number of queries an order below $2^{n/2}$ where n is the block size [BDJR97]. However, it is not always clear how we can effectively recover any secret information as we come near the bound.

3.1 Setting and notations

In the following we assume that the counter mode is implemented such that the input to the block cipher never repeats. For simplicity we consider a stateful variant of the counter mode with a global counter that is maintained across messages (as shown in Figure 2) :

$$c_i = E_k(IV + i) \oplus m_i,$$

where E_k is an n-bit block cipher, IV the initial value of the counter, m_i an n-bit block of plaintext and c_i an n-bit block of ciphertext.

Our attacks do not depend on the details of how the input to the block cipher is constructed [1], but only require that all inputs are different. Note that some variants of the counter mode have repetitions in the block cipher input [2], but this gives easy attacks because repetitions leak the xor of two plaintext blocks.

Throughout the attack the key k and the IV will be invariant so we will write $E_k(IV + i)$ as K_i to represent the i^{th} block of CTR keystream. We can immediately notice that if we

1. For instance, some variants concatenate a per-message nonce and a counter within a message.
2. For instance, using a random n-bit initial count for each message and incrementing it for every block in a message leads to collisions after roughly $2^{n/2}$ messages

have partial knowledge of the plaintext, for every block m_i that we know we can recover the associated K_i as $c_i \oplus m_i = K_i$. Assume further that we have access to the repeated encryption E_j of some secret S so that $E_j = K_j \oplus S$. The first property of the CTR mode is that $E_k(\cdot)$ being a permutation, the keystream K_i never repeats thus we have the following inequalities :

$$\begin{aligned} \forall i \neq j : K_i \neq K_j \quad &\Rightarrow \quad K_i \oplus K_j \neq 0 \\ &\Rightarrow \quad K_i \oplus K_j \oplus S \neq S \\ &\Rightarrow \quad K_i \oplus E_j \neq S \end{aligned}$$

From now on we will always assume that we can observe and collect lists of many K_i and E_j and use them with the previous inequality to recover S. This setting is similar to the practical attack Sweet32 on CBC mode mounted by Bhargavan and Leurent using repeated encryption of an authentication token to obtain many different ciphertext blocks for the same secret information [BL16].

Formally, let $\mathcal{K} \subseteq \{0,1\}^n$ be the set of observed keystream blocks, $\mathcal{E} \subseteq \{0,1\}^n$ the set of observed encryptions and $\Phi \subseteq \{0,1\}^n$ the set of possible secret. We redefine the missing difference algorithmic problem in terms of set as :
Given two sets $\mathcal{K}$ and $\mathcal{E}$, find the value $S \in \Phi$ such that :

$$\forall (k,e) \in \mathcal{K} \times \mathcal{E},\ S \neq k \oplus e\ .$$

3.2 Previous work

An attack can only be carried to the end if the secret S is the only value in Φ such that $\forall (k,e) \in \mathcal{K} \times \mathcal{E},\ S \neq k \oplus e$ or else it will be indistinguishable from the other values that satisfy the same condition (those values could have produced the same output with same probability). The coupon collector's problem predicts that N out of N different coupons are found after $N \cdot H_N \simeq N \ln N$ draws (with H_N the harmonic number) assuming uniform distribution of the draws. In our case we will assume that every differences $k \oplus e$ are independents and uniformly distributed over $\{0,1\}^n \backslash S$, which is very close to what we expect and observe in practice. To carry the attack to the end we require to collect $N = |\Phi| - 1$ differences thus we will need $\mathcal{O}(|\Phi| \log |\Phi|)$ "draws". A draw is a couple (k,e) s.t. $k \oplus e \in |\Phi|$, else we discard it, which happens with probability $(|\Phi| - 1)/(2^n - 1)$. Therefore we need to observe enough data to have $|\mathcal{K}| \cdot |\mathcal{E}|$ in the order of $\mathcal{O}(2^n \log |\Phi|)$; may be achieved by having both sets in the order of $\mathcal{O}(2^{n/2}\sqrt{\log |\Phi|})$. This size of the observed sets can be understood as the query complexity, that is the number of network data requests the attacker will have to intercept in order to carry out the attack. Notice that even for $|\Phi| = \mathcal{O}(2^n)$, $|\mathcal{K}| = |\mathcal{E}| = \mathcal{O}(\sqrt{n} \cdot 2^{n/2})$ is quite close to the theoretical lower bound of $\mathcal{O}(2^{n/2})$ given by the distinguishing attack and the security proof for the CTR mode. So we already have a "good" data or query complexity but then the issue is about solving the algorithmic problem and recovering the actual value of S.

A first approach consists in computing all the impossible values of S from the large set of $\mathcal{K} \times \mathcal{E}$ and discard any new value we encounter as impossible until there's only one possible plaintext left. This is algorithm 1. This approach works but this requires to actually compute $\mathcal{O}(2^n \log |\Phi|)$ values and maintain in memory a sieve of size $|\Phi|$. In the case where the key size is equal to the block size n, like AES-128, then this attack is actually worse than a simple exhaustive search of the key that would require only a few blocks of keystream, $\mathcal{O}(2^n)$ computations and near to no memory. In a 2012 work, McGrew first described this sieving algorithm and noticed that the sieving is inefficient with a small set Φ. Therefore he proposed

Algorithm 1 Simple sieving algorithm

Input: K, E, Φ
Output: Φ s.t. $\forall S \in \Phi\, \forall i, j : K[i] \oplus E[j] \neq S$
  for k in K **do**
    for e in E **do**
      Remove $(k \oplus e)$ from Φ ;
    end for
  end for
  return Φ

Algorithm 2 Searching algorithm

Input: K, E, Φ
Output: Φ s.t. $\forall S \in \Phi\, \forall i, j : K[i] \oplus E[j] \neq S$
  Store E so that operation $\in$ is efficient.
  for ϕ in Φ **do**
    for k in K **do**
      if $(\phi \oplus k) \in E$ **then**
        Remove ϕ from Φ ;
        break ;
      end if
    end for
  end for
  return Φ

a second algorithm, algorithm 2, to test and eliminate remaining values one by one. This algorithm loops over Φ and $\mathcal{K}$ to efficiently test whether $\phi \oplus k \in \mathcal{E}$; if yes then we sieve the value ϕ out of Φ, if not continue.

The algorithms act on a given sieving set Φ to reduce it so one can switch at any time from one algorithm to the other in order to reduce the searching space as quickly as possible. This results in a hybrid algorithm [McG12]. This hybrid algorithm surely improves the attack but it is easy to see that each impossible values will require some computations and therefore will require at least $\Omega(|\Phi|)$ computations. Still, if $|\Phi|$ is small, then we directly run the second part of the hybrid algorithm, algorithm 2, which loops over Φ and $\mathcal{K}$ that is a complexity of $\mathcal{O}(|\mathcal{E}|+|\mathcal{K}|\cdot|\Phi|)$. Assuming we carry out the attack to the end (implies $|\mathcal{K}|\cdot|\mathcal{E}| = \mathcal{O}(2^n \log|\Phi|)$) and that we optimize the sizes of the observed set, we force $|\mathcal{E}| = |\mathcal{K}| \cdot |\Phi|$ to obtain an overall optimized complexity of $\mathcal{O}(2^{n/2}\sqrt{|\Phi| \log|\Phi|})$ in both time and queries. Therefore for small Φ this algorithm is satisfying as the complexity gets close to the birthday bound $\tilde{\mathcal{O}}(2^{n/2})$.

However this is still interesting and starting from these observations we will show how we can recover a block, or more, of secret information by devising two quicker attacks that try to avoid big exhaustive searches : one that relies on some additional assumptions on the plaintext to reduce the set Φ to a vector space of smaller size $|\Phi| = 2^{n-z}$ and one in the case where $\Phi = \{0,1\}^n$ that uses greater query complexity of $\tilde{\mathcal{O}}(2^{2n/3})$ to lessen the computation and memory usage.

4 Efficient Algorithms

In this section we will propose two solutions to solve the missing difference algorithmic problem in an efficient way for two different settings inspired by real applications. On the basis of these algorithms, we will be able to describe attacks on the CTR mode and more in the next section.

4.1 Known prefix sieving

Algorithm 3 Known prefix sieving algorithm

Input: n, $z < n$, K, E, $\Phi \subset \{0\}^z \times \{0,1\}^{n-z}$
Output: Φ s.t. $\forall S \in \Phi \, \forall i,j : K[i] \oplus E[j] \neq S$

```
h_K, h_E ← Empty hash tables.
preK ← {}
for k in K do
    pref||suff ← k_{0..(z-1)}||k_{z..n}
    h_K[pref] ← h_K[pref] ∪ {suff}
    preK ← preK ∪ {pref}
end for
for e in E do
    pref||suff ← e_{0..(z-1)}||e_{z..n}
    h_E[pref] ← h_E[pref] ∪ {suff}
end for

for p in preK do
    if h_E[p] ≠ ∅ then
        for v_k in h_K[p] do
            for v_e in h_E[p] do
                Remove 0̄||(v_k ⊕ v_e) from Φ ;
            end for
        end for
    end if
end for
return Φ
```

In the case where Φ is an affine subspace of $\{0,1\}^n$ of dimension $n-z$ for some natural $z < n$ then we can modify $\mathcal{K}$ and $\mathcal{E}$ accordingly so that the solution S belongs to the linear subspace $\Phi = \{0\}^z \times \{0,1\}^{n-z}$. Concretely let Φ be a known affine space (or a subset of one) then we can deduce a bijective affine function ϕ that maps Φ unto $\{0\}^z \times \{0,1\}^{n-z}$ then the reduction unfolds :

$$\begin{aligned} \forall (k,e) \in \mathcal{K} \times \mathcal{E},\ S \neq k \oplus e \quad &\Leftrightarrow \quad \phi(S) \neq \phi(k \oplus e), \text{ as } \phi \text{ is a bijection.} \\ &\Leftrightarrow \quad \phi(S) \neq \phi(k) \oplus \phi(e) \oplus \phi(\overline{0}), \text{ as } \phi \text{ is affine} \\ &\Leftrightarrow \quad \phi(S) \neq \phi(k) \oplus (\phi(e) \oplus \phi(\overline{0})) \end{aligned}$$

Thus finding $\phi(S)$ is equivalent to finding S, as ϕ is easily invertible. We get to this form simply by applying the transformations :

$$\begin{aligned}\Phi' &\leftarrow \{0\}^z \times \{0,1\}^{n-z}\\ \mathcal{K}' &\leftarrow \{\phi(k) \mid k \in \mathcal{K}\}\\ \mathcal{E}' &\leftarrow \{\phi(e) \oplus \phi(0) \mid e \in \mathcal{E}\}\end{aligned}$$

Equivalently, we can say we reduced the problem to the case where we know the prefix of S i.e. we look for a solution that starts with z zeros ; we can then perform the known prefix sieving algorithm 3.

The algorithm is quite straightforward ; it looks for a prefix collision before sieving in the same way as before to recover S. The complexity will be function of the dimension $n-z$; the sieving will require $\mathcal{O}(2^{n-z})$ memory and $\mathcal{O}((n-z)\cdot 2^{n-z})$ XOR computations in expectation while looking for collisions only requires to store the prefix keys and go through one of the set. Looking for collisions allows us to skip the computations of many pairs (k,e) that would be irrelevant as $k \oplus e \notin \Phi$. The total optimized complexity (with balanced sets $\mathcal{K}$ and $\mathcal{E}$) to recover an $n-z$ bits secret with this algorithm is :

$\mathcal{O}((n-z)\cdot 2^{n/2})$	queries
$\mathcal{O}(2^{n-z} + n(n-z)\cdot 2^{n/2})$	bits of memory (sieving + queries)
$\mathcal{O}((n-z)\cdot 2^{n-z} + (n-z)\cdot 2^{n/2})$	computations (sieving + collisions searching)

As we can see from the complexity, when $z = 0$ this is the naive algorithm with its original complexity. When z nears n, this performs similarly to McGrew's searching algorithm i.e. the cost of looking for collisions (or storing $\mathcal{E}$ so that the search is efficient) will dominate the overall cost of the algorithm therefore the time and query complexity will match. In fact, we gain the most of this algorithm for intermediate values of z and as we will see in Section 5 the parameter $z = n/2$ may be the most pertinent parameter to mount a practical attack on the CTR mode which uses repetitions of this algorithm to recover n or more bits of secret.

4.2 Fast XOR-counter

Algorithm 4 Fast XOR-counters computation

Input: $K, E, n' \leq n$
Output: C_X s.t. $\forall c < 2^{n'} : C_X[c] = |\{(i,j) : K[i] \oplus E[j] = c \parallel *\}|$
  $C_K, C_E, C_X \leftarrow$ arrays of $2^{n'}$ integers initialized to 0 ;
  for k in K **do**
    Increment $C_K[k_{0..(n'-1)}]$
  end for
  for e in E **do**
    Increment $C_E[e_{0..(n'-1)}]$
  end for
  Perform fast Walsh-Hadamard transform in-place : FWHT(C_K) ; FWHT(C_E) ;
  for $c = 0 ; c < 2^{n'}$ **do**
    $C_X[c] \leftarrow C_K[c] \cdot C_E[c]$
  end for
  FWHT(C_X) ;
  return C_X

Algorithm 5 Sieving using fast XOR-counting

Input: $K, E, n' \leq n$
Output: S s.t. $\forall i, j : K[i] \oplus E[j] \neq S$
  $X \leftarrow$ Run the fast XOR-counters on $(\mathcal{K}, \mathcal{E}, n')$ using algorithm 4
  $\Phi \leftarrow \{\}$
  while $\Phi = \{\}$ **do**
    $s \leftarrow \text{argmin}_i\, C_X[i]$ (size n' bits)
    $C_X[s] \leftarrow +\infty$
    $\Phi \leftarrow \{0\}^{n'} \times \{0,1\}^{n-n'}$
    $\mathcal{E}' \leftarrow \{e \oplus (s||\overline{0}) \mid e \in \mathcal{E}\}$
    Run known prefix sieving algorithm 3 using $(n, n', \mathcal{K}, \mathcal{E}', \Phi)$.
  end while
  $s' \leftarrow$ sole value left in Φ.
  return $(s||\overline{0}) \oplus s'$

Going back to the hard setting and assumptions where the secret is always encrypted as a whole block with no prior information on it, $|\Phi| = 2^n$, there is a way to avoid exhaustive complexity and retrieve the secret S with high probability when the query complexity is higher. The idea is to look at $2n/3$ bit's positions, on those positions the XOR differences $(k \oplus e)$ can take any values but there exists a statistical bias from the fact that $k \oplus e$ is uniformly distributed over $\{0,1\}^n \backslash S$. *I.e* on fixed $2n/3$ bit's position there's is slightly less chance a XOR difference corresponds to S than to any other values. So if we simply count how many times each combination on those $2n/3$ bits occur for every couple (k, e) then the combination that appears the less is the most likely to correspond to S. To statistically discriminate the "good" combination (corresponding to S) against every other "bad" combinations on $2n/3$ bit's positions will cost an increased query complexity of $\mathcal{O}(\sqrt{n} \cdot 2^{2n/3})$ according to the analysis in Section 8.1.

Now we show an algorithm to quickly count the number of occurrences for each combination. First we count over the $2n/3$ chosen bits the number of occurrences in $\mathcal{K}$ and $\mathcal{E}$ in two separated lists of counters of size $2^{2n/3}$, one counter for each $2n/3$ bits combination. Let C_K and C_E be the counters of $\mathcal{K}$ and $\mathcal{E}$ respectively. Then we need to compute C_X the same counters' list but for all the elements in $\mathcal{K} \times \mathcal{E}$. We observe that :

$$C_X[i] = \sum_j C_K[j] \cdot C_E[i \oplus j]$$

which is a form of convolution that can be quickly computed using the Fast-Walsh-Hadamard transform in the same way we use the classical Fourier transform. This is algorithm 4. Because $k \oplus e$ can be any value except S, we hope that the lowest counter corresponds to the value of S over those $2n/3$ bits :

$$S_{2n/3bits} \stackrel{?}{=} \underset{i}{\text{argmin}}\, C_X[i]$$

The probability that this is true depends on the number of pairs (k, e) we could gather. We found that this probability gets to $\Omega(1)$ when we have lists of size $\mathcal{O}(\sqrt{n} \cdot 2^{2n/3})$, see Section 8.1 for the detailed proof. When we have a good guess, we can perform the previous known prefix sieving algorithm over the $n/3$ bits left (*i.e.* we look for S in an affine space of

dimension $n-z=n/3$) to either find the whole value for S or, if every values have been sieved out, conclude the chosen combination does not match with the true value of S and continue. This is algorithm 5.

The complexity to recover a whole n bits secret with fast XOR counting is as follow :

$\mathcal{O}(\sqrt{n} \cdot 2^{2n/3})$	queries
$\mathcal{O}(n \cdot 2^{2n/3}) + \mathcal{O}(n\sqrt{n} \cdot 2^{n/2})$	bits of memory (counters + sieving)
$\mathcal{O}(n \cdot 2^{2n/3}) + \mathcal{O}(n\sqrt{n} \cdot 2^{n/2})$	computations (fast Walsh-Hadamard + sieving)

For the memory complexity, notice that we don't need to store all the blocks we requested but simply to increment a counter. We only need to store enough blocks for the second part of the algorithm so that the sieving yields a unique result. However how big are those counters ? Initially for C_K and C_E they are quite small, $\sqrt{n}$ in expectation, but C_X will have ultimately much bigger entries, $n \cdot 2^{2n/3}$ in expectation, so this will require $\mathcal{O}(n)$ bits to store each entry thus $\mathcal{O}(n \cdot 2^{2n/3})$ bits of memory complexity.

5 Attacks using known prefix sieving

Block splitting. We often assume that the secret is encrypted in its own block but a mode of operation is abstractly used as a way to encrypt messages of any length, or even a stream of messages and will typically cut the message into blocks independently of their contents as they will be reconstructed later on the receiving side before reaching the application layer. Therefore, if we have incomplete knowledge of the plaintext, we could very well have known information mixed with a secret inside the same block. For example we could have a fixed header followed by critical information, a plaintext of the form : "SECRET_INFO :<$secret\_info$>" in a predictable position then a mode of operation could cut the message as ... {"SECRET_I"} {"NFO :"$||S_1$} {$S_2||S_3$} ... with $S = S_1||S_2||S_3$ the secret information and encrypt those blocks as shown in Table 1. Here we see a known and predictable header that is encrypted on the same block as a part of the secret. We can use this to apply the known prefix sieving algorithm multiple time and ultimately recover the secret S as a whole.

Plaintext	SECR ET_I	NFO : S_1	S_2 S_3	S_4 ...
		$\oplus$		
Keystream	K_1	K_2	K_3	K_4
		$\oplus$		
Known part	SECR ET_I	NFO : 0	0 0	0 0
		=		
Exploitable data	K_1	$K_2 \oplus (\overline{0}\|\|S_1)$	$K_3 \oplus (S_2\|\|S_3)$	$K_4 \oplus (S_4\|\|...)$

TABLE 1 – Example of how a plaintext is cut into blocks, encrypted in a CTR mode and transformed to be exploited in known prefix sieving algorithm. $S = S_1||S_2||S_3||S_4$ the secret information. Known information in blue, unknown information in red.

In the same way we can recover keystream blocks from known plaintexts, we can recover part of the keystream blocks used to encrypt the secret. In the example of Table 1, we recover

(1)	First round	SECR	ET_I	NFO :	S_1	S_2	S_3	S_4 ...
(2)	Inject "JAVA"	JAVA	SECR	ET_I	NFO :	S_1	S_2	S_3 S_4
(3)	Reuse data from (1)	SECR	ET_I	NFO :	S_1	S_2	S_3	S_4 ...
(4)	Reuse data from (2)	JAVA	SECR	ET_I	NFO :	S_1	S_2	S_3 S_4

TABLE 2 – Example of an attack on two blocks secret $S = S_1||S_2||S_3||S_4$. Each step performs the known prefix sieving algorithm. Known information in blue, unknown information in red, attacked information in yellow.

$K_1 \in \mathcal{K}$ and $K_2 \oplus \{\overline{0}||S_1\} \in \mathcal{E}$ thus our missing difference algorithmic problem is well defined with those sets $\mathcal{K}$, $\mathcal{E}$ and $\Phi = \{0\}^z \times \{0,1\}^{n-z}$ with $n-z$ the size of S_1. This is therefore the setting where Φ is an affine subspace of dimension $n-z$ and we use the known prefix sieving algorithm 3 described in Section 4.1.

With this we recover a part of the secret, now to recover a whole block S we repeat the same process by observing, either naturally via the protocol or by actively manipulating part of the plaintext, the next secret bits encrypted in the same manner with known information, which is always possible since we now know part of S. In our example Table 2, after recovering S_1, we inject in the plaintext the string "JAVA" to push around how the message is cut into blocks. We repeat the algorithm with $K_1 \in \mathcal{K}$, $K_2 \in \mathcal{K}$, $K_3 \oplus \{\overline{0}||S_2\} \in \mathcal{E}$. Notice that $\mathcal{E}$ must be reset between two rounds while $\mathcal{K}$ keeps on growing. This scenario where we shift around the secret is inspired by the practical attacker model BEAST for HTTPS [DR11].

The complexity to recover an n bits secret block by slices of size $n-z$ is as follow :

$\mathcal{O}(\sqrt{n} \cdot 2^{n/2})$	queries
$\mathcal{O}(n\sqrt{n} \cdot 2^{n/2}) + \mathcal{O}(2^{n-z})$	bits of memory (queries + sieving)
$\mathcal{O}(n\sqrt{n} \cdot 2^{n/2}) + \mathcal{O}(n \cdot 2^{n-z})$	computations (sorting + sieving)

Interestingly, it appears there is no price to pay in term of query complexity to perform this attack. Indeed even in the extreme, but unpractical, case where we can slice the secret bit by bit we found that the number of queries needed is of the same order though the proof is different (proof Section 8.2). Then any z is simply a combination of the two extreme cases $z = 0$ and $z = n-1$ which have same asymptotic query complexity.

Multiple blocks secret. To extend this attack on multiple blocks, we need just realize that after finding the first block then we probably have enough information to deduce the second block just by sieving. In fact, we already have more information to attack the second block than we had for the first one as knowledge of raw keystream blocks $\mathcal{K}$ keeps increasing and we can reuse the data of previous rounds to attack the next block (see Table 2) so that the size of $\mathcal{E}$ is as big as it was for the previous round while $\mathcal{K}$ is bigger then without observing any new queries we already have a good probability of performing the attack to the end.

Practical parameters. Following on the complexity analysis, $z = n/2$ is a most natural choice to have the sieving part lighter than the sorts and data retrieval without making the

practical implementation too complex. For $n = 64$ bits this complexity is definitely tractable and we could simulate the attack on locally encrypted CTR mode (see Section 7.1). Therefore we have an algorithm with query complexity of $\mathcal{O}(\sqrt{n}2^{n/2})$ to recover repeated encryption of a secret over multiple blocks in the BEAST attacker model.

Other attacks. There are surprisingly many settings where unknown plaintext will naturally lie in some known affine subspace and makes the application of the known prefix sieving algorithm straightforward. For instance a credit card number (or any number) could be encoded in 16 bytes of ASCII then encrypted ; because in ASCII the encoding of any digit starts by `0x3` (`0x30` to `0x39`) we know the values and positions for half of the bits of the plaintext. This is a case where $z = n/2$. That the credit card number has 64 bits unknown to the attacker is not relevant for this attack, we only care about the proportion of known bits in the plaintext to reduce the searching space. Other examples are information encoded by `uuencode` that uses ASCII values `0x20` to `0x5F` or HTML authentication cookies used by `wikipedia.org` encoding lower case letters and digits in Base64 ; in those cases the first two bits of each bytes are known to be `0` therefore $z = 3n/4$.

Counter-measures. As for many modes of operation, the common wisdom to counter this kind of attacks asks for rekeying before the birthday bound, i.e. before $2^{n/2}$ blocks. However rekeying too close to the birthday bound may not be enough. For example let's consider an implementation of a CTR based mode of operation that rekeys every $2^{n/2}$ blocks, then in the worst case scenario every session leaks $2^{n/2-1}$ values for $\mathcal{K}$ and $\mathcal{E}$ that means 2^{n-2} couples that each have roughly probability $2^{-n/2}$ of colliding on $z = n/2$ bits. So on average $2^{n/2-2}$ impossible values are leaked for the first half of S_1 each session and sieved out of Φ. Thus after $2n$ sessions there's a good probability of recovering S_1, same for S_2 that is a total of $4n \cdot 2^{n/2}$ data complexity compared with $\sqrt{n} \cdot 2^{n/2}$ in the total absence of rekeying. However rekeying every $2^{n/2-16}$ blocks makes the data complexity goes up to $2^{33}n$ sessions that is $2^{17}n \cdot 2^{n/2}$ blocks to recover S_1, twice more for the whole S and the following blocks. Notice that the security gain is comparable with the CBC mode where rekeying every $2^{n/2-16}$ blocks forces the attacker to intercept 2^{32} sessions, compared with $2^{34}n$ for the CTR mode it is a factor $4n$ in the query complexity. That is why frequent rekeying is an effective counter-measure against this threat as long as one keeps a sufficient margin away from the birthday bound in accordance with the guidelines of Luykx and Paterson [LP16].

6 Indivisible secret

6.1 Counter mode

Use of the fast XOR-counter algorithm to recover one block of CTR mode plaintext is quite straightforward and requires fewer assumptions than the previous attack at a cost of a greater complexity. Indeed here the attacker is completely passive and observes one type of encryption of S without the need of injecting anything or playing with the blocks. First gather many $k \in \mathcal{K}$ from ciphertexts of known plaintexts and many $e \in \mathcal{E}$ from the secret's encryptions to run the algorithm 5. As mentioned before, when gathering values for $\mathcal{K}$ and $\mathcal{E}$, instead of keeping the whole block one can simply increment the respective counter as he sees the blocks pass by and only keep enough values for a successful sieving in the second part of the algorithm.

When attacking multiple blocks secret, the counter for $\mathcal{K}$ will be common for all the blocks and each block will have its own set $\mathcal{E}$ and associated counters ; because we attack a whole block, this is easily parallelizable over many blocks in contrast with the previous attack that has to go step by step.

As we can see on the tests we ran for this algorithm in Section 7.2, it has a very good probability of success for the chosen complexity parameters showing there are no big "hidden constant" in the complexity analysis. Moreover they are multiple improvements we could think of to improve the success probability of the algorithm, for example it is independent of the position of the $2n/3$ bits we choose for counting, so having multiple instances of the algorithm that count on different positions may be interesting to avoid the few bad cases we could observe in the simulations where the counter on the secret value grows abnormally high and gets hidden in all of the other counters.

The value of $2n/3$ has been chosen to balance the query and post-processing complexities though the algorithms are stated with this value as a parameter. In general if we choose to count over n' bits we will need $\mathcal{O}((n-n') \cdot 2^{2n-n'})$ values so a query complexity of at least $\mathcal{O}(\sqrt{n-n'} \cdot 2^{n-n'/2})$ and memory/computation complexities of $\mathcal{O}(n' \cdot 2^{n'})$ for the fast XOR-counter algorithm. There is therefore a trade-off between the exponents $n - n'/2$ and n' best balanced at $n' = 2n/3$.

6.2 Wegman-Carter MAC construction

Because this algorithm requires less assumptions, we can adapt the attack for use on other modes of operation based on CTR and particularly on Wegman-Carter type of constructions for MAC. Wegman-Carter MACs use a keyed permutation E and a keyed universal hash function h, it takes as input a message M and a nonce N and is defined as follow, with k_1 and k_2 two private keys :

$$\mathrm{MAC}(N, M) = h_{k1}(M) + E_{k2}(N)$$

Again, the construction requires that all block ciphers input are different. Then the attack consists in observing for two known messages M and M' many values of $\mathrm{MAC}(N, M) \in \mathcal{K}$ and $\mathrm{MAC}(N', M') \in \mathcal{E}$ all using unique nonces then solve the missing difference problem to recover $h_{k1}(M) - h_{k1}(M')$ as we know that $\forall N \neq N' : E_{k2}(N) - E_{k2}(N') \neq 0$. It is often sufficient to know this difference and the two messages M and M' to recover the key $k1$. We give two concrete examples.

Galois/Counter Mode. In the GCM mode, we are allowed to send authenticated data that will be signed but not encrypted. In the case we send one block of authenticated data A alone, then the GCM mode will sign it as :

$$\mathrm{MAC}(N, A) = A \cdot H^2 \oplus H \oplus E_k(N)$$

with H the hash key and $(\cdot)$ the multiplication in a Galois Field defined by a public polynomial. So for two different blocks of authenticated data A and A' we collect $\mathcal{O}(\sqrt{n} \cdot 2^{2n/3})$ signatures and perform the Fast-XOR algorithm to recover $A \cdot H^2 \oplus H \oplus A' \cdot H^2 \oplus H = (A \oplus A') \cdot H^2$. We known $A \oplus A'$ and the field is known so we invert that value and recover H^2 then compute the square root and recover the hash key H.

Poly1305. This scheme has been described for 128-bits blocks [Ber05]. It uses a keyed permutation (usually AES), and a hash function whose key, r, has 106 free bits (22 bits of the key are set to 0, including in particular the 4 most significant ones). It also defines c_i as some 129-bits padding of each block of the message M so it is easily predictable. Thus for the nonce N and message M of length q the Poly1305's MAC is defined as :

$$T(M,N) = (((c_1 r^q + c_2 r^{q-1} + ... + c_q r) \mod 2^{130} - 5) + E_k(N)) \mod 2^{128}$$

applying the same strategy for two different messages M and M' we recover the missing difference

$$(((c_1 - c'_1) r^q + (c_2 - c'_2) r^{q-1} + ... + (c_q - c'_q) r) \mod 2^{130} - 5) \mod 2^{128}$$

and in the case where we chose M and M' such that $c_i - c'_i = 0$ and $c_q - c'_q = 1$, as we know that, by design, $r < 2^{124}$, then the value recovered is simply the hash key r.

Notice that Poly1305 doesn't use the XOR operation but the modular addition. So the algorithms to recover the missing difference must be adapted to this case before being applied. However one can easily adapt the fast-XOR algorithm to this case. First count over the $2n/3$ least significant bits to avoid issues the carry, something the XOR operation doesn't have. Then when the lists of counters are up, we need to compute their cyclic convolution which is done with a fast convolution algorithm based on fast Fourier transform (instead of fast Walsh-Hadamard). Then we test for the lowest counter by running the known prefix algorithm looking for collisions on the least significant bits and sieving the modular subtraction of the most significant bits. This adaptation has similar complexities and proofs than the one described earlier and, in the case of Poly1305, one can further adapt the algorithms to take into account the fact that 22 bits of the key r are fixed at 0 effectively reducing the dimension of Φ.

7 Simulations' results

We have performed several simulated attacks to validate our algorithms.

7.1 Half-a-block secret recovery

We choose the blocks size $n = 64$ bits, secret S of size $n/2 = 32$ bits. We'll focus on recovering these 32 bits of secret encrypted using Tiny Encryption Algorithm (TEA) implemented in C in CTR mode to create two lists, the keystream output list $K_i \in \mathcal{K}$, and the encryptions $E_j = K_j \oplus (\overline{0}||S) \in \mathcal{E}$. Lists were of size 2^{35} and 2^{34} respectively so that we have $2^{35} \cdot 2^{34} = 2^{69} = n/2 \cdot 2^n$ which is sufficiently high so that everytime we could recover the secret. The two lists required around 400GB of disk space for storage.

The whole simulation takes less than 7 hours with a simple implementation on 4 cores. On the few trials that we ran we found that the generation and sorting of the lists took from 4 to 6 times the time needed to find enough prefix collisions and successfully recover S. This confirms that sieving over half a block has complexity comparable to birthday bound and is probably the most practical case for an attack scenario.

On the sieving itself, we tracked how many prefix collisions where uncovered before recovering totally S. We found that on average $\simeq 22.64 \times 2^{32}$ collisions were required ranging from 21×2^{32} to 26×2^{32} collisions. This result is consistent with the theoretical expectation predicted by the coupon collector problem : $2^{n/2} \cdot H_{2^{n/2}} \simeq 22.76 \times 2^{32}$ where H_n is the n^{th} harmonic number.

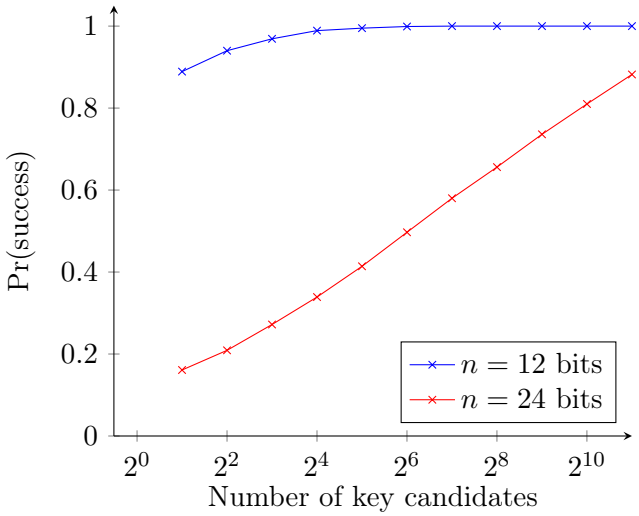
7.2 Indivisible one block secret

These simulations were made for block sizes $n = 12$, 24, 32 and 48 bits so we could do some statistical estimations of the success probability for this attack. We first create two lists of same size, one of raw keystream output and one XORed with an n-bit secret S. Then we pass the two lists in algorithm 4 counting over $n' = 2n/3$ (otherwise specified) to get a list of counters for each possible XOR outputs on those n' bits. Then the expected behavior of the attack would be to look for a solution whose n' first bits correspond to the position of the lowest counter and test this hypothesis with algorithm 3. If it returns a unique value then this is S and we are done, if it returns an empty set then test with the position of the second lowest counter, etc. We can therefore know the number of key candidates that would be required to recover S and, over many trials, have an estimation of the probability of success after a given number of candidates in these parameters.

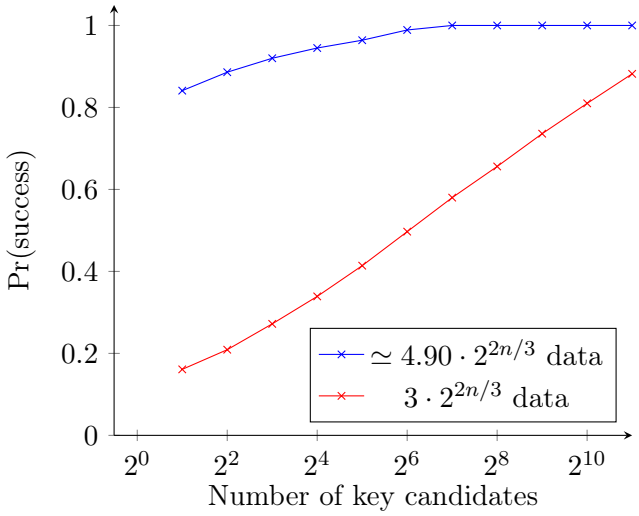
For block sizes of 12 and 24, we simulated a permutation simply by shuffling a range into a list. For bigger sizes of 32 and 48, we used the Simon's lightweight cipher from the NSA [BSS+15] as that is one of the rare block cipher who can act on 48-bit blocks.

FIGURE 3

(a) Results for lists size of $3 \cdot 2^{2n/3}$



(b) Results for $n = 24$ bits

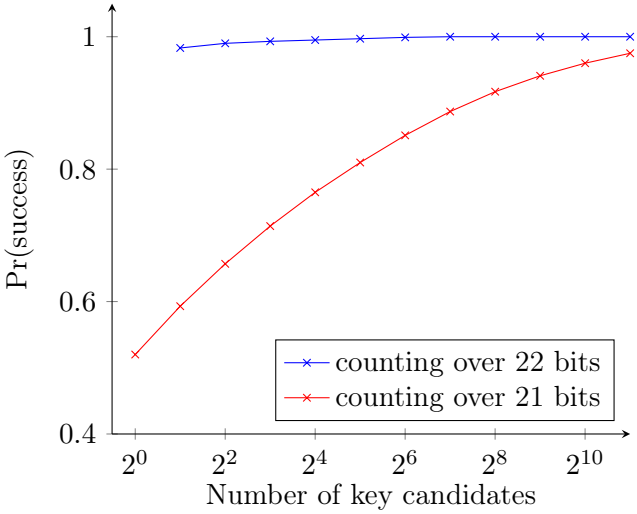


In general we observe in Figure 4b that the algorithm has a good chance of success with the first few candidates when using the generic parameters. Moreover the sensibility with respect to the data complexity (Figure 3b) and to the number of bits counted over (Figure 4a) is fairly high. These results back up our complexity analysis and are a good indication that no big constant is ignored by the big-$\mathcal{O}$ notation.

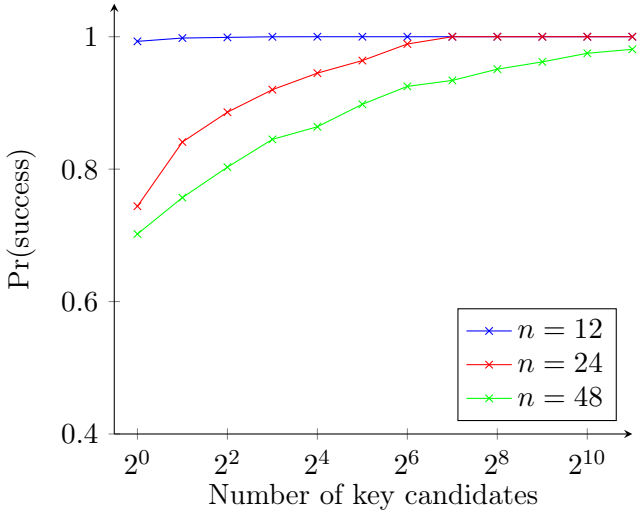
On the speed at which the probability increases we realized that, despite the log scale on the x axis, the curves take a straight (Figure 3a) or concave shape (Figure 4a 4b). That means that the probability of success with the next key candidate decreases very quickly with the number of key candidates already tested and proved wrong. For example for $n = 48$ bits (Figure 4b) over 756 trials the right key candidate was in the 2048 lowest counters in 98.1% of the time but the worst case found was 1 313 576 and these "very bad" cases push the mean rank of the right key candidate to 2287 and its sample variance to 2 336 937 008.

FIGURE 4

(a) Results for $n = 32$ bits ; $\sqrt{n}2^{2n/3} \simeq 5.66 \cdot 2^{2n/3}$ data



(b) Results for $\sqrt{n}2^{2n/3}$ data ; counting over $2n/3$ bits



8 Analysis

8.1 Impossible plaintext attack on indivisible n-bit block secret with large memory

Consider, without loss of generality and for blocks of size n, that we possess $a(n) \cdot 2^{2n/3}$ blocks of cipher keys and the same number of blocks of encrypted secret for $a(n)$ an unknown function of n that we will refer to as simply a. So in this setting we have $a^2 \cdot 2^{4n/3}$ different XORed-values possible between the two list that we will consider as indepedent and uniformly distributed over $2^n - 1$ values. We will then focus on $2n/3$ bits and ignore the rest. We count the number of occurrences for each possible combination on those $2n/3$ bit positions and store them in two lists of size $2^{2n/3}$. Using the fast Walsh-Hadamard transform 3 times, algorithm 4, we can therefore compute the same counters but for all the XORed-values. We hope that the counter for values that collides with the secret on those positions, the good counter, will be lower than all of the other counters, the bad counters, with probability $\Omega(1)$. In which case we say the algorithm succeeds.

Let X_i represents the fact that the i^{th} value belongs to a counter so that X_i follows a Bernoulli distribution and any counter $X = \sum_{i=1}^{a^2 2^{4n/3}} X_i$. Now we have to discriminate between the distributions of the good and bad counters :

Good case :	$Pr(X_i = 1) = (2^{n/3} - 1)/2^n = 2^{-2n/3} - 2^{-n}$	$\mathbf{E}[X_{good}] = 2^{2n/3}a^2 - 2^{n/3}a^2$
Bad case :	$Pr(X_i = 1) = (2^{n/3})/2^n = 2^{-2n/3}$	$\mathbf{E}[X_{bad}] = 2^{2n/3}a^2$

Now we are interested by the probability that a bad counter gets a value below $\mathbf{E}[X_{good}]$ as a measure of how distinct the distributions are. Using Chernov Bound we get :

$$Pr(X_{bad} < \mathbf{E}[X_{good}]) = Pr(X_{bad} < (1 - 2^{-n/3})2^{2n/3}a^2) = Pr(X_{bad} < (1 - 2^{-n/3})\mathbf{E}[X_{bad}])$$
$$\leq e^{-((2^{-n/3})^2 \cdot 2^{2n/3}a^2)/2)} = e^{-a^2/2}$$

And to compute the probability that no bad counter gets below $\mathbf{E}[X_{good}]$ we will have to assume their independence, which is wrong, but we will come back later to discuss this assumption.

$$\begin{aligned} Pr(X_{bad} \geq \mathbf{E}[X_{good}] | \forall X_{bad}) &= \prod_{X_{bad}} (1 - Pr(X_{bad} < \mathbf{E}[X_{good}])) \\ &\geq \left(1 - e^{-a^2/2}\right)^{2^{2n/3}} \end{aligned}$$

To conclude, we need to find an $a = a(n)$ such that this probability remains greater than some positive value as n grows. This is clearly achieved with $a = \mathcal{O}(\sqrt{n})$ as for example taking $a = \frac{2\sqrt{n}}{\sqrt{3 \cdot log2(e)}} \simeq 0.96\sqrt{n}$ we get :

$$\begin{aligned} Pr(X_{bad} \geq \mathbf{E}[X_{good}] | \forall X_{bad}) &\geq (1 - e^{-a^2/2})^{2^{2n/3}} \\ &\geq (1 - 2^{-2n/3})^{2^{2n/3}} \\ &\geq 0.25, \forall n \geq 3/2 \end{aligned}$$

Therefore we can bound the probability of success by the events '$X_{good} < \mathbf{E}[X_{good}]$', probability $1/2$, and '$X_{bad} \geq \mathbf{E}[X_{good}] | \forall X_{bad}$', probability at least $1/4$. Then we indeed have a probability of at least $1/8$ of having a successful algorithm. We can conclude that with $\mathcal{O}(n \cdot 2^{4n/3})$ XORed-values the algorithm has probability $\Omega(1)$ of succeeding.

Notice that this requires lists of size $\mathcal{O}(\sqrt{n} \cdot 2^{2n/3})$ but for the proof we only need the total number of pairs between the two lists. So we can break the requirement that the two lists are of comparable sizes as long as the product of their sizes sum up to the order of required values.

On the independence of the counters, this is obviously wrong as they are bound by the relation $X_{good} + \sum X_{bad} = a^2 2^{4n/3}$. However this relation becomes looser and looser as n grows so the approximation obtained should still be correct asymptotically. Moreover, the covariances implied is negative *i.e.* knowing one draw is big makes the other draws smaller in expectation to compensate. Small negative covariances will make the distribution look more evenly distributed in the sense that we can't observe too many extreme events in a particular direction which is good for the success rate of the algorithm. So the assumption of independence may be a conservative one for this complexity analysis.

8.2 Bit by bit n-bit block secret recovery

We place ourselves in a simple setting where one query returns a block of keystream and the encryption of $0||s_i$ with unknown bit s_i. We are interested in the query complexity for recovering n bits of secret one by one ; that is we need to know the first bit to ask for the second one, etc. The intuition is that we will need less and less queries to uncover the next bit as we go forward. Let :

$$\begin{aligned} U_i &\leftarrow \text{The expected number of encryption of } 0||s_i \text{ to recover } s_i \\ K_i &\leftarrow \text{The expected number of raw keystream outputs to recover } s_i \end{aligned}$$

From the definition of a query, the above description and because each time we find a bit of secret we can deduce a range of raw keystream outputs for the next step we have the relations :

$$
\begin{array}{ll}
(1) & K_1 = U_1 \\
(2) & K_{i+1} = K_i + U_i + U_{i+1} \text{ for } i \geq 1 \\
(3) & K_i \cdot U_i = 2^n \text{ (in expectation)}
\end{array}
$$

Let the following proposition $P_i : U_i = 2^{n/2}(\sqrt{i} - \sqrt{i-1})$
Which implies from (2) : $K_i = 2\sum_{k=1}^{i-1} U_k + U_i = 2^{n/2}(\sqrt{i} + \sqrt{i-1})$

(1) and (3) implies $K_1 = U_1 = 2^{n/2}$ so P_1 is true. Now suppose P_k true for all $k \leq i$, let's prove it holds for P_{i+1} :

$$
\begin{array}{rcl}
(3) & \Rightarrow & K_{i+1} \cdot U_{i+1} = 2^n \\
(2) & \Rightarrow & U_{i+1}^2 + (K_i + U_i) \cdot U_{i+1} - 2^n = 0 \\
P_i & \Rightarrow & U_{i+1}^2 + 2^{n/2} \cdot 2\sqrt{i} \cdot U_{i+1} - 2^n = 0 \\
\text{Positive root} & \Rightarrow & U_{i+1} = 2^{n/2}(\sqrt{i+1} - \sqrt{i}) \\
 & \Rightarrow & P_{i+1} \text{ is true.}
\end{array}
$$

Now that we have a closed form for each U_i we can recover the total query complexity to recover n bits of secret by summing over the U_i :
Average Query Complexity $= \sum_{i=1}^{n} U_i = 2^{n/2}\sqrt{n}$

9 Use of CTR mode in Communication Protocols

The CTR mode is widely used in internet protocols, in particular as part of the GCM authenticated encryption mode [MV04], with the AES block cipher. For instance, Mozilla telemetry data show that 85% of HTTPS connections from Firefox 53 use AES-GCM [3]. While attacks against modes with a 128-bit block cipher are not practical yet, it is important to limit the amount of data processed with a given key, in order to keep the probability of a successfull attack negligible, following the guidelines of Luykx and Paterson [LP16].

Surprisingly, there are also real protocols that use 64-bit block ciphers with the CTR mode (or variants of the CTR mode), as shown below. Attacks against those protocols would be (close to) practical, assuming a scenario where an attacker can generate the encryption of a large number of messages with some fixed secret.

SSH. Ciphersuites based on the CTR mode were added to SSHv2 in 2006 [BKN06]. In particular, 3DES-CTR is one of the recommended ciphers, but actual usage of 3DES-CTR seem to be rather low [ADHP16]. In practice, 3DES-CTR is optionally supported by the dropbear server, but it is not implemented in OpenSSH. According to a scan of the full IPv4 space by Censys.io [4], around 5% of SSH servers support 3DES-CTR, but actual usage is hard to estimate because it depends on client configuration. The SSH specification requires to rekey after 1GB of data, but an attack is still possible, although the complexity increases.

3. `https://mzl.la/2uGj0XV`
4. `https://censys.io/data/22-ssh-banner-full_ipv4`

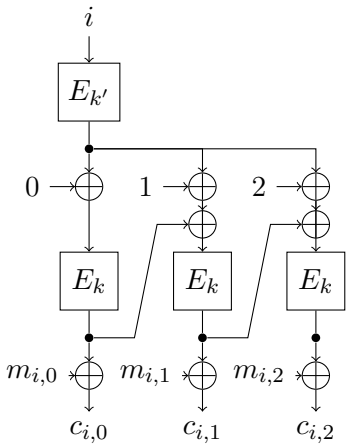


FIGURE 5 – f8 mode (i is a message counter)

3G telephony. The main encryption algorithm in UMTS telephony is based on the 64-bit blockcipher Kasumi. The mode of operation, denoted as f8, is represented in Figure 5. While this mode in not the CTR mode and was designed to avoid its weaknesses, our attack can be applied to the first block of ciphertext. Indeed the first block of message i is encrypted as $c_{i,0} = m_{i,0} \oplus E_k(E_{k'}(i))$, where the value $E_k(E_{k'}(i))$ is unique for all the messages encrypted with a given key.

There is a maximum of 2^{32} messages encrypted with a given key in 3G, but this only has a small effect on the complexity of attacks.

Because of the low usage of 3DES-CTR in SSH, and the difficulty of mounting an attack against 3G telephony in practice, we did not attempt to demonstrate the attack in practice, but the setting and complexity of our attacks are comparable to recent results on the CBC mode with 64-bit ciphers [BL16].

Conclusion

In this work, we have studied the missing difference problem and its relation to the security of the CTR mode. We have given efficient algorithms for the missing difference problem in two practically relevant cases : with an arbitrary missing difference, and when the missing difference is known to be in some low-dimension vector space. These algorithms leads to a message-recovery attack against the CTR mode with complexity $\tilde{\mathcal{O}}(2^{n/2})$, and a universal forgery attack against some Carter-Wegman MACs with complexity $\tilde{\mathcal{O}}(2^{2n/3})$.

In particular, we show that birthday attacks against the CTR mode can be mounted with roughly the same requirements and the same complexity as attack against the CBC mode. While both modes have similar security proofs, there was a folklore assumption that the security loss of the CTR mode with large amounts of data is slower than in the CBC mode because the absence of collision in the CTR keystream is harder to exploit than CBC collisions [FSK11, Section 4.8.2]. Our results show that this is baseless, and use of the CTR mode with 64-bit block ciphers should be considered unsafe (unless strict data limits are in place).

Références

[ABP+13] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. On the Security of RC4 in TLS. In Samuel T. King, editor, *USENIX Security 2013*, pages 305–320. USENIX Association, 2013.

[ADHP16] Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 1480–1491. ACM Press, October 2016.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997.

[Ber05] Daniel J. Bernstein. The poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 32–49. Springer, Heidelberg, February 2005.

[BKN06] M. Bellare, T. Kohno, and C. Namprempre. The Secure Shell (SSH) Transport Layer Encryption Modes. IETF RFC 4344, 2006.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers : Collision attacks on HTTP over TLS and OpenVPN. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 456–467. ACM Press, October 2016.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1) :3–72, 1991.

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK : Block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/585, 2015. `http://eprint.iacr.org/2015/585`.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 149–165. Springer, Heidelberg, January 1997.

[DL14] Itai Dinur and Gaëtan Leurent. Improved generic attacks against hash-based MACs and HAIFA. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 149–168. Springer, Heidelberg, August 2014.

[DR11] Thai Duong and Juliano Rizzo. Here come the ⊕ ninjas. 2011.

[Fer05] Niels Ferguson. Authentication weaknesses in gcm. Comment to NIST, 2005. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/CWC-GCM/Ferguson2.pdf`.

[FLS15] Thomas Fuhr, Gaëtan Leurent, and Valentin Suder. Collision attacks against CAESAR candidates - forgery and key-recovery against AEZ and Marble. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 510–532. Springer, Heidelberg, November / December 2015.

[FSK11] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography engineering : design principles and practical applications*. John Wiley & Sons, 2011.

[GPSW14] Jian Guo, Thomas Peyrin, Yu Sasaki, and Lei Wang. Updates on generic attacks against HMAC and NMAC. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 131–148. Springer, Heidelberg, August 2014.

[LKS+06] Changhoon Lee, Jongsung Kim, Jaechul Sung, Seokhie Hong, and Sangjin Lee. Forgery and key recovery attacks on PMAC and mitchell's TMAC variant. In Lynn Margaret Batten and Reihaneh Safavi-Naini, editors, *ACISP 06*, volume 4058 of *LNCS*, pages 421–431. Springer, Heidelberg, July 2006.

[LP16] Atul Luykx and Kenneth G. Paterson. Limits on authenticated encryption use in TLS, march 2016. `http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf`.

[LPW13] Gaëtan Leurent, Thomas Peyrin, and Lei Wang. New generic attacks against hash-based MACs. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2013.

[Mat94] Mitsuru Matsui. Linear cryptoanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397. Springer, Heidelberg, May 1994.

[McG12] David McGrew. Impossible plaintext cryptanalysis and probable-plaintext collision attacks of 64-bit block cipher modes. Cryptology ePrint Archive, Report 2012/623, 2012. `http://eprint.iacr.org/2012/623`.

[MV04] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, Heidelberg, December 2004.

[Pv95] Bart Preneel and Paul C. van Oorschot. MDx-MAC and building fast MACs from hash functions. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 1–14. Springer, Heidelberg, August 1995.

[Pv96] Bart Preneel and Paul C. van Oorschot. On the security of two MAC algorithms. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 19–32. Springer, Heidelberg, May 1996.

[PW14] Thomas Peyrin and Lei Wang. Generic universal forgery attack on iterative hash-based MACs. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 147–164. Springer, Heidelberg, May 2014.

[Rog11] Phillip Rogaway. Evaluation of some blockcipher modes of operation, 2011.