CLRW1^3 is not Secure Beyond the Birthday Bound

Breaking TNT with O($2^{n/2}$) queries

Mustafa Khairallah

Seagate Research Group, Singapore, Singapore
mustafa.khairallah@seagate.com

Abstract. In this paper, we present a new distinguisher for the Tweak-aNd-Tweak (TNT) tweakable block cipher with $O(2^{n/2})$ complexity. The distinguisher is an adaptive chosen ciphertext distinguisher, unlike previous attacks that are only non-adaptive chosen plaintext attacks. However, the attack contradicts the security claims made by the designers. Given TNT can be seen as the three-round CLRW1 tweakable block cipher, our attack matches its more conservative bound. We provide the distinguisher description, a probabilistic analysis of its behaviour, experimental verification and an analysis of why the proof fails to capture the security of TNT. In summary, the distinguisher is based on collision counting and exploits non-uniformity in the statistical behaviour of random permutations. It reduces the goal of finding the collision to solving a difference equation defined over a random permutation. Due to this relation, the number of collisions observed by the distinguisher is twice as expected from an ideal tweakable block cipher.

Keywords: Tweakable Block Cipher · TBC · Random Permutation · Provable Security · TNT · Tweak-aNd-Tweak · CLRW1

1 Introduction

Tweakable Block Ciphers (TBCs) have become important symmetric key primitives. They were introduced by Liskov *et al.* in their seminal paper "Tweakable Block Ciphers" [15, 16]. They have gained popularity due to their simplicity and how they can be used to build modes with Beyond Birthday Bound (BBB) security and simple security proofs. Several TBC designs have been proposed over the years. The design of a TBC falls into one of two categories: adhoc designs, and provable designs. Adhoc designs are designs built from scratch using adhoc techniques and their security depends mainly on cryptanalysis. Examples of this category are Deoxys-TBC [11], Skinny [3] and Qarma [1]. Provable designs are designs where The security of the TBC reduces to the security of an underlying primitive, such as a block cipher, a permutation or a pseudo-random function. Examples of this approach is XEX [21], LRW1 and LRW2 [15]. In this work, we focus on the second category. Particularly, we study the cipher known as

Tweak-aNd-Tweak (TNT) [2]. TNT was proposed in Eurocrypt 2020 and the designers of TNT claim that given three random permutations, the TNT TBC is secure against all adversaries that make up to $2^{2n/3}$ adaptive chosen plaintext or ciphertext queries.

Contribution In this paper, we study the security claim of TNT. We furnish a distinguisher between TNT and a family of Tweakable Uniformly Random Permutations (TURPs) using $O(2^{n/2})$ chosen plaintext queries and $O(2^{n/2})$ adaptive chosen ciphertext queries. We study the distinguisher analytically using the statistics of random permutations and analysis of the behaviour of difference equations and Difference Distribution Tables (DDTs) of random permutations. We also implement and verify the attack experimentally on small instances of TNT. Since the attack clearly contradicts the security claims of the designers of TNT, we study their security proof and identify a bug, where a random variable is erroneously assumed to have a uniform distribution, leading to an over estimation of the security.

Impact As mentioned, the authors of [2] claimed the sTPRP security of TNT to be $2n/3$ bits. In Asiacrypt 2020, the authors of [8] conjectured that the sTPRP security of TNT is probably $3n/4$ bits. In [22], the authors have stated: *"A natural open problem is the exact security of CLRW1^{r,E}. Unlike CLRW2^{r,E}, exact security of CLRW1^{r,E} for $r = 3$ already appears challenging, and might require new proof approaches*". We believe this work answers a critical research question of both practical and theoretical implications. On one hand, it studies the exact security of an efficient construction that has several practical applications. On the other hand, it offers another cautionary tale on how to use statistical proof techniques such as the χ^2 method.[1]

Additionally, the attack applies to practical instances of TNT: TNT-AES in [2] and TNT-SM4-128 in [9]. The authors of [9] also introduced TNT-SM4-32, where the tweak size is limited to 32-bits. Our distinguisher requires $O(2^{n/2})$ tweaks, where $n = 128$ in case of TNT-SM4. Hence, the distinguisher directly applies to TNT-SM4-128, which has a tweak size of 128 bits. It does not directly apply to TNT-SM4-32, since the tweak space is too small. However, since our distinguisher breaks the the BBB security proof in [2], the exact security of TNT-SM4-32 and whether it is has BBB security is an open question.

Outline In Section 2, we give necessary definitions needed to follow the paper. In Section 3, we motivate the paper by giving an overview of a variety of issues related to LRW1, TNT and CLRW1 [22]. Sections 4 and 5 describe our main result; in Section 4 we describe our birthday bound adaptive CCA distinguisher, while in Section 5 we study the statistical behaviour of the distinguisher showing why it succeeds with birthday bound complexity. In Section 6 we provide experimental verification of the distinguisher using small ciphers. In Section 7, we describe the bug in the security proof in [2] that leads to the proven bound being contradicted by our attack. Finally, the paper is concluded in Section 8.

[1] Refer to [5] for another example of erroneously estimated distributions.

2 Preliminaries

Block Ciphers A Block Cipher (BC): $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$ is a family of permutations that are indexed by a secret key $K \in \mathcal{K}$. In other words, for each key K selected uniformly randomly from $\mathcal{K}$, $E_K(\cdot)$ is a permutation over $\{0,1\}^n$. A secure BC is indistinguishable from a uniformly random permutation (URP): $P : \{0,1\}^n \to \{0,1\}^n$, for a uniformly random key $K \in \mathcal{K}$. We call this Pseudo-Random Permutation (PRP) security. We refer to the inverse of a block cipher as its decryption function. We use E_K^{-1} and D_K interchangeably. If (E_K, D_K) are indistinguishable from (P, P^{-1}), then we refer to this as strong PRP (sPRP) security.

Tweakable Block Ciphers A Tweakable Block Cipher (TBC): $\tilde{E} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ is a family of permutations that are indexed by a secret key $K \in \mathcal{K}$ and a public tweak $T \in \mathcal{T}$. In other words, for each key K selected uniformly randomly from $\mathcal{K}$ and any choice of $T \in \mathcal{T}$, $\tilde{E}_K^T(\cdot)$ is a permutation over $\{0,1\}^n$. A secure TBC is indistinguishable from a family of uniformly random permutations. Such family is sometimes referred to as Tweakable URP (TURP). We call this Tweakable Pseudo-Random Permutation (TPRP) security. We refer to the inverse of a TBC as its decryption function. We use $\tilde{E}_K^{-1}$ and $\tilde{D}_K$ interchangeably. If $(\tilde{E}_K, \tilde{D}_K)$ are indistinguishable from $(\tilde{P}, \tilde{P}^{-1})$, where $\tilde{P}$ is a TURP, then we refer to this as strong TPRP (sTPRP) security. This is also commonly referred to as the CCA security of the TBC.

Poisson Distribution The Poisson distribution is a discrete distribution with parameter λ and its Probability Mass Function (PMF) is defined as:

$$\text{Poisson}(i;\lambda) = \Pr[X = i] = \frac{\lambda^i e^{-\lambda}}{i!}$$

where the mean and variance are both equal to λ.

Difference Distribution Tables Let $\pi : \{0,1\}^n \to \{0,1\}^n$ be a permutation. The equation

$$\pi(x \oplus \delta) \oplus \pi(x) = \Delta$$

is known as the difference equation (δ, Δ) over π, where $\delta, \Delta \in \{0,1\}^n$ and $\oplus$ is addition in the Galois Field GF(2^n). Since π is a permutation, then any difference equation must have an even number of solutions; either no solutions at all (0), or an even non-zero number of solutions. Note that if M is a solution for the difference equation (δ, Δ), then $M \oplus \delta$ must also be a solution. A Difference Distribution Table (DDT) is a $2^n \times 2^n$ table constructed by counting the number of solutions of each possible difference equation. It looks like Table 1. Each row or column adds up to 2^n and all the entries are even. The entry $(0,0)$ is always 2^n and the rest of the entries of the first row and first column are all zero. If all the entries are either 0 or 2^n, then the permutation is linear. If all the entries are either 0 or 2, except one, then the permutation is known as an Almost Perfect Non-linear (APN) permutation. A random permutation is likely to fall somewhere in between.

δ \ Δ	0	1	2	...	$2^n - 1$
0	2^n	0	2	...	0
1	0	0	0	...	4
2	0	2	0	...	0
...	...	...	...	...	...
$2^n - 1$	0	0	8	...	2

Table 1. An example of a DDT.

3 Motivation

In their seminal paper on Tweakable Block Ciphers [15], Liskov, Rivest and Wagner presented two constructions for building TBCs using BCs. The first method requires two calls to a BC and is given by

$$C \leftarrow E_K(T \oplus E_K(M)).$$

The second construction requires requires a universal hash function and a BC and is given by

$$C \leftarrow h(T) \oplus E_K(h(T) \oplus M).$$

These two constructions have inspired a lot of work on TBCs. They came to be known as LRW1 and LRW2, in reference to the authors. We note that in an extended version of their paper [16]. The authors have renamed the constructions as CBC-MAC-Tweaked (CMT) and LRW, respectively. However, the names LRW1 and LRW2 are more common in the literature.

In 2012, Landecker *et al.* [13] proposed a generalization of LRW2, dubbed CLRW2^r, where r is a parameter of the construction. The construction simply cascades r instances of LRW2 with r independent keys and r independent hash functions. However, generalizations of LRW1 have received less attention until 2020, potentially because LRW1 was proven CPA-secure only up to birthday bound, and this bound is tight.

Tightness of LRW1 The tightness of the birthday bound of LRW1 is obvious. Consider an adversary that fixes a message M and chooses q tweaks $T_1, \ldots T_q$. When M is fixed, then a TURP behaves as a random function with domain $\mathcal{T}$, while LRW1 behaves as a permutation over $\mathcal{T}$. When $q > 2^{n/2}$, the adversary can observe whether there are collision in the output or not. Besides, the lack of a security proof for CCA security is also not an open question. Consider an adversary that works as follows:

1. Encrypt $\tilde{E}_K^0(A) \rightarrow C$.
2. Decrypt $\tilde{D}_K^T(C) \rightarrow \bar{A}$.
3. Encrypt $\tilde{E}_K^X(\bar{A}) \rightarrow \bar{C}$.

4. Decrypt $\tilde{D}_K^{X \oplus T}(\bar{C}) \to Z$.

5. Output 1 if $Z = A$. Output 0 otherwise.

Since all the queries use different tweaks, the responses of a TURP are all uniformly distributed. Hence, the adversary outputs 1 with probability 2^{-n}. In the real world, we consider the four queries as one query:

$$Z = D_K(T \oplus X \oplus D_K(E_K(X \oplus E_K(D_K(T \oplus D_K(E_K(E_K(A)))))))),$$

where there are three instances of forward and backward queries cancelling each other, leading to

$$Z = D_K(T \oplus X \oplus X \oplus T \oplus E_K(A)) = D_K(E_K(A)) = A.$$

This observation that we can eliminate the effect of some BC calls by alternating forward and backward TBC queries will also apply to generalizations of LRW1.

Tweak-aNd-Tweak (TNT) Bao *et al.* [2] proposed TNT as a three-round generalization of LRW1. Given three independent random permutations, TNT is given by

$$C \leftarrow \pi_3(T \oplus \pi_2(T \oplus \pi_1(M))).$$

They give a security proof in the information theoretic setting (the three permutations are uniformly random and the adversary is unbounded) that shows the construction is CCA secure up to $2^{2n/3}$ queries, *i.e.,* beyond the birthday bound. The authors rightly point out that there is a missing link between this security bound and real constructions built using independent ciphers. In [8], Guo *et al.* presented distinguishing attacks on TNT with complexity $\sqrt{n}2^{3n/4}$, alongside a CPA security proof up to $2^{3n/4}$. However, the attacks presented by the authors are non-adaptive chosen plaintext attacks. Similar to LRW1, it is possible that TNT has different CCA and CPA bounds, and these attacks do not address the tightness of the CCA bound.

CLRW1^{r,E} Zhang *et al.* [22] recently presented a generalization of LRW1 and TNT, dubbed as Cascaded LRW1 (CLRW1), and parameterized by the number of rounds and keys r. It is given by

$$C \leftarrow E_{K_r}(T \oplus \ldots E_{K_2}(T \oplus E_{K_1}(M))),$$

and its idealized variant is

$$C \leftarrow \pi_r(T \oplus \ldots \pi_2(T \oplus \pi_1(M))).$$

The idealized variant coincides with TNT when $r = 3$. Besides, when $r = 2$, it is similar to LRW1 except with two independent keys. However, its security is the same as LRW1 and the described attacks on LRW1 also apply to CLRW1^2 in both the CPA and CCA settings. Interestingly, the authors provide a security

proof in the CPA setting, and rely on the compositional theorem in [12] to give the a CCA bound when r is odd, given CPA security of two instances with $(r+1)/2$ rounds. This means that their bound for the CCA security of TNT (CLRW1^3) is almost the same as the CPA bound for CRLW1^2, which is the birthday bound ($2^{n/2}$), while $2^{2n/3}$ is only reached with $r = 5$. This is significantly worse than the bound given by the TNT designers.

Observation on CLRW1^r Assume two adversaries that are interested in key recovery of CLRW1^r. Both adversaries are only allowed to make a very small number of queries q, say one query to each available oracle. The first adversary, **P** is a CPA adversary, and can make 1 query. The second adversary **C** is a CCA adversary and can make 1 adaptive query to each of the forward and backward oracles. **P** receives

$$C \leftarrow E_{K_r}(T \oplus \ldots E_{K_2}(T \oplus E_{K_1}(M)))$$

and is required to guess r keys. **C**, on the other hand, can make two carefully selected queries

$$C \leftarrow E_{K_r}(T_0 \oplus \ldots E_{K_2}(T_0 \oplus E_{K_1}(M)))$$
$$X \leftarrow D_{K_1}(T_1 \oplus \ldots (T_1 \oplus D_{K_r}(C)))$$

and try to guess the keys corresponding to

$$X \leftarrow D_{K_1}(T_1 \oplus \ldots D_{K_{r-1}}(T_1 \oplus T_0 \oplus E_{K_{r-1}}(T_0 \oplus \ldots E_{K_2}(T_1 \oplus E_{K_1}(M)))))$$

which only requires guessing $r-1$ keys. This indicates that similar to LRW1/ CLRW1^2, CLRW^r is indeed weaker in the CCA setting that in the CPA setting. However, this observation in itself is not an attack on any of the bounds of TNT or CLRW1^r, since the TNT is only proven secure in the idealized model and the bound of CLRW1^r falls apart if any of the keys is guessed correctly. Nevertheless, this observation merits a deeper look into the CCA security of CLRW1^r. In this work, we start by studying the CCA security of CLRW1^3/TNT in the idealized model: the three cipher calls use three independent uniformly random permutations. For simplicity, we will refer to this variant as TNT, while all the results apply to CLRW1^3, as well.

4 Birthday-Bound Distinguisher for TNT

In this section, we describe our CCA distinguishing attack against TNT. The distinguisher **D** is parameterized by the complexity q and a threshold $\theta(q)$. It makes q forward queries and q backward queries. It is described in Algorithm 1.

The description of the distinguisher is quite simple: *Cascade the forward and inverse queries, with tweaks T and $T \oplus \Delta$ where Δ and the plaintext M are fixed for all queries, and $\Delta \neq 0$. Make sure that for all $0 < i < q$, $0 < j < q$ and $i \neq j$, $T_i \neq T_j$ and $T_i \neq T_j \oplus \Delta$. Count the number of collisions at the output of backward queries.* One iteration of the distinguisher is visually depicted in Figure 1, and Figure 2 depicts the effective behavior as the effect of π_3 is removed and we have an XOR with a constant Δ between the forward and backward queries. Figure 3 shows the internal values in the effective trace during one iteration.

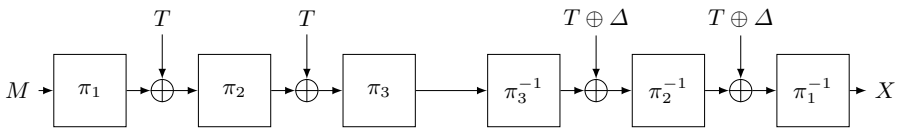

Fig. 1. One iteration of the distinguisher in Algorithm 1.

Algorithm 1 The distinguisher **D** against the CCA security of TNT.

$M \xleftarrow{\$} \{0,1\}^n$
$\Delta \leftarrow \mathtt{itob}_n(2^{n-1})$
$L \leftarrow [0 \ \forall \ 1 \leq i \leq 2^n]$
$\mathtt{coll} \leftarrow 0$
for $i \in \{0, 1, \cdots, q-1\}$ **do**
    $C \leftarrow \tilde{E}(\mathtt{itob}_n(i), M)$
    $X \leftarrow \tilde{E}^{-1}(\mathtt{itob}_n(i) \oplus \Delta, C)$
    $\mathtt{coll} \leftarrow \mathtt{coll} + L[\mathtt{btoi}(X)]$
    $L[\mathtt{btoi}(X)] \leftarrow L[\mathtt{btoi}(X)] + 1$
end for
if $\mathtt{coll} \geq \theta(q)$ **then**
    return 1
else
    return 0
end if

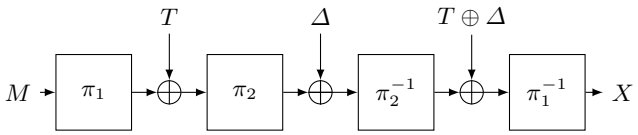

Fig. 2. The effective iteration of the distinguisher in Algorithm 1.

Analysis of the distinguisher In the ideal world, each query uses a unique tweak and a new uniform random permutation is sampled for each query. Hence, all the responses X are uniformly distributed. Given two queries, the probability of collision is $1/2^n$, and the behaviour follows the birthday collision search. The input space of the construction in Figure 2 is $\mathcal{T}$ and has size of 2^n possibilities. Thus, the expected number of collisions in the range of X can be estimated by

$$\frac{\binom{2^n}{2}}{2^n} = \frac{2^n - 1}{2}.$$

In the real world, we have a relation that is maintained across all queries:

$$V_o \oplus V_e = \Delta.$$

Furthermore, each query defines a difference equation over π_2:

$$\pi_2(U_o \oplus \delta) \oplus \pi_2(U_o) = \Delta,$$

where $\delta = U_o \oplus U_e$. By construction, this equation must have at least two

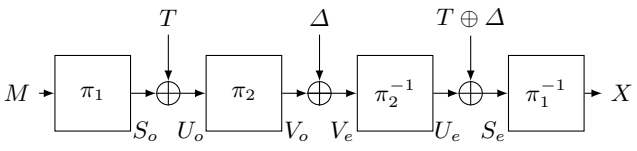

Fig. 3. The internal values of an iteration of the distinguisher in Algorithm 1.

solutions. The first is is U_o and the second is U_e. Hence, the query $(T^\star, M)$, where $T^\star = U_e \oplus S_o$ collides with (T, M). However, whether this is the only collision that leads to X, or not, depends on the difference distribution of the permutation π_2. For now, let the set of solutions to the difference equation

$$\pi_2(x \oplus \beta) \oplus \pi_2(x) = \Delta$$

be $\mathcal{S}_{\beta,\Delta}$. Consider an equation $\pi_2(x \oplus \delta) \oplus \pi_2(x) = \Delta$ that has four solutions:

$$\pi_2(U_o \oplus \delta) \oplus \pi_2(U_o) = \Delta$$

$$\pi_2((U_o \oplus \delta) \oplus \delta) \oplus \pi_2(U_o \oplus \delta) = \Delta$$

$$\pi_2(U_o \oplus \gamma \oplus \delta) \oplus \pi_2(U_o \oplus \gamma) = \Delta$$

$$\pi_2((U_o \oplus \gamma \oplus \delta) \oplus \delta) \oplus \pi_2(U_o \oplus \gamma \oplus \delta) = \Delta$$

and the four corresponding tweaks

$$S_o \oplus U_o$$

$$S_o \oplus U_o \oplus \delta$$

$$S_o \oplus U_o \oplus \gamma$$

$$S_o \oplus U_o \oplus \gamma \oplus \delta.$$

Then,

$$S_e^{(0)} = (U_o \oplus \delta) \oplus S_o \oplus U_o \oplus \Delta = S_o \oplus \delta \oplus \Delta$$

$$S_e^{(1)} = U_o \oplus (\delta \oplus S_o \oplus U_o) \oplus \Delta = S_o \oplus \delta \oplus \Delta$$
$$S_e^{(2)} = (U_o \oplus \gamma \oplus \delta) \oplus (S_o \oplus U_o \oplus \gamma) \oplus \Delta = S_o \oplus \delta \oplus \Delta$$
$$S_e^{(3)} = (U_o \oplus \gamma) \oplus (\delta \oplus S_o \oplus U_o \oplus \gamma) \oplus \Delta = S_o \oplus \delta \oplus \Delta.$$

Thus,

$$S_e^{(0)} = S_e^{(1)} = S_e^{(2)} = S_e^{(3)}$$

and they form a multi-collision. The value propagation of this example is visually depicted in Figure 4.

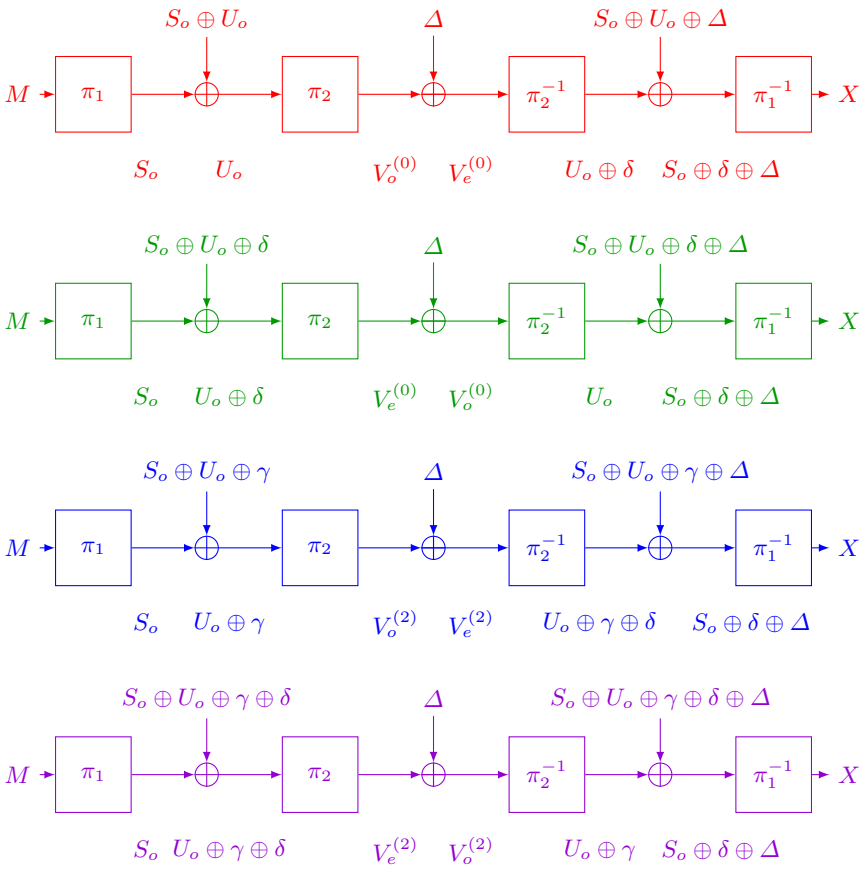

Fig. 4. The propagation in a four-way multi-collision.

If we know the exact values of $|\mathcal{S}_{\beta,\Delta}| \ \forall \beta \in \{0,1\}^n$, we can calculate the exact number of collisions in the range of X. However, since the permutation

is secret, such information is not available. The next best thing is to know for a given Δ, how many equations have 0 solutions, how many equations have 2 solutions,...etc. Let Q_i be the number of values β such that $\pi_2(x \oplus \beta) \oplus \pi_2(x) = \Delta$ has i solutions and m is the maximum number of possible solutions for any such equation. Then, the number of collisions is given by

$$\texttt{coll} = Q_2 + Q_4 * 6 + Q_6 * 15 + Q_8 * 28 + ... + Q_m \binom{m}{2}. \quad (1)$$

If π_2 is an APN, then $\texttt{coll} = 2^{n-1}$. This means that on the average, the APN case has half a collision more than the ideal case. This may not be enough to distinguish between the two cases. However, if π_2 deviates in the slightest from being an APN, *e.g.,* if one of the considered equations has 4 solutions, we get

$$\texttt{coll} = 2^{n-1} - 2 + 6 = 2^{n-1} + 4,$$

which is 4.5 more than the ideal case. As we consider that more equations have more than two solutions, the number of expected collisions increases. The worst case scenario is when π_2 is an affine permutation, in which case, the expected number of collisions is $\binom{2^n}{2}$. However, this case is not relevant for attacks on designs based on block ciphers. We are interested in the expected number of collisions when π_2 is a random permutation. In the next section, we show that the expected number of collisions is 2^n, twice that of the ideal world.

5 On the Statistics of Random Permutations

A random permutation over n bits is sampled uniformly from the set of all possible permutations over n bits. We recall that the DDT of a permutation π is a $2^n \times 2^n$ table that for an input difference β and output difference Δ includes the number of solutions of the difference equation:

$$\pi(X \oplus \beta) \oplus \pi(X) = \Delta.$$

O'Connor showed in Eurocrypt 93 [19] that the expected percentage of zeros in such table for a random permutation is 60.65%. If π is an APN, then the percentage of zeros will be slightly higher than 50%. This already shows that the distinguishing advantage is non-negligible, as the relatively high percentage of zeros will be offset by many entries that are larger than 2, since each row and column in the DDT must add up to 2^n. In fact, Daemen and Rijmen [6] showed that the distribution of the entries of the DDT is given by Poisson's distribution. Particularly,

$$\Pr[|S_{\beta,\Delta}| = x] = \frac{0.5^{x/2} e^{-0.5}}{(x/2)!}.$$

Using Bayes' theorem, then for $x > 0$,

$$\Pr[|S_{\beta,\Delta}| = x || S_{\beta,\Delta}| > 0] =$$

$$\frac{\Pr[|S_{\beta,\Delta}| = x] \Pr[|S_{\beta,\Delta}| > 0 || S_{\beta,\Delta}| = x]}{\Pr[|S_{\beta,\Delta}| > 0]} =$$

$$\frac{\Pr[|S_{\beta,\Delta}| = x]}{\Pr[|S_{\beta,\Delta}| > 0]} =$$

$$\frac{0.5^{x/2} e^{-0.5}}{(x/2)!(1 - e^{-0.5})}$$

These distributions can be used to estimate Equation 1. Let $x = 2i$, then[2]

$$E[\texttt{coll}] = e^{-0.5} \cdot 2^n \sum_{i>0} \frac{0.5^i \binom{2i}{2}}{i!} =$$

$$e^{-0.5} \cdot 2^n \sum_{i>0} \frac{0.5^i \frac{2i(2i-1)}{2}}{i!} =$$

$$e^{-0.5} \cdot 2^{n+1} \sum_{i>0} \frac{0.5^i i(i - 0.5)}{i!} =$$

$$e^{-0.5} \cdot 2^{n+1} \sum_{i>0} \frac{0.5^i i(i - 1 + 0.5)}{i!} =$$

$$e^{-0.5} \cdot 2^{n+1} \sum_{i>0} \left(\frac{0.5^i i(i-1)}{i!} + \frac{0.5^i i \times 0.5}{i!} \right) =$$

$$e^{-0.5} \cdot 2^{n+1} \left(\sum_{i>0} \frac{0.5^i i(i-1)}{i!} + \sum_{i>0} \frac{0.5^i i \times 0.5}{i!} \right) =$$

$$e^{-0.5} \cdot 2^{n+1} \left(\sum_{i>1} \frac{0.5^i}{(i-2)!} + 0.5 \sum_{i>0} \frac{0.5^i}{(i-1)!} \right) =$$

$$e^{-0.5} \cdot 2^{n+1} \left(0.5^2 \sum_{i>1} \frac{0.5^{i-2}}{(i-2)!} + 0.5^2 \sum_{i>0} \frac{0.5^{i-1}}{(i-1)!} \right) =$$

$$e^{-0.5} \cdot 2^{n+1} \left(0.5^2 e^{0.5} + 0.5^2 e^{0.5} \right) = 0.5 \cdot 2^{n+1}.$$

Therefore,

$$E[\texttt{coll}] = 2^n,$$

which means that the distinguisher in Algorithm 1 is expected to have twice as many collisions in the real world as in the ideal world. $\theta(q)$ can be generalized as:

$$\theta(q) = 2^{2d-1} + 2^{2d-2}$$

when $q = 2^{n/2+d}$, which is $\approx 1.5\times$ the expected number of collisions in the ideal case.

[2] $e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ and $e^x = \sum_{i>b} \frac{x^{i-b-1}}{(i-b-1)!}$.

To verify that the sampled permutations follow the same distribution, we have implemented a Monte-Carlo experiment to estimate the probability distribution of the number of solutions of a difference equation given that solutions exist by generating many random permutations for $16 \geq n \geq 30$. Almost all the generated permutations satisfied that the percentage of zero entries is around 60.65%. We found that the distribution settles around approximately the distribution in Table 2.

Table 2. The estimated probability distribution of the number of solutions for a difference equation over a random permutation, when it is known that solutions exist.

x	2	4	6	≥ 8
$\Pr(x)$	0.772	0.192	0.032	0.004

The distribution in Table 2 helps us estimate the number of expected solutions and the probability of a collision. Note that while stopped at 8 solutions, including mores solutions only increases the probability of collision. Since the probability of more than 8 solutions seems to be very small, we believe estimation to be a good enough approximation. Assuming the maximum number of solutions is 8, we can estimate Q_i as

$$E[Q_i] = 0.3935 \times \Pr[i] \times 2^n.$$

By substituting in Equation 1, we get

$$E[\texttt{coll}] = 0.3935 \times 2^n(0.772 + 0.192 \times 6 + 0.032 \times 15 + 0.004 \times 28) =$$

$$0.3935 \times 2.516 \times 2^n \approx 2^n.$$

This estimation indicates that when π_2 is a random permutation (or a well-designed block cipher), the expected number of collisions is twice that of the ideal world. Hence, by setting $q = c2^{n/2}$ for a small constant c, and setting the appropriate θ, in Algorithm 1, we get a distinguisher that succeeds with very high probability. In the next section, we verify these estimations with practical experiments.

6 Experimental Verification

In order to gain more confidence in the attack, we have implemented two experiments to verify the distinguishing advantage. In the first experiment, we used random permutations generated using Python NumPy's `shuffle` and `argsort` functions, to generate and invert a permutation, respectively. We generated permutations of sizes 16, 20, 24, 28 and 32 bits and performed the distinguishing attack on each generated permutation. This implementation is given in Appendix B. Results where taken over an average of $1,000 \sim 10,000$ random generations (each consisting of 3 independent permutations). In the ideal world,

random values are sampled, since the uniqueness of the tweak ensures each permutation is sampled at most once. Table 3 includes the average number of collisions for $n = 16$ and $n = 20$, which matches the number of collisions observed for other values of n, as well. The distinguisher reaches 16 expected collisions in the real world $4\times$ faster than the distinguisher in [8] for $n = 16$ and $16\times$ faster for $n = 20$.

Table 3. Average number of collisions using random permutations.

n	16					
$log_2(q)$	6	7	8	9	10	11
real	0.06	0.27	0.96	3.72	15.62	63.59
ideal	0.023	0.12	0.48	1.98	7.91	31.17
n	20					
$log_2(q)$	8	9	10	11	12	13
real	0.073	0.203	1.02	4.01	15.69	63.63
ideal	0.023	0.11	0.47	1.94	7.92	32.57

Table 4 shows the success rate for the different values of n and different parameters q and $\theta(q)$. The distinguisher reaches $\geq 85\%$ with complexity $2^{n/2+3}$ and and 99% success rate with complexity $2^{n/2+4}$, since each iteration includes two queries to the construction. For large n, the factors 2^3 and 2^4 are small. For a visual representation, Figure 5 shows the comparison between the complexity of the distinguisher against the birthday bound and the claim in [2]. The distinguisher breaks the claim with $\geq 85\%$ success rate for $18 < n \leq 24$, and breaks it with $\geq 99\%$ for $n > 24$. With complexity $2^{n/2+5}$, we get a success rate of almost 100%, and an attack that breaks the security claim for In practice, $n \geq 64$.

Table 4. The success rate achieved for different values of n and q.

n	q (85%)	$\theta(q)$ (85%)	Success Rate	q (99%)	$\theta(q)$ (99%)	Success Rate
16	10	12	87.2%	11	48	99%
20	12	12	86.6%	13	48	99%
24	14	12	90%	15	48	99%
28	16	12	85%	17	48	99%
32	18	12	87.5%	19	48	99%

In order to validate our experiments further, and eliminate any issues that may arise from Python's random generation, we ran a second experiment using the implementation of the 16-bit cipher Small-Present-16 [14] provided by the authors of [8]. This implementation is provided in Appendix C. The number of collisions is taken as an average over $10,000$ executions of the attack. The results are presented in Table 5. The results statistically match the random permutation case. A sample of the distribution of the number of solutions for

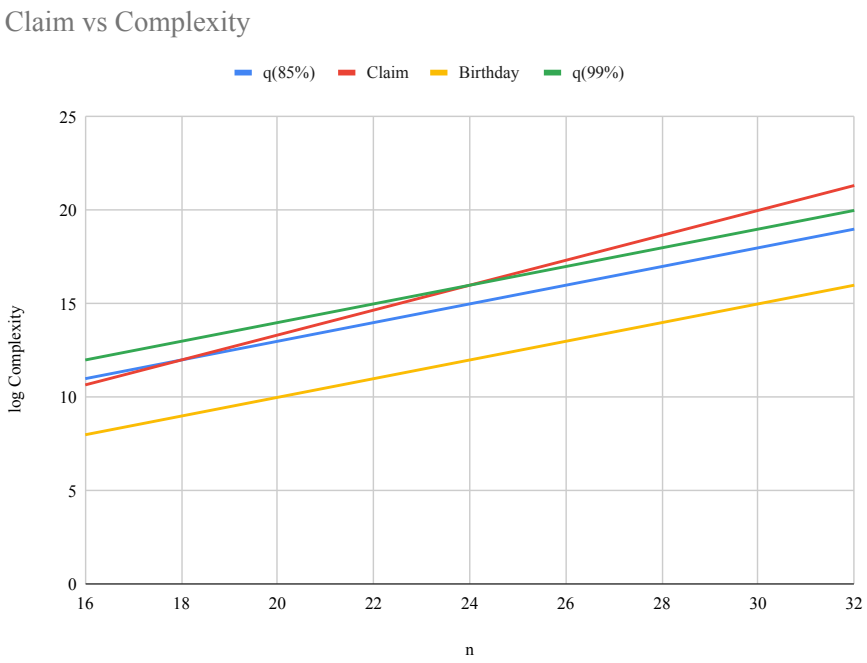

Fig. 5. The complexity of the distinguisher for different success rates compared to the claim of [2] and the birthday bound.

a input difference against all possible output differences and for a given key is given in Table 6. The distribution follows closely the simulated distribution in Table 2, which both validates our simulations and indicates that Small-Present-16 behaves closely to a randomly selected permutation. We have also replicated the success rate experiment and got 90.9% for $q = 2^{10}$ and 99.7% for $q = 2^{11}$.

Table 5. Average number of collisions using Small-Present-16.

n	16					
$log_2(q)$	6	7	8	9	10	11
real	0.058	0.25	0.98	4.02	16	63.94
ideal	0.027	0.12	0.49	1.98	7.98	31.92

Table 6. A sample of the distribution of the number of solutions for a difference equation defined over Small-Present-16 for a given secret key.

x	2	4	6	≥ 8
$\Pr(x)$	0.773	0.191	0.031	0.005

7 On the Security Proof of TNT

The authors of [2] presented a CCA security proof of TNT that clearly contradicts our attack. Assuming our attack is correct, given it is supported by practical verification, theoretical analysis and practical estimations, the contradiction must stem from a bug in the proof. The proof follows the χ^2 method proposed by Dai *et al.* [7]. Compared to other proof methods, this method is quite recent. After carefully studying the security proof, we identified an issue that involves a fundamental, yet subtle, case analysis. The main technique of the proof, from a high level point of view, works as follows:

- A deterministic distinguisher observes the first $l-1$ queries and selects whether the next query is a forward or inverse query as well as the tweak T_l and the plaintext M_l or ciphertext C_l.
- Find the probability distribution of all the internal values of the construction given the first $l-1$ query. We call a set of possible vectors of internal values **Inter**.
- For each possible **Inter**, estimate the probability distribution of each possible response to query l.

The authors then analyze different possible cases and apply the χ^2 method on the resulting distribution.

In order to better understand the issue, we analyze our distinguisher in the flow of the security proof. The distinguisher in Algorithm 1 works as follows:

- If l is odd, it makes a forward query $(X_0, T_{l-2}+1)$.
- If l is even, it makes a backward query $(Y_{l-1}, T_{l-1} \oplus \Delta)$.

Let (S_o, U_o, V_o) are the output of π_1, input of π_2 and output of π_2 in the last (odd) query $l-1$, and we estimate the probability

$$\Pr[X_l = X_i | X_i \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}].$$

Let (S_i, U_i, V_i) and (S_e, U_e, V_e) are the corresponding internal values of X_i and X_l, respectively. Then, we know that

$$V_o \oplus V_e = \Delta$$

and

$$\Pr[X_l = X_i | X_i \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}] =$$

$$\Pr[S_e = S^{'} | X^{'} \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}] =$$

$$\Pr[U_e \oplus T_{l-1} \oplus \Delta = U_i \oplus T_{i-1} \oplus \Delta | X^{'} \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}] =$$

$$\Pr[U_e \oplus U_i = T_{l-1} \oplus T_{i-1} | X^{'} \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}] =$$

Since X_0 is fixed for all odd queries, so is S_o. Thus, $U_o \oplus T_{l-1} = U_{i-1} \oplus T_{i-1}$. Therefore,

$$\Pr[U_e \oplus U_i = U_o \oplus U_{i-1} | X^{'} \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}] =$$

$$\Pr[U_e \oplus U_o = U_i \oplus U_{i-1} | X^{'} \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}] \approx \frac{|\mathcal{S}_{\delta,\Delta}| - 1}{2^n}$$

where $\delta = U_o \oplus U_e$. As discussed in the analysis of the distinguisher, this probability depends on the DDT of π_2 and is not the same for every permutation. Thus, it deviates from the distribution assumed in [2]. In terms of the proof presented in [2], the event we are discussing belongs to case 5 (case 1 if we swap all the forward and backward queries). In this case, the authors claim

$$\Pr[X_l = X_i | X_i \in \mathcal{Q}_l \texttt{ and } i \text{ is odd}] \leq \frac{4l}{2^{2n}} + \frac{1}{2^n - l}$$

(Equation (9) of [2]). It is easy to see that our analysis/distinguisher violates this bound. We argue that the distribution assumed for case 5/case 1 - class $\mathcal{B}$ erroneously underestimates the probability of certain bad events, and by changing the distribution to account for these bad events, the proof argumentation falls apart. Besides, it is not clear how to do so in the existing proof framework using the χ^2 method.

In particular, we look at the term $4l/2^{2n}$. The term stems from the following argument in [2]:

"It remains to bound $\Pr[\textbf{Inter} \in \mathcal{A} | \mathcal{Q}_{l-1}]$. *For this, note that once the values in* **Inter** *except for* (S_l, W_l) *have been fixed, the number of choices for* (S_l, W_l) *is at least* $(2^n - \alpha(\mathcal{Q}_{l-1}))(2^n - \gamma(\mathcal{Q}_{l-1})) \geq 2^{2n}/4$, *where* $\alpha(\mathcal{Q}_{l-1}) \geq q \geq 2^n/2$ *and* $\gamma(\mathcal{Q}_{l-1}) \geq q \geq 2^n/2$ *are the number of distinct values in* $(S_1, \ldots S_{l-1})$ *and* $(W_1, \ldots W_{l-1})$. *Out of these* $\geq 2^{2n}/4$ *choices, the number of choices that ensure the desired property* $TNT(T_l, X_l) = Y_l$ *is at most* $l-1$, *which results from the following selection process: we first pick a pair of input-oput* (U_i, V_i) *with* $i \leq l-1$, *and then set* $S_l = T_l \oplus U_i$ *and* $W_l = T_l \oplus V_i$. *Therefore,* $\Pr[\textbf{Inter} \in \mathcal{A} | \mathcal{Q}_{l-1}] \leq 4l/2^{2n}$,*and thus the upper bound in this case is*

$$\frac{4l}{2^{2n}} + \frac{1}{2^n - l}".$$

Consider the first case of the 4-way multi-collision in Figure 4, which we recall in Figure 6. We note that if the triplet (δ, S_o, U_o) is known, then the collision happens with probability 1, which puts it in class $\mathcal{A}$. Then, what remains is to calculate what is the probability that the adversary can force this collision, *i.e.,*

$$\Pr[\textbf{Inter} \in \mathcal{A} | \mathcal{Q}_{l-1}] = \Pr[U_e \oplus U_o = T_1 \oplus T_2 | \mathcal{Q}_{l-1}],$$

where T_1 and T_2 are determined by the adversary during previous queries. This means than once U_o in **Inter** is fixed (both U_o and U_e belong to a queries

$i, j < l$), U_e has at most $2^n - 1 - \alpha(\mathcal{Q}_{l-1})$ choices[3], where $\alpha(\mathcal{Q}_{l-1}) \leq q \leq 2^{n-1}$ is the number of distinct values in $\{U_1, \ldots U_l\} \setminus \{U_o, U_e\}$ only 1 of them enforces the collision. In other words,

$$\begin{aligned}
&\Pr[\textbf{Inter} \in \mathcal{A} | \mathcal{Q}_{l-1}] = \\
&\Pr[U_e \oplus U_o = T_1 \oplus T_2 | \mathcal{Q}_{l-1}] \\
&\geq \frac{1}{2^n - 1 - \alpha(\mathcal{Q}_{l-1})} \\
&\geq \frac{1}{2^n - 1} \gg \frac{4l}{2^{2n}},
\end{aligned}$$

when $l \ll q$, contradicting Equation (9) of [2].

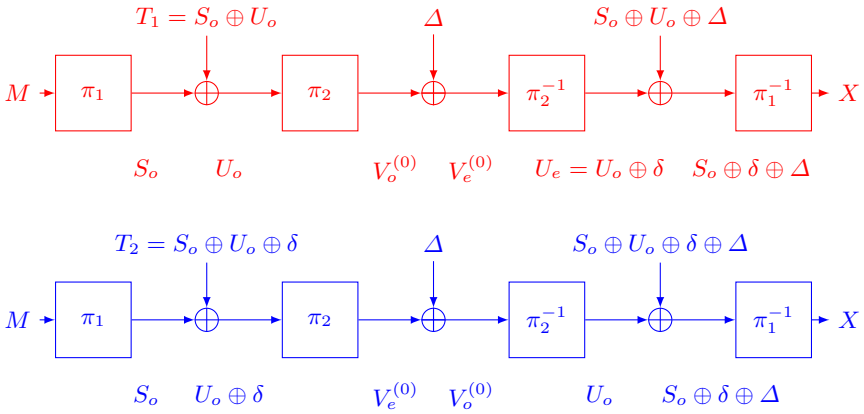

Fig. 6. A class $\mathcal{A}$ Collision.

Note that the values of V_i and W_i for $i < l$ did not affect the behaviour of the collision or the probability that **Inter** is in class $\mathcal{A}$. It seems the ambiguity may stem from applying the χ^2 method to a primitive with two dependent functions ($\tilde{E}$ and its inverse). By cascading forward and backward queries, we managed to eliminate W_i for all $1 \leq i \leq q$ and the values of W_l do not matter for the attack. Similarly, by fixing the difference between V_o and V_e to a constant Δ, we minimize the effect of their exact values on the attack.

A potential fix of this issue could be to add a tweak dependent operation after π_3, to prevent π_3 and π_3^{-1} from cancelling each other out. However, such solution may introduce new issues and is beyond out scope of study. On the other hand, we argue that fixing the proof using the exact same method is

[3] We use the notation of [2] in this part.

neither required nor needed, since [22] already provides a birthday bound proof and our distinguisher shows its tightness.

8 Conclusion

In this paper, we studied the security of the TNT tweakable block cipher, showing an adaptive chosen ciphertext attack with birthday bound complexity. This contradicts the claims made by the designers, and answers an important open research question on the exact security of the CLRW1[3]. We show that it has tight birthday bound security. We believe our observations on the use of the statistics of random permutations will be beneficial in the analysis of other schemes, especially those whose security is based on statistical techniques such as the χ^2 method.

The designers of TNT were motivated by a simple question: *what is the minimum number of iterations (or tweak addition) required to produce a secure TBC (especially those with BBB security), with provable security?* They answered this question as 3 iterations and 2 tweak additions. However, in the light of our work, the answer to this question in the BBB case has to be deferred to [22], giving answer of 5 iterations and 4 additions to get $2n/3$-bit security.

As future work, we plan to investigate the application of observations on the statistics of random permutations in the analysis of other constructions, including hash-based TBCs. We give an example of using such statistics to show the lack of simple collision counting distinguishers against the XOR-PRF construction. Another interesting research direction is investigating the χ^2 method more closely to understand why the proof in [2] failed, and how to correctly estimate distributions to avoid these pitfalls in the future.

A Counting Collisions in the XOR-PRF Construction

The XOR-PRF construction is a simple method to convert a BC into a PRF using two calls to the BC and a single key. In the analysis of the construction, the BC is replaced by a single random permutation. Assume π is a random permutation, then the XOR-PRF is given by

$$f(M) \leftarrow \pi(0\|M) \oplus \pi(1\|M).$$

Several efforts have been presented to prove its security and applications [7, 5, 4, 10, 18, 20], with the best bound on the form $q/2^n$. This bound is obviously tight, as pointed out in [7]. The adversary can simply query all 2^{n-1} possible inputs and observe if any of the outputs is 0^n, which is impossible using XOR-PRF. It is still interesting to find if there are other distinguishers. In this appendix, we show that it is unlikely that distinguishers based on simple collision counting exist. We do so by relying on similar observations to those used in Algorithm 1. Note that each query defines a difference equation as

$$\pi(0\|M) \oplus \pi(0\|M \oplus \delta) = \Delta.$$

This time, the input difference is always fixed to $\delta = 1\|0^{n-1}$. The query also already gives two solutions to the equation: $0\|M$ and $1\|M$. If the equation $(1\|0^n, \Delta)$ has only two solutions, then it is impossible to find another input to the XOR-PRF construction that outputs Δ. If the number of solutions $s \geq 4$, then the query is part of a multi-collision of size:

$$\binom{s/2}{2}.$$

The expected number of collisions in the output space of XOR-PRF can be counted as

$$2^n \times \sum_{s>2} \Pr[s] \times \binom{s/2}{2} =$$

$$2^n \times \sum_{i>1} \frac{0.5^i e^{-0.5}}{i!} \times \binom{i}{2} =$$

$$2^n \times \sum_{i>1} \frac{0.5^i e^{-0.5}}{(i-2)! \times 2} =$$

$$2^n \times 0.5^3 \sum_{i>1} \frac{0.5^{i-1} e^{-0.5}}{(i-2)!} =$$

$$2^n \times 0.5^3 = 2^{n-3}.$$

In the ideal world, when XOR-PRF is replaced by a random function, the expected number of collision in the output space is

$$\frac{\binom{2^{n-1}}{2}}{2^n} = \frac{2^{n-1}(2^{n-1}-1)}{2^{n+1}} = \frac{2^{n-1}-1}{2^2} = 2^{n-3} - \frac{1}{2},$$

which implies that there is very small difference in the expected number of collisions between XOR-PRF and a random function, and it is unlikely that even with 2^{n-1} queries a distinguisher based on collision counting has a significant advantage. In fact, not only does the number of collisions match that of a random function, but also the distribution of szes of multi-collision, *e.g.,* the expected number of outputs with 0 occurences, 1 occurrence, 2 occurences, 3 occurences,... etc. It is interesting to find a distinguisher for the XOR-PRF construction when the ideal random function is defined over the output space $\{0,1\}^n \setminus \{0\}$. However, collision counting may not be the way to find such distinguisher, which attests to the soundness of the construction.

B Python Implementation of Algorithm 1

```python
from numpy import random
import numpy as np
from random import randint
from math import sqrt
from tqdm import trange

# Select the world: 1: Real, 0: Ideal
#world = randint(0,1)
world = 0

# TBC block size
n = 28

c = 2
# Attack complexity
q = (1 << int(n/2+c))

# Distinguisher threshold
theta = (1 << (2*c-1)) + (1 << (2*c-2))

# Generate 3 random permutations
pi1 = np.array([_ for _ in range(1<<n)])
random.shuffle(pi1)

pi2 = np.array([_ for _ in range(1<<n)])
random.shuffle(pi2)

pi3 = np.array([_ for _ in range(1<<n)])
random.shuffle(pi3)

# Generate the inverse of each of the 3 permutations
beta1 = np.argsort(pi1)
beta2 = np.argsort(pi2)
beta3 = np.argsort(pi3)

# TNT
def enc (m,t):
    c = 0
    c = pi1[m] ^ t
    c = pi2[c] ^ t
    c = pi3[c]

    return c

# TNT^-1
def dec (m,t):
    c = 0
    c = beta3[m] ^ t
    c = beta2[c] ^ t
    c = beta1[c]
```

```
51
52      return c
53
54  # Fixed message
55  m0 = randint(0,((1<<n)-1))
56
57  # A list to store the multi-collisions.
58  L = [[] for _ in range((1<<n))]
59
60  # Collision counter
61  cnt = 0
62
63  # Delta ensures all the queries in the attack have different tweaks
64  # Helps simplify the ideal case, as no tweaks are repeated and we can
65  # simply sample a random value each time.
66  delta = q
67
68  # A list of the number of collisions on value c
69  coll = np.zeros((1<<n))
70
71  # For every iteration, take a random tweak and itself xor delta, and
72  # make an encryption call followed by a decryption call.
73  # Due to the uniqueness of tweaks, in the ideal case this amounts to
74  # sampling a uniform block.
75  print("Attack Started")
76  for _ in trange(q):
77      t0 = _
78      t1 = t0 ^ delta
79      c = enc(m0,t0)
80      c = dec(c,t1)
81      if (world == 0):
82          c = randint(0,(1<<n)-1)
83
84      coll[c] = coll[c] + len(L[c])
85      cnt = cnt + len(L[c])
86      L[c].append([t0,t1,c,m0])
87
88  # Guess the world
89  if (cnt >= theta):
90      guess = 1
91  else:
92      guess = 0
93
94  print("cnt:",np.mean(cnt))
95
96  if (world == 0):
97      print("Answer: Ideal World")
98  else:
99      print("Answer: Real World")
100
```

```
if (world == guess):
    print("Success")
else:
    print("Fail")
```

C C++ Implementation of Algorithm 1 on TNT-Small-Present-16

This implementation needs to be integrated into the library at [17].

```

#include <cstdint>
#include <gtest/gtest.h>

#include "ciphers/small_present16.h"
#include "utils/utils.h"
#include "ciphers/tnt_small_present16.h"

#include <stdlib.h>
#include <time.h>
#include <iostream>

using namespace std;

using ciphers::tnt_small_present16_context_t;
using ciphers::tnt_small_present16_state_t;
using ciphers::tnt_small_present16_tweak_t;
using ciphers::tnt_small_present16_key_t;

// ----------------------------------------------------------

int main(int argc, char **argv) {

  tnt_small_present16_key_t k;
  tnt_small_present16_state_t m,x,xe;
  tnt_small_present16_state_t t;
  tnt_small_present16_context_t ctx;

  int i, q, c, j;
  int L [65536];
  int coll[10000];
  float avg;
  int world;
  int guess;
  int success;

  srand(time(NULL));

  success = 0;
```

```
40
41  world = rand()%2;
42
43  for (i = 0; i < 10000; i++) {
44    coll[i] = 0;
45  }
46  q = 1 << 11;
47  for (j = 0; j < 10000; j++) {
48    world = rand()%2;
49    for (i = 0; i < 6; i++) {
50      k[i] = rand()%256;
51    }
52    for (i = 0; i < 65536; i++) {
53      L[i] = 0;
54    }
55    for (i = 0; i < 2; i++) {
56      m[i]  = rand()%256;
57    }
58
59    tnt_small_present16_key_schedule(&ctx, k);
60
61    for (i = 0; i < q; i++) {
62      t[0] = i & 0xff;
63      t[1] = (i>>8) & 0xff;
64      tnt_small_present16_encrypt(&ctx, t, m, x);
65      t[1] ^= 0x80;
66      tnt_small_present16_decrypt(&ctx, t, x, xe);
67      c = xe[0] + (xe[1] << 8);
68      if (world == 0) {
69  c = rand()%65536;
70      }
71      coll[j] += L[c];
72      L[c] += 1;
73    }
74    guess = 1;
75    if (coll[j] < 48) {
76      guess = 0;
77    }
78    if (guess == world) {
79      success++;
80    }
81  }
82
83  cout << "success: " << success << endl;
84
85 }
```

References

1. Avanzi, R.: The qarma block cipher family. almost mds matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. IACR Transactions on Symmetric Cryptology pp. 4–44 (2017)
2. Bao, Z., Guo, C., Guo, J., Song, L.: Tnt: how to tweak a block cipher. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 641–673. Springer (2020)
3. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The skinny family of block ciphers and its low-latency variant mantis. In: Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II 36. pp. 123–153. Springer (2016)
4. Bellare, M., Impagliazzo, R.: A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to prp to prf conversion. IACR Cryptol. ePrint Arch. **1999**, 24 (1999)
5. Bhattacharya, S., Nandi, M.: A note on the chi-square method: A tool for proving cryptographic security. Cryptography and Communications **10**, 935–957 (2018)
6. Daemen, J., Rijmen, V.: Probability distributions of correlation and differentials in block ciphers. Journal of Mathematical Cryptology **1**(3), 221–242 (2007)
7. Dai, W., Hoang, V.T., Tessaro, S.: Information-theoretic indistinguishability via the chi-squared method. In: Annual International Cryptology Conference. pp. 497–523. Springer (2017)
8. Guo, C., Guo, J., List, E., Song, L.: Towards closing the security gap of tweak-and-tweak (tnt). In: Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part I 26. pp. 567–597. Springer (2020)
9. Guo, Z., Wang, G., Dunkelman, O., Pan, Y., Liu, S.: Tweakable sm4: How to tweak sm4 into tweakable block ciphers? Journal of Information Security and Applications **72**, 103406 (2023)
10. Iwata, T.: New blockcipher modes of operation with beyond the birthday bound security. In: Fast Software Encryption: 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers 13. pp. 310–327. Springer (2006)
11. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: The deoxys aead family. Journal of Cryptology **34**(3), 31 (2021)
12. Lampe, R., Seurin, Y.: Tweakable blockciphers with asymptotically optimal security. In: International Workshop on Fast Software Encryption. pp. 133–151. Springer (2013)
13. Landecker, W., Shrimpton, T., Terashima, R.S.: Tweakable blockciphers with beyond birthday-bound security. In: Annual Cryptology Conference. pp. 14–30. Springer (2012)
14. Leander, G.: Small scale variants of the block cipher present. Cryptology ePrint Archive (2010)
15. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22. pp. 31–46. Springer (2002)

16. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. Journal of cryptology **24**, 588–613 (2011)
17. List, E.: TNT · GitLab — gitlab.com. https://gitlab.com/elist/tnt, [Accessed 08-08-2023]
18. Lucks, S.: The sum of prps is a secure prf. In: Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19. pp. 470–484. Springer (2000)
19. O'Connor, L.: On the distribution of characteristics in bijective mappings. In: Advances in Cryptology—EUROCRYPT'93: Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings 12. pp. 360–370. Springer (1994)
20. Patarin, J.: A proof of security in $o(2^n)$ for the xor of two random permutations. In: International Conference on Information Theoretic Security. pp. 232–248. Springer (2008)
21. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 16–31. Springer (2004)
22. Zhang, Z., Qin, Z., Guo, C.: Just tweak! asymptotically optimal security for the cascaded lrw1 tweakable blockcipher. Designs, Codes and Cryptography **91**(3), 1035–1052 (2023)