Survey on the Effectiveness of DAPA-Related Attacks against Shift Register Based AEAD Schemes

Shivam Bhasin[1], Dirmanto Jap[1], Wei Cheng Ng[2], Siang Meng Sim[2]

[1] Temasek Laboratories, NTU Singapore
[2] DSO National Laboratories, Singapore
{sbhasin,djap}@ntu.edu.sg,{weicheng.derrick,crypto.s.m.sim}@gmail.com

1 Introduction

NIST Lightweight Cryptography (LWC) standardization process was initiated in August 2018 to solicit, evaluate, and standardize lightweight cryptographic algorithms that are suitable for use in constrained environments. The algorithm shall implement authenticated encryption with associated data (AEAD) functionality and optionally the hashing functionality.

In March 2021, NIST announced ten finalists and is currently in the final round of the standardization process, with one of the main focuses being the side-channel evaluation of the finalists. Besides performance evaluation of the finalists with side-channel protections/countermeasures implemented, side-channel analysis (SCA) of the finalists is also a consideration factor for the standardization process.

1.1 Side-channel Attacks on Shift Register based Primitives

For evaluating the security of cryptographic algorithms, in addition to theoretical vulnerability, one should also investigate the physical characteristics when implemented on a practical device. This device dependent attack is commonly known as the side-channel analysis or SCA in short [KJJ99]. In this attack, any physical characteristics, such as timing, power consumption, electromagnetic emanation, can be exploited to recover secret information, such as the secret key of a block cipher implementation. In general, the procedure is to measure the physical leakage of the target device while executing the cryptographic algorithm, which leaks sensitive information about the algorithm.

While block ciphers have been the primary target of SCA, some stream ciphers have also been scrutinized for resistance to SCA. While most attacks on stream ciphers aim to recover the internal state, SCA provides a great advantage to an attacker by giving direct information about the internal state, for example the internal of linear/non-linear feedback shift register (LFSR/NFSR) state, as leaked in side-channel traces. Availability of such internal information can greatly simplify the otherwise high complexity theoretical cryptanalysis of stream ciphers.

For SCA (as well as countermeasures) on stream ciphers, there is a survey by Rechberger and Oswald [RO04]. Several SCAs have also been reported on some stream ciphers, for example, Fisher et al. [FGKV07] who proposed SCA on GRAIN and Trivium. Dobraunig et al. [DEKM17] then demonstrated that SCA can also be used on shift registers to extract the bit relations of neighbouring bits, allowing attacker to significantly reduce the internal state guessing space, by applying an attack on Keymill. Based on their work, a

later work from Sim et al. [SJB21] generalised their methodology and combined it with differential analysis, which is referred as Differential Analysis aided Power Attack (DAPA), to uncover more bit relations and take into account the linear or non-linear functions that feedback to the shift registers (i.e. LFSRs or NFSRs). Sim et al. then reported their attacks on LR-Keymill (an improved version of Keymill to resist attack from [DEKM17]) and Trivium. The work is then followed by Kumar et al. [KDB+22], who then developed a generic automated framework for SCA on stream ciphers.

We outline the key idea of how side-channel can be applied to learn information about internal state of a stream cipher as used in [DEKM17, SJB21, KDB+22]. As a toy example, consider the leakage when there is data switching, which can be modeled as Hamming Distance (HD) model. In Figure 1, one can observe that taking the difference between two traces, the HD difference can be easily distinguished on side-channel traces, even through visual inspection. In a noisier scenario where the boundaries would be overlapping, the adversary can either build a simple statistical distinguisher or use advanced approaches like machine or deep learning, with prior profiling. The adversary can then use this information to identify each HD value, which already provide some information regarding the state of the intermediate value being processed by the device.

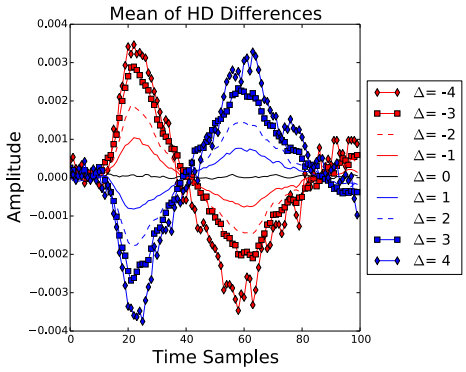

Figure 1: Mean for each class of HD difference

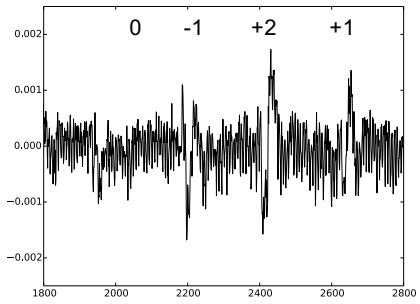

Figure 2: Investigating ΔHD to distinguish sign of the difference

1.2 Motivation and Contributions

Among the ten NIST LWC finalists, Grain-128AEADv2 and TinyJAMBU use LFSR/NFSR as the building blocks for their primitive. Thus, we are interested to investigate their resistance against SCA, specifically against DAPA. Although both finalists can be implemented in 32-bit architecture, in extremely constrained environments, 1-bit architecture (e.g. bit-slice implementation) may be used to minimise the resources needed. Thus, we analyse both finalists in both 1-bit and 32-bit settings.

Our results are summarised in the Table 1.

Table 1: Summary of DAPA on Shift Register based NIST finalists. 32* refers to recovering 32 bits of the secret internal register state.

Finalist	Key size	Implementation	Bits recovered	Ref.
Grain-128AEADv2	128	1-bit	128	Sect. 3.2
		32-bit	64	Sect. 3.3
TinyJAMBU	128/192/256	1-bit	128/192/256	Sect. 4.2
		32-bit	32*	Sect. 4.3

Organisation. First, we revisit DAPA and highlight the main observations that we will be using for our analysis in Section 2. Next, we give a brief description of Grain-128AEADv2

in Section 3.1, followed by our analysis on its 1-bit and 32-bit implementation in Section 3.2 and 3.3 respectively. We give a brief description of TinyJAMBU is provided in Section 4.1 and our analysis on its 1-bit and 32-bit implementation in Section 4.2 and 4.3 respectively. Finally, we conclude our work in Section 5.

2 Preliminaries

2.1 Revisiting DAPA

DAPA is proposed by Sim et al. [SJB21] as an extension to the attack by Dobraunig et al. [DEKM17]. The former extended the attack beyond the first clock cycle after some difference is injected, and investigated the power consumption behaviour under both the linear and non-linear feedback functions.

2.1.1 Notations

A feedback function typically consists of common binary operations. In this study, we only consider 3 of them — `AND` ($\wedge$), `NAND` ($\overline{\wedge}$), and `XOR` ($\oplus$).

Let $[x]y$ denote a register bit of interest in the square parenthesis with bit value x, and y is the succeeding bit value. A hat symbol $\hat{x}$ denote having a difference, which is simply flipping of the bit value.

We define the power consumption difference the subtraction of the original power trace from the power trace with some differences. If a register bit has an increase in power consumption difference, we denote it as $+1$, -1 if it is a decrease, and 0 if there is no difference in the power consumption difference. In practice, the power trace is the summation of the power consumption of all the register bits. Hence, we can apply simple arithmetic to compute the combined power consumption difference.

2.1.2 Differential patterns and bit relations

[SJB21] detailed various differential configurations, how the power consumption are sufficient and necessary conditions to derive the bit relations or values. For our analysis, we only need the following observations (see Table 2).

Table 2: Power consumption differences and relations of register bits

Case	Possible outcomes	Power diff.
1.1: $[x]y$ vs $[x]\hat{y}$	$x = y$	$+1$
	$x \neq y$	-1
1.2: $[x]y$ vs $[\hat{x}]y$	$x = y$	$+1$
	$x \neq y$	-1
1.3: $[x]y$ vs $[\hat{x}]\hat{y}$	$x = y$	0
	$x \neq y$	

2.1.3 On non-linear operations

As explained in [SJB21], when a feedback bit is the product of some non-linear operation, say $w = x \wedge y$, and there is a difference in y (a.k.a. active), then value of x will determine if there is a difference in w. Using this knowledge, we can deduce the actual value of x (0 or 1 respectively) given the knowledge of whether there is a difference in w (inactive or active respectively).

Lemma 1. *Let $\circ$ be either* `AND` *($\wedge$) or* `NAND` *($\overline{\wedge}$) and $x \circ y$ be a single bit value. We have the following observation.*

Case	Possible outcomes	Power diff.
$[z]x \circ y$ vs $[z]x \circ \hat{y}$	$x = 1, x \circ y = z$	$+1$
	$x = 0$	0
	$x = 1, x \circ y \neq z$	-1
$[x \circ y]z$ vs $[x \circ \hat{y}]z$	$x = 1, x \circ y = z$	$+1$
	$x = 0$	0
	$x = 1, x \circ y \neq z$	-1

Proof. This is a direct combination of Case 1.1 or 1.2 and the observation on non-linear operation. Since $\circ$ is `AND` or `NAND`, $x = 0$ will result in a constant value (0 or 1 respectively) regardless of the value of y, and if $x \circ y$ is inactive, then naturally there is no power consumption differences. Otherwise, $x = 1$ and $x \circ y$ is active and we get either Case 1.1 or 1.2. □

It is also trivial to see that if there is a difference in $x \circ y$, then the contribution to the power difference will be ± 1, which changes the parity (from odd to even or vice versa) of the overall power consumption difference. Thus, if we only want to know the value of x, it is sufficient to only check the parity of the power consumption difference.

2.1.4 Attack phases

The DAPA methodology can be broken down into the following three steps:

Step 1 (Offline): Determine the differential patterns. In this preparation phase, the goal is to choose a differential pattern that we would want to have in the shift register. Although the choice is highly dependent on the target algorithm, there is a general strategy.

One obvious entry point is through the IV[1], which essentially every (N)LFSR-based algorithm should have. The main idea is to introduce some difference in the IV and analyse how it would propagate throughout the internal state.

This step would take up a significant amount of time as the attack complexity depends heavily on the selected differential patterns. Generally, there is no need to find optimal differential patterns, so long as the execution, say introducing the different IVs, is feasible and the attack complexity is practical, we are good to move to the next step.

Step 2 (Online): Perform the power measurements. This is the only online phase of the attack and rather straightforward — collect the power traces of various computations and take the difference to obtain power consumption differences.

Step 3 (Offline): Recover the internal state. In this step, we try to gather as many pieces of bit relations to link the internal state bits together. From the rise, drop or no change in power consumption (collected in Step 2) of the differential patterns (determined in Step 1), we can deduce the bit relations.

Finally, after gathering as many bit relations as we can, we enumerate the possible values for the leading bit in each chained bit relation, other bits within the chain will be defined according to the bit relations. The true internal state will be one of these candidates.

[1] Or something equivalent that is public and preferably can be chosen.

3 Grain-128AEADv2

3.1 Specification

Grain-128AEADv2 [HJM+] is an AEAD scheme with the underlying primitive based on Grain-128a, which is in turn based on Grain v1 and Grain-128 that are extensively analysed. It takes a variable-length plaintext and associated data, a 96-bit nonce (IV) $IV_0IV_1...IV_{95}$, and a 128-bit key $k_0k_1...k_{127}$. It consists of a pre-output generator and an authenticator generator. It starts with a "Key and Nonce Initialisation" phase before starting the "Operating Mode" to process input associated data or plaintext.

Our analysis targets the "Key and Nonce Initialisation" phase which only involves the pre-output generator. Thus for brevity sake, we only describe the pre-output generator and the relevant updating steps of the "Key and Nonce Initialisation" phase.

The pre-output generator is constructed using an 128-bit LFSR (denoted $S_t = [s_0^t, s_1^t, ..., s_{127}^t]$), an 128-bit NFSR (denoted $B_t = [b_0^t, b_1^t, ..., b_{127}^t]$) and a pre-output function, where t refers to the clock cycle. Note that the computations are defined over GF(2), thus $a + b$ represents `XOR` ($a \oplus b$) and ab represents `AND` ($a \wedge b$).

During the "Key and Nonce Initialisation" phase, the NFSR is loaded with the key bits $b_i^0 = k_i$, $0 \leq i \leq 127$ and the first 96 bits of the LFSR is loaded with the nonce bits $s_i^0 = IV_i$, $0 \leq i \leq 95$, the last 32 bits are loaded with 31 ones and a zero $s_i^0 = 1$, $96 \leq i \leq 126$, $s_{127}^0 = 0$. It is then updated 320 times with the feedback bit for LFSR and NFSR defined as follows:
For $0 \leq t \leq 319$,

$$\begin{aligned} s_{127}^{t+1} &= \mathcal{L}(S_t) + y_t \\ b_{127}^{t+1} &= s_0^t + \mathcal{F}(B_t) + y_t, \end{aligned}$$

where

$$\begin{aligned} \mathcal{L}(S_t) &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t, \\ \mathcal{F}(B_t) &= b_0^t + b_{26}^t + b_{56}^t + b_{91}^t + b^t 96 + b_3^t b_{67}^t + b_{11}^t b_{13}^t \\ &\quad + b_{17}^t b_{18}^t + b_{27}^t b_{59}^t + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t \\ &\quad + b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t \\ y^t &= h^t + s_{93}^t + b_2^t + b_{15}^t + b_{36}^t + b_{45}^t + b_{64}^t + b_{73}^t + b_{89}^t \\ h^t &= b_{12}^t s_8^t + s_{13}^t s_{20}^t + b_{95}^t s_{42}^t + s_{60}^t s_{79}^t + b_{12}^t b_{95}^t s_{94}^t \end{aligned}$$

It is then updated for another 64 times with key material reintroduced into the pre-output generator. However, since our analysis does not need to go beyond the initial 320 updates, we omit the description for the rest of Grain-128AEADv2 and refer readers to [HJM+] for more details.

3.2 DAPA on 1-bit Implementation

3.2.1 Implementation setup

We consider an implementation similar to the reference implementation provided by the Grain-128AEADv2 designers, which can be found in the submission package
...\grain-128aead\Implementations\crypto_aead\grain128aeadv2\ref\grain128aead-v2.c

Specifically, we measure the power trace at line 144: `fsr[127] = fb;`, where `fsr[127]` is the current bit s_{127}^t and `fb` is the succeeding bit s_{127}^{t+1}.

3.2.2 Overview of Full Key-recovery using DAPA

We are able to develop a chosen nonce DAPA attack that is able to perform full key recovery on the 1-bit implementation of Grain-128AEADv2. This attack targets the key and nonce initialization phase of the scheme. In particular, we will target the h non-linear function in the pre-output generator.

There are three main properties of the 1-bit implementation of the Grain-128AEADv2 pre-output generator that render it susceptible to DAPA:

1. Low algebraic degree of the nonlinear combiner h, and the low weight of variables from the NFSR in h;
2. Direct application of nonce into the shift register during initialization; and
3. Slow diffusion of the non-linear component in the LFSR.

We shall now provide a rough sketch of the attack, along with the key insights that motivate the important steps along the way. We describe our attack in incremental phases, where the techniques performed in each phase utilize all the techniques and properties acquired in prior phases. Our attack compares the difference in the power differential of each 1-bit clock during the initialization phase at the rightmost 1-bit register of the LFSR, i.e. s_{127}, between a pair of differently initialized registers. We do so by considering pairs of registers initialized by nonces with a single bit difference. In particular, we define our reference nonce to be the zero nonce, i.e. $n_0 := 0^{96}$, and consider all nonces by their difference with respect to n_0. We define p_i to be the nonce pair $(n_0, n_0 \oplus e_i)$, where e_i is the 96-bit vector that is 1 at register i and 0 everywhere else. For the subsequent analyses, we shall use the following notation:

$b_i^t(n)$ (resp. $s_i^t(n)$) refers to the value of the i-th register of the NFSR (resp. LFSR) at clock t given nonce n, where $0 \leq i < 128$.

$h^t(n)$ (resp. $y^t(n)$) refers to the value of the non-linear combiner output h (resp. the non-linear feedback y) at clock t given nonce n.

$b_j(n)$ (resp. $s_j(n)$) refers to the value of the 0-th register of the NFSR (resp. LFSR) at clock j given nonce n, for any $j \geq 0$. Since Fibonacci shift registers are used, we note that $b_j(n) = b_i^{j-i}(n)$ (resp. $s_j(n) = s_i^{j-i}(n)$) for any $0 \leq i < \max(j+1, 128)$.

$\bar{x}$ refers to the value of a component x when the nonce is the zero nonce, i.e. $\bar{x} = x(n_0)$.

3.2.3 Phase 1: Deducing Values of LFSR Registers and y^t for the First 70 Clocks

In this phase, we apply DAPA iteratively on successive clocks of the cipher initialization, starting from the clock $t = 0$. At each clock t, we hope to recover the value of the register $\overline{s_{128+t}}$, the full internal state of the LFSR initialized with the zero nonce at $t+1$, as well as the non-linear feedback into the LFSR y^t. We proceed by induction on t.

We first note that since we control the nonce and because the nonce is applied directly into the internal state of the LFSR during initialization, we know the full internal state of the LFSR at $t = 0$. This provides us with the base case for our induction. To prove the inductive step, we consider the nonce pair p_k. By the inductive hypothesis, we know the full internal LFSR state for the zero nonce for all $0 \leq t \leq k$. We see that the input difference in our nonce pair propagates to a difference at s_{127}^{k+1}, assuming that there is no difference in the other tapped registers for the LFSR feedback and no difference in the tapped registers for the non-linear feedback function y^k. Therefore, if $s_{127}^k(e_k) = s_{127}^k(n_0)$, we can use the DAPA characteristic of Case 1.1 in Section 2.1.2 to recover the value of register $s_{127}^{k+1} = \overline{s_{128+k}}$. Furthermore, since we know the full internal state of the LFSR initialized with the zero nonce at $t = k$ by the inductive hypothesis, we will know the full internal state of the LFSR at $t = k+1$ and the value of the linear feedback bit $\mathcal{L}(S_k)$. This allows us to obtain the value $\overline{y^k}$.

We now prove the assumptions stated in the inductive step above. In order to ensure that there is no difference in the other tapped registers for the LFSR feedback besides

s_0^k, and no difference in the tapped registers for the non-linear feedback function y^k at $t = k$, we need to track the propagation of our input difference through the registers across $0 \leq t < k$. The difference propagation patterns for different nonce pairs p_k for $0 \leq k < 70$ is summarized in Table 3 below.

Table 3: Grain-128AEADv2 Active Registers at Clock $t = k$ for Various Nonce Pairs. **Bold font** indicates a register from the LFSR, while normal font indicates a register from the NFSR. Underline indicates a possible active register due to propagation through a prior non-linear function. The set S_{118+} comprises of some registers from both the LFSR and NFSR with every index 118 or above.

Nonce Pair p_k	Set of active registers at clock $t = k$
$0 \leq k < 7$	$S_1 = \{\mathbf{s_0}\}$
$k = 7$	$S_2 = S_1 \cup \{\mathbf{s_{121}}\}$
$8 \leq k < 38$	$S_3 = S_2 \cup \{b_{120}, \mathbf{s_{120}}\}$
$39 \leq k < 42$	$S_4 = S_3 \cup \{\mathbf{s_{90}}, \mathbf{s_{122}}, b_{124}, \mathbf{s_{124}}, b_{125}, \mathbf{s_{125}}\}$
$42 \leq k < 70$	$S_5 = S_4 \cup \{b_{86}, \mathbf{s_{86}}\} \cup S_{118+}$

We can easily verify from Table 3 above that none of the tapped registers for the LFSR feedback besides s_0^k, and none of the tapped registers for the non-linear feedback function y^k are active at $t = k$, thus our assumption holds.

We are left to show that $s_{127}^k(e_k) = s_{127}^k(n_0)$. This is done by similarly showing that none of the tapped registers for the LFSR feedback (including s_0^k), and none of the tapped registers for the non-linear feedback function y^k are active, this time at $t = k - 1$ instead of $t = k$. We summarize the difference propagation patterns for different nonce pairs p_k for $0 \leq k < 70$ up to clock $t = k - 1$ in the Table 4 below.

Table 4: Grain-128AEADv2 Active Registers at Clock $t = k - 1$ for Various Nonce Pairs. **Bold font** indicates a register from the LFSR, while normal font indicates a register from the NFSR. Underline indicates a possible active register due to propagation through a prior non-linear function. The set S_{119+} comprises of some registers from both the LFSR and NFSR with every index 119 or above.

Nonce Pair p_k	Set of active registers at clock $t = k - 1$
$0 \leq k < 7$	$S_1 = \{\mathbf{s_1}\}$
$k = 7$	$S_2 = S_1 \cup \{\mathbf{s_{122}}\}$
$8 \leq k < 38$	$S_3 = S_2 \cup \{b_{121}, \mathbf{s_{121}}\}$
$39 \leq k < 42$	$S_4 = S_3 \cup \{\mathbf{s_{91}}, \mathbf{s_{123}}, b_{125}, \mathbf{s_{125}}, b_{126}, \mathbf{s_{126}}\}$
$42 \leq k < 70$	$S_5 = S_4 \cup \{b_{87}, \mathbf{s_{87}}\} \cup S_{119+}$

Once again, we see that there is no active bit being feedback into the LFSR at $t = k-1$, and the condition that $s_{127}^k(e_k) = s_{127}^k(n_0)$ holds. We have thus shown that we are able to recover the full internal state of the LFSR, as well as the input from non-linear feedback function $\overline{y^k}$ given our reference nonce for the first 70 clocks.

3.2.4 Phase 2: Obtaining 85 NFSR Register Values by Exploiting Quadratic Terms in the Non-Linear Combiner h

We see that the non-linear combiner h comprises of 4 quadratic terms and 1 cubic term. These are all of very low degree and are prime targets to target using the DAPA characteristics described in Section 2.2. In particular, we shall exploit the quadratic terms that combine NFSR and LFSR inputs, $b_{12}s_8$ and $b_{95}s_{42}$, via Lemma 1.

Therefore, we select input nonce pairs p_{k+8} (resp. p_{k+42}) such that the difference propagates to s_8^k (resp. s_{42}^k). Then at clock $t = k$, we perform DAPA to determine whether $\overline{s_{127}^{k+1}} = s_{127}^{k+1}(e_{k+8})$ (resp. $s_{127}^{k+1}(e_{k+42})$)). Assuming that there is no difference in the other

tapped registers for the LFSR feedback and the non-linear feedback function y^k, we can deduce that $b_{12}^k = \overline{s_{127}^{k+1}} \oplus s_{127}^{k+1}(e_{k+8})$ (resp. $b_{95}^k = \overline{s_{127}^{k+1}} \oplus s_{127}^{k+1}(e_{k+42})$). We are able to perform this technique for $0 \leq k < 34$ to target $b_{12}^k s_8^k$, and $0 \leq k < 51$ to target $b_{95}^k s_{42}^k$, before things begin to get messy.

3.2.5 Phase 3: Recovering the Full Internal State and Key

Thus far, we have obtained the internal LFSR register values $\overline{s_k}$ for $0 \leq k < 198$, and the non-linear feedback function values $\overline{y^k}$ for $0 \leq k < 70$ from Phase 1. Furthermore, we are able to derive 85 internal NFSR register values $\overline{b_k}$ for $12 \leq k < 46$ and $95 \leq k < 146$ from Phase 2. We now combine this information to recover the full internal state for both the LFSR and NFSR with the reference nonce, and roll back this internal state to recover the full key.

Note that the non-linear feedback function $\overline{y^k}$ for $0 \leq k < 70$ gives us 70 non-linear equations in 319 variables; 163 from the NFSR and 156 from the LFSR. By selecting only the equations defined by $\overline{y^k}$ for $0 \leq k < 51$, we get 51 equations in the variables $\overline{b_i}$ for $2 \leq i < 146$ and $\overline{s_i}$ for $8 \leq i < 145$. Applying the values of b_i and s_i that we have already recovered thus far, we manage to obtain 51 ***linear*** equations[2] in the 58 unknown variables b_i for $2 \leq i < 11$ and $46 \leq i < 95$. By considering the equation for the NFSR feedback from $12 \leq t < 27$, we can obtain an additional 15 non-linear equations of degree at most 3 in the same 58 variables. This gives us 66 equations in 58 variables which we can solve to get the full cipher internal state for clocks $2 \leq t < 18$, allowing us to recover the key with an additional 15-bit check.

3.2.6 Summary

In summary, we demonstrated that it is possible to construct a full key recovery attack on the 1-bit implementation of Grain-128AEADv2 using DAPA techniques. This attack requires 94 instantiations of the cipher with different nonces (via the nonce pairs p_k for $0 \leq k < 93$) under the chosen nonce, noiseless scenario. Additionally, our attack provides an additional 15-bit check by recovering a sliding window of 16 contiguous full internal register states, which allows us to verify and possibly rectify the accuracy of the power traces recovered.

3.3 DAPA on 32-bit Implementation

3.3.1 Implementation setup

We consider an implementation similar to the optimised implementation provided by the Grain-128AEADv2 designers, which can be found in the submission package
...\grain-128aead\Implementations\crypto_aead\grain128aeadv2\sse\grain128aead-v2_opt.cpp

Specifically, we measure the power trace at line 100: `L32(12) XOR= ks;`, where `L32(12)` is the current 32-bit register $[s_{96}^t s_{97}^t ... s_{127}^t]$ and `ks` is the 32-bit feedback value[3] $[y_t y_{t+1} \cdots y_{t+31}]$.

3.3.2 64-bit Key-recovery using DAPA

The 32-bit implementation of Grain-128AEADv2 helps it to gain some resistance against DAPA. Due to the fact that the registers are clocked 32 times per step, we have to

[2] We manage to reduce the non-linear terms in h to linear terms by substituting in known values of $\overline{s_t^k}$ and $\overline{b_{95}^k}$ for $0 \leq k < 51$

[3] Note that the LFSR feedback value $\{\mathcal{L}(S^t)\}_{t=0}^{t=31}$ are updated separately in the implementation which simplifies our analysis a bit. But even if they are updated simultaneously, our analysis will still work as $\mathcal{L}$ is linear and we only need to account for differential patterns contributed by that.

consider the cumulative effect of all taps within each 32-bit step, instead of isolating the effect of our target difference at the desired clock. Additionally, it can be difficult to track the propagation of differences of individual registers and its effect on the power trace, especially through the interference of other intermediate bit taps. In particular, it is challenging to track the effect of an input difference of a nonce pair through more than one 32-bit step, since after 64 1-bit clocks, the difference would probably propagate through the registers s_{94} and b_{95}, which are taps for the non-linear feedback function. This means that any power analysis we perform beyond the first step will have to take the undesired contributions of differences through non-linear functions into account, which greatly increases the complexity of any potential attack.

Nevertheless, we are still able to recover 64 non-contiguous bits of the NFSR from the optimized implementation of Grain-128AEADv2 relatively easily. In this scenario, our attack compares the difference in the power differential of each 32-bit clock during the initialization phase at the rightmost 32-bit slice of the LFSR, i.e. s_{96} to s_{127}, between a pair of differently initialized registers. We aim to vary the LFSR registers s_8 and s_{42} to target the NFSR registers b_{12} and b_{95} respectively, akin to Phase 2 of the attack described in Section 3.2 above. We do so by considering the nonce pairs p_{8+k} and p_{42+k} for $0 \leq k < 32$, clocking the cipher through a single 32-bit step, and taking the power trace of the rightmost register. Given these conditions, the only other positions that contribute to power differences in the rightmost register are from the LFSR taps s_0, s_7, s_{38} and s_{70}, which can be accounted for easily. The key insight here is to use the parity of the difference in power traces between our nonce pair to determine whether a difference in s_8^k or s_{42}^k triggered a difference in s_{127}^{k+1}.

As an example, suppose we are targeting the register s_8^0 to determine b_{12}^0. In order to do this, we consider the nonce pair p_8. After one 32-bit step, the difference in the nonce pair is guaranteed to propagate through s_7^1 and s_0^8, and may possibly propagate through s_8^0. Each propagated difference will contribute a difference of ± 1 to the power traces of our nonce pair. Therefore, by the property in Section 3.2.2, we derive the value of b_{12}^0 as 0 if the power trace difference between our nonce pair is even, and 1 if the power trace difference is odd.

Following this line of reasoning, we are able to recover the NFSR register values b_{12+k} and b_{95+k} for $0 \leq k < 32$, which correspond to 64 bits of the key.

3.3.3 Summary

Here, we have shown a relatively straightforward method to partially recover the key from a 32-bit implementation of Grain-128AEADv2 using DAPA techniques. This attack requires 65 instantiations of the cipher with different nonces (via the nonce pairs p_k for $8 \leq k < 40$ and $42 \leq k < 74$) to recover 64 key bits under the chosen nonce, noiseless scenario. This section reveals that even though the 32-bit implementation helps Grain-128AEADv2 to be more robust against a direct application of DAPA-based attacks, it is still somewhat vulnerable. We believe that there is potential to improve key yield through a more in-depth study of this approach, and work is currently in progress to enhance this attack even further.

4 TinyJAMBU

4.1 Specification

TinyJAMBU [WH] is an AEAD scheme based on JAMBU which is one of the finalists of the CAESAR competition [cae]. It has three variants with key size 128 bits, 192 bits and 256 bits respectively $k_0 k_1 ... k_{klen-1}$ where $klen$ is the key size, all variants takes in a 96-bit

nonce $n_{127}n_{126}...n_{32}$[4]. The TinyJAMBU mode is similar to a Duplex mode and uses a keyed permutation as the underlying permutation.

The keyed permutation consists of a 128-bit state $S_t = [s^t_{127}s^t_{126}...s^t_0]$. An update is defined as

$$s^{t+i+1}_{127} = s^{t+i}_0 \oplus s^{t+i}_{47} \oplus (s^{t+i}_{70} \overline{\wedge} s^{t+i}_{85}) \oplus s^{t+i}_{91} \oplus k_{i \bmod klen},$$

where i is the i-th update (index from 0) on the state S_t.

The initialisation phase consists of a "Key setup" and a "Nonce setup". The "Key setup" initialises an all zero state S_{-1664}[5], and update that state 1024 times. Next, in the "Nonce setup", the nonce is partitioned into three 32-bit blocks $N0 = n_{127}...n_{96}$, $N1 = n_{95}...n_{64}$, and $N2 = n_{63}...n_{32}$. After XORing a 3-bit Framebits (value 1) to the state S_{-640}, update the state for another 640 times before XORing $N0$ to $s^0_{127}...s^0_{96}$, $s^0_i = s^0_i \oplus n_i$, $96 \leq i \leq 127$. This process is repeated for another two times to take in $N1$ and $N2$. However, since our target window is within the next 640 updates (from S_0 to S_{640}) and before $N1$ is XORed to the state, we omit the description for the rest of TinyJAMBU and refer readers to [WH] for more details.

4.2 DAPA on 1-bit Implementation

4.2.1 Implementation setup

We consider an implementation where only 1 bit is updated at every step and assume we are able to measure the power trace of the register s_{127}. Although such implementation is not provided by the designers, in extremely constrained environment, 1-bit architecture might be deployed.

4.2.2 Full Key-recovery using Simple Power Analysis (SPA)

Since the internal state S_{-1664} starts from an all zero state, we could potentially apply SPA during the "Key setup" to recover the key materials bit by bit. However, since this setup is independent of the nonce, one could perform pre-computation and stored the state for future computation, or for scenarios with fixed key, the resultant state S_{-640} can be hard-coded as the initial state and omit the "Key setup" completely.

The "Nonce setup", however, has to be computed every time for a nonce, thus this is a more practical window of opportunity for SCA.

4.2.3 Full Key-recovery using DAPA

As with Grain-128AEADv2, we shall target the cipher (keyed permutation P_n) used in the nonce setup phase of the TinyJAMBU to perform our DAPA analysis. In general, TinyJAMBU is slightly more resistant to DAPA than Grain-128AEADv2 under a chosen nonce adversarial model, primarily due to the following two reasons. Firstly, the LFSR is not loaded into the register directly during the initialization phase, but injected over multiple rounds with intermediate state updates P_{640} in between. This means that we can only affect the input of 32 registers of the NFSR at a time, as opposed to 96 registers in the case of Grain-128AEADv2. Secondly, the non-linear and linear contributions to the keyed permutation originate from the same register, and there is no component to the cipher that has strictly linear feedback like the LFSR used in Grain-128AEADv2. This makes it more difficult for us to isolate characteristics from either the non-linear component by only tracing differences in the linear component (as we did in Section 3.2.2). Nevertheless, we demonstrate that we are able to recover the full key TinyJAMBU

[4] We use a different indexing from the original specification [WH] for the ease of our discussion.

[5] For the ease of our discussion, we define the internal state where our attack begins as S_0, using that point as a reference, the initial state is denoted as S_{-1664}.

with a 1-bit implementation with the aid of SPA, and at least 32 bits of the internal state of the keyed permutation of TinyJAMBU with both the optimized and unoptimized 32-bit implementations. While there is no 1-bit implementation of TinyJAMBU in its specifications submitted to NIST, we describe the attack on this variant of TinyJAMBU as a study of its side-channel resistance alongsideGrain-128AEADv2, and to hopefully deter the community from misusing TinyJAMBU in a context with constraints that only permit a 1-bit implementation.

The DAPA analysis on TinyJAMBU is more straightforward than that of Grain-128AEADv2. In this analysis, we shall focus our attention on the only non-linear component of the keyed permutation: the operation $s_{85} \bar{\wedge} s_{70}$. As before, we shall introduce the following notation:

$s_i^t(n)$ refers to the value of the i-th register of the NFSR at clock t given nonce n, where $0 \leq i < 128$.

$s_j(n)$ refers to the value of the 0-th register of the LFSR at clock j given nonce n, for any $j \geq 0$. Since Fibonacci shift registers are used, we note that $s_j(n) = s_i^{j-i}(n)$ for any $0 \leq i < \max(j+1, 128)$.

$\bar{x}$ refers to the value of a component x when the nonce is the zero nonce, i.e. $\bar{x} = x(n_0)$.

Our attack compares the difference in the power differential of each 1-bit clock at the rightmost 1-bit register of the NFSR, i.e. s_{127}, between a pair of differently initialized registers. We do so by considering pairs of registers initialized by nonces with a single bit difference. In particular, we define our reference nonce to be the zero nonce, i.e. $n_0 := 0^{96}$, and consider all nonces by their difference with respect to n_0. We define p_i to be the nonce pair $(n_0, n_0 \oplus e_i)$, where e_i is the 96-bit vector that is 1 at register i and 0 everywhere else. We shall consider the nonce pairs p_i for $0 \leq i < 32$ for our attacks on TinyJAMBU. The difference in the nonce will be introduced to the NFSR via the last operation in the first iteration of the nonce setup phase during initialization. This allows us to target the power difference at 1-bit clocks of the state update using P_{640} in the second iteration of the nonce setup. We thus take the initial clock $t = 0$ to refer to the state at the beginning of the second iteration of P_{640} in the nonce setup,

For each nonce pair p_k, we record the power trace at clocks $5+k, 11+k, 26+k, 42+k, 48+k$, and $63+k$. Using the DAPA characteristic of Case 1.1 in Section 2.1.2, we are able to obtain the bit differential at register $s_{127}^{k+1} = s_{128+k}$ between our nonce pair, assuming the register s_{127}^k is not active. This bit differential is contributed by the feedback taps that are active registers at clock $t+k$. We consolidate a list of active registers at each of the aforementioned clocks in the Table 5 below.

Table 5: TinyJAMBU Keyed Permutation Active Registers at Clock t for Various Nonce Pairs p_k. **Bold font** indicates a register that will be tapped by the feedback function. Underline indicates a possible active register due to propagation through a prior non-linear function. Note that s_{127}^k is not active at any of these clocks.

Clock t	Set of indices i such that s_i^k is active
5	{**91**}
11	{**85**, 122}
26	{**70**, 107, 113}
42	{54, **91**, 97, 112}
48	{48, **85**, **91**, 106, 122}
63	{33, **70**, 76, **91**, 107, 113, 114, 117}

We shall now explain how the active feedback taps at each of the above clocks yield information about the internal state of the keyed permutation for $0 \leq k < 32$.

1. At clock $t = 5 + k$, our nonce difference gets propagated to a solitary feedback tap s_{91}^t. This allows us to use Case 1.1 and 1.2 to determine whether $\overline{s_{127+t}} = \overline{s_{128+t}}$, and whether $\overline{s_{128+t}} = \overline{s_{129+t}}$.

2. At clock $t = 11 + k$, our nonce difference gets propagated to a solitary feedback tap at s^t_{85}. This allows us to use Lemma 1 to recover the value of $\overline{s_{70+t}}$.

3. At clock $t = 26 + k$, our nonce difference gets propagated to a solitary feedback tap at s^t_{70}. This allows us to use Lemma 1 to recover the value of $\overline{s_{85+t}}$.

4. At clock $t = 42 + k$, our nonce difference gets propagated to a solitary feedback tap at s^t_{91} via the initial feedback at clock $5 + k$. This allows us to use Case 1.1 and 1.2 to determine whether $\overline{s_{127+t}} = \overline{s_{128+t}}$, and whether $\overline{s_{128+t}} = \overline{s_{129+t}}$.

5. At clock $t = 48 + k$, our nonce difference gets propagated to the feedback tap at s^t_{85} via the initial feedback at clock $5 + k$, and also possibly propagated to the feedback tap at s^t_{91} via the initial feedback at clock $11 + k$. However, note that due to (2) we know whether or not the register s^t_{91} is active. This allows us to subtract away the contribution of the s^t_{91} register to the power trace difference at s_{128+t}, and use Lemma 1 to recover the value of $\overline{s_{85+t}}$.

6. At clock $t = 63 + k$, our nonce difference gets propagated to the feedback tap at s^t_{70} via the initial feedback at clock $5 + k$, and also possibly propagated to the feedback tap at s^t_{91} via the initial feedback at clock $26 + k$. However, note that due to (3) we know whether or not the register s^t_{91} is active. This allows us to subtract away the contribution of the s^t_{91} register to the power trace difference at s_{128+t}, and use Lemma 1 to recover the value of $\overline{s_{70+t}}$.

When taken together, this information allows us to derive the internal state of the 122 contiguous registers $\overline{s_{81}}$ to $\overline{s_{202}}$, with an additional 60-bit check[6]. We thus have to guess 6 bits of information to derive the entire internal state for a certain clock $76 \leq t \leq 81$. For each 6 bit guess of the entire internal state for a certain clock t, we perform SPA on the NFSR for clocks $t \leq k \leq t + K$ to extract the entire key of size K.

4.2.4 Summary

In summary, we demonstrated that it is possible to construct a full key recovery attack on the 1-bit implementation of Grain-128AEADv2 using DAPA and SPA techniques. This attack requires 33 instantiations of the cipher with different nonces (via the nonce pairs p_k for $96 \leq k < 128$), and a 6-bit guess under the chosen nonce, noiseless scenario. Additionally, our attack provides an additional 60-bit check through an overlap in the recovered register values, which allows us to verify our 6-bit guess, and confirm and possibly rectify the accuracy of the power traces recovered.

4.3 DAPA on 32-bit Implementation

4.3.1 Implementation setup

We consider an implementation similar to the reference implementation provided by the TinyJAMBU designers, which can be found in the submission package
...\tinyjambu\Implementations\crypto_aead\tinyjambu128v2\ref\encrypt.c

Specifically, we measure the power trace at line 34: `state[3] = feedback;` where `state[3]` is the current 32-bit register $[s^t_{127}s^t_{126}...s^t_{96}]$ and `feedback` is the 32-bit feedback value $[s^{t+32}_{127}s^{t+31}_{127}...s^{t+1}_{127}]$.

The optimized implementation provided by the designers is similar except the update is in-place, meaning the 32 feedback bits replace the values in the register with the 32 bits that are pushed out of the state. Nevertheless, our analysis can be applied to both the reference and optimized implementations.

[6] We obtain a 60-bit check from the overlap in register values recovered in Steps 1 to 6.

4.3.2 32-bit Internal State-recovery using DAPA

Similar to GRAIN-128 AEAD, the 32-bit implementation of TinyJAMBU is more robust against our DAPA cryptanalysis efforts than its 1-bit implementation. There are two variants for the 32-bit implementations of TinyJAMBU submitted by the designers: the optimized variant updates the 32-bit slices of the NFSR in-place, while the reference variant performs a cyclical shift update of the 32-bit slices. In this section, we shall apply DAPA to recover 32 bits of the internal NFSR state for both variants of the 32-bit TinyJAMBU implementation. We define the variables and the initial clock $t = 0$ as in Section 4.2.

In this scenario, our attack compares the difference in the power differential of each 32-bit step during the second iteration of the P_{640} update in the nonce setup phase, at the rightmost 32-bit slice of the NFSR, i.e. s_{96} to s_{127}, between a pair of differently initialized registers. The gist of the attack is again to target the non-linear feedback taps s_{85} and s_{70}. We are able to induce a difference in a single *non-linear* feedback tap (this does not preclude the possibility of differences in linear feedback taps), s_{85}^{k-85}, if we select nonce pairs p_k for $102 \leq k < 117$ and clock them through a single 32-bit step. This allows us to recover the corresponding 15 bits of the internal state $\overline{s_{70}^{k-85}}$ using Lemma 1. Additionally, if we select nonce pairs p_k for $112 \leq k < 117$, we are similarly able to induce a difference in a single non-linear feedback tap, $\overline{s_{70}^{k-70}}$, when we clock them through a second 32-bit step, thus allowing us to recover the corresponding 5 bits of the internal state s_{85}^{k-70}.

However, when we consider nonce pairs for p_k for k outside the above ranges, things begin to become complicated, due to the fact that each 32-bit slice now has power difference contributions from more than one register with a non-linear feedback difference. We are able to rectify this multiple interference for the case $96 \leq k < 102$, by considering **rectangular differentials**.

For nonce pairs p_k, $96 \leq k < 102$, the register difference will propagate through s_{91}^{k-91}, s_{85}^{k-85}, and s_{70}^{k-70} over a single 32-bit step. The effect of this difference on the power trace is cumulative across the non-linear contributions s_{85}^{k-85} and s_{70}^{k-70}, thus by Lemma 1 we know the following: if the power trace difference in our nonce pair is even, then either $s_{70}^{k-85} = 1$ or $s_{85}^{k-70} = 1$, but not both. Conversely, if the power trace difference in our nonce pair is odd, then either $s_{70}^{k-85} = 0$ and $s_{85}^{k-70} = 0$, or $s_{70}^{k-85} = 1$ and $s_{85}^{k-70} = 1$. Unfortunately, this does not give us the definitive value for either of these variables. Rectangular differentials help to isolate these non-linear contributions, in order to obtain the values of these variables individually.

Rectangular differentials work by considering a set of 4 cipher instantiations with 4 different nonce values. We have our reference zero nonce, n_0, and a nonce each with a difference at registers a, b, and both a and b, which we shall label n_a, n_b and n_{ab} respectively. Let the power trace difference for nonce pairs p_a, p_b and p_{ab} at clock $t = a - 70 = b - 91$ be $\mathcal{P}_a, \mathcal{P}_b$ and $\mathcal{P}_{ab}$ respectively. For this particular scenario, we shall target $96 \leq a < 102$ and $b = a + 21$. When we consider the nonce pair p_b, we see that the difference propagates to just a single feedback tap, the linear feedback tap s_{91}^{b-91}, after a single 32-bit step. Hence, we know the effect of injecting a difference at $s_{128+b-91} = s_{128+a-70}$ on the power trace, whether it is $+1$ or -1. Next, we consider the nonce pair p_a. As illustrated in the above paragraph, the parity of the power trace difference will reveal some information on the values of the targeted bits. Finally, we consider the nonce pair p_{ab}. We see that if $s_{70}^{a-70} s_{85}^{a-70}$ was active, then this difference would be cancelled out by the active bit at s_{91}^{b-91}, and conversely if $s_{70}^{a-70} s_{85}^{a-70}$ was inactive then there would be a new difference induced by s_{91}^{b-91}. This means that $\overline{s_{85}^{a-70}} = 1 \Leftrightarrow \mathcal{P}_{ab} = \mathcal{P}_a - \mathcal{P}_b$, and $\overline{s_{85}^{a-70}} = 0 \Leftrightarrow \mathcal{P}_{ab} = \mathcal{P}_a + \mathcal{P}_b$. This leads us to the differential Table 6.

We can thus obtain the internal register states $\overline{s_{85}^{k-70}}$ and $\overline{s_{70}^{k-85}}$ for $96 \leq k < 102$. Combined with our earlier analysis on nonce pairs p_k for $102 \leq k < 117$, we can recover a total of 32 internal register states $\overline{s_t}$ for $81 \leq t < 102$, $111 \leq t < 117$, and $127 \leq t < 132$.

Table 6: Rectangular Differential Table for $96 \leq a < 102$, $b = a + 21$.

$(\mathcal{P}_a \mod 2, \mathcal{P}_{ab} - \mathcal{P}_a)$	$\left(\overline{s_{85}^{a-70}}, \overline{s_{70}^{a-85}}\right)$
$(0, \mathcal{P}_b)$	$(1, 0)$
$(0, -\mathcal{P}_b)$	$(0, 1)$
$(1, \mathcal{P}_b)$	$(0, 0)$
$(1, -\mathcal{P}_b)$	$(1, 1)$

4.3.3 Summary

From our analysis thus far, we observe that it can be more challenging to apply DAPA techniques to recover the key from the 32-bit implementation of TinyJAMBU, than for its 1-bit implementation. We find that it is rather straightforward to recover 32-bits of the internal state of the keyed register during the nonce setup phase, through DAPA rectangular differential analysis. Further investigation is required to extend this line of attack to recover more bits.

5 Conclusion

In this work, we conducted side-channel analysis on LFSR/NFSR based AEAD schemes from NIST LWC finalists, namely Grain-128AEADv2 and TinyJAMBU. We analysed how DAPA can be used to recover secret information.

For both finalists, their 1-bit implementation is susceptible to DAPA and the full key could be recovered. The 32-bit implementation, however, makes the analysis significantly harder because the power trace collected is the accumulation of multiple bit updates, making it non-trivial to isolate and learn the bit relation or value. Work is currently in progress to enhance the attack on the 32-bit variants to recover more secret bits.

Through our analysis, we find that it is more challenging to recover the internal state of TinyJAMBU as compared to Grain-128AEADv2 because we are limited to gaining information using only 32 bits of the nonce, the injection of the next 32 bits of nonce occurs after 640 keyed updates, making it difficult to correlate any bit information obtain from injecting differences in the next 32 bits of the nonce. This reminds us of ISAP that uses rate r_B to bound the amount of leakage.

We hope this work will help in better understanding the side-channel resilience of LFSR/NFSR based cryptographic primitives.

References

[cae] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/ caesar-submissions.html.

[DEKM17] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, and Florian Mendel. Side-Channel Analysis of Keymill. In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2017.

[FGKV07] Wieland Fischer, Berndt M. Gammel, O. Kniffler, and Joachim Velten. Differential power analysis of stream ciphers. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 2007.

[HJM+] Martin Hell, Thomas Johansson, Alexander Maximov, Willi Meier, Jonathan Sönnerup, and Hirotaka Yoshida. Grain-128AEADv2 - A lightweight AEAD stream cipher. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/grain-128aead-spec-final.pdf.

[KDB+22] Satyam Kumar, Vishnu Asutosh Dasu, Anubhab Baksi, Santanu Sarkar, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Side channel attack on stream ciphers: A three-step approach to state/key recovery. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):166–191, 2022.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[RO04] Christian Rechberger and Elisabeth Oswald. Stream ciphers and side-channel analysis. In *In SASC 2004 – The State of the Art of Stream Ciphers*, pages 320–326, 2004.

[SJB21] Siang Meng Sim, Dirmanto Jap, and Shivam Bhasin. DAPA: differential analysis aided power attack on (non-) linear feedback shift registers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):169–191, 2021.

[WH] Hongjun Wu and Tao Huang. TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms (Version 2). https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/tinyjambu-spec-final.pdf.