# `Transistor`: a TFHE-friendly Stream Cipher

Jules Baudrin[1,7], Sonia Belaïd[4], Nicolas Bon[1,4,5], Christina Boura[3], Anne Canteaut[1], Gaëtan Leurent[1], Pascal Paillier[4,6], Léo Perrin[1], Matthieu Rivain[4], Yann Rotella[2], and Samuel Tap[6]

[1] Inria, France
[2] University of Versailles St Quentin, France
[3] IRIF, Université Paris Cité, France
[4] CryptoExperts, France
[5] DIENS, Ecole normale supérieure, PSL University, CNRS, Paris, France
[6] Zama, France
[7] UCLouvain, Louvain-la-Neuve, Belgium

**Abstract.** Fully Homomorphic Encryption (FHE) allows computations on encrypted data without requiring decryption, ensuring data privacy during processing. However, FHE introduces a significant expansion of ciphertext sizes compared to plaintexts, which results in higher communication. A practical solution to mitigate this issue is *transciphering*, where only the master key is homomorphically encrypted, while the actual data is encrypted using a symmetric cipher, usually a stream cipher. The server then homomorphically evaluates the stream cipher to convert the encrypted data into a homomorphically encrypted form.

We introduce `Transistor`, a stream cipher specifically designed for efficient homomorphic evaluation within the TFHE scheme, a widely-used FHE framework known for its fast bootstrapping and ability to handle low-precision data. `Transistor` operates on $\mathbb{F}_{17}$ which is chosen to optimize TFHE performances. Its components are carefully engineered to both control noise growth and provide strong security guarantees. First, a simple TFHE-friendly implementation technique for LFSRs allows us to use such components to cheaply increase the state size. At the same time, a small *Finite State Machine* is the only part of the state updated non-linearly, each non-linear operation corresponding in TFHE to a rather expensive *Programmable Bootstrapping*. This update is done using an AES-round-like transformation. But, in contrast to other stream ciphers like `SNOW` or `LEX`, our construction comes with information-theoretic security arguments proving that an attacker cannot obtain any information about the secret key from three or fewer consecutive keystream outputs. These information-theoretic arguments are then combined with a thorough analysis of potential correlations to bound the minimal keystream length required for recovering the secret key.

Our implementation of `Transistor` significantly outperforms the state of the art of TFHE transciphering, achieving a throughput of over 60 bits/s on a standard CPU, all while avoiding the need for an expensive initialization process.

## 1 Introduction

*Fully Homomorphic Encryption (FHE)* refers to cryptographic systems that enable computations to be performed directly on encrypted data, without needing to decrypt it first. For instance, imagine Alice wants to use an online service to analyze her confidential medical data. With traditional encryption methods, the service would need to decrypt her data before processing it, potentially exposing sensitive information. In contrast, FHE allows the service to perform the analysis while the data remains encrypted, ensuring privacy and security throughout the entire process. Using FHE, Alice can encrypt her medical data and send the encrypted version (ciphertext $C$) to the online service. The service can then perform the necessary computations on $C$ and produce a new ciphertext $C'$. When $C'$ is decrypted, the result will be the same as if the operations had been applied directly to the original, unencrypted data.

Modern FHE schemes rely on encrypting data with added noise to ensure security, but, as computations are performed, the noise grows. To maintain accuracy and prevent errors, they must manage and reduce the noise during the computation process, typically through techniques like bootstrapping. Different FHE schemes are available in the literature, such as BGV/FV [48,20], CKKS [26], and TFHE [28,29,30]. Each of them offers different features in terms of efficiency, parallelization, noise management, and the types of plaintext and operations they support. In this paper, we focus on TFHE (Torus FHE) which achieves bootstrapping with much lower latency compared to other FHE schemes, making it highly-suitable for real-time applications involving complex computations.

One common challenge with all FHE schemes is that the ciphertexts are much larger than the corresponding plaintexts. For example, a plaintext message of a few kilobytes can require tens or even hundreds of megabytes of data, making the processing of large data sets impractical. While compression techniques can help reduce the expansion factor in TFHE ciphertexts, the encrypted data still remains an order or two of magnitude larger than the original plaintext.

*Transciphering.* It is possible to mitigate this issue using *transciphering* [66]. The idea is to off-load the task of actually encrypting the data to a symmetric cipher, and to simply encrypt homomorphically the key that is used. The user then sends both the homomorphically encrypted key and the ciphertext to the server, which can then homomorphically decrypt the received ciphertexts. This is done by running a fully homomorphic evaluation of the decryption function of the symmetric cipher, producing valid homomorphic ciphertexts representing the data. The server can then proceed to the homomorphic operations, like in the traditional FHE setting. This principle is illustrated in Figure 1.

Implementing FHE encryption through transciphering solves the bandwidth issue: the data sent by the client to the server is encrypted using a symmetric cipher, thus avoiding the significant ciphertext expansion implied by direct FHE encryption. The only exception is the symmetric key, which does experience expansion, but this overhead is amortized across the entire data set.

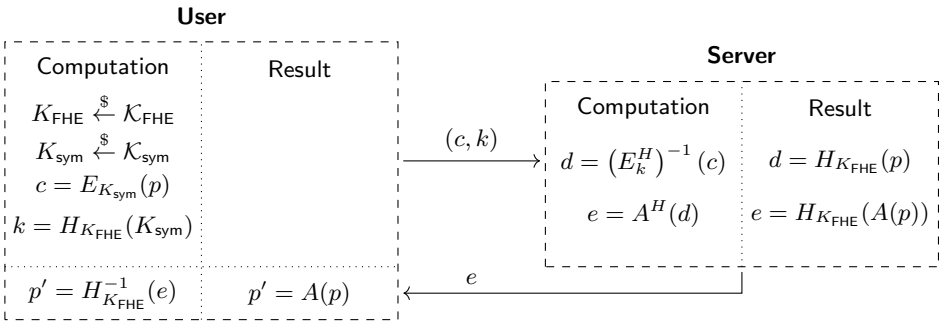
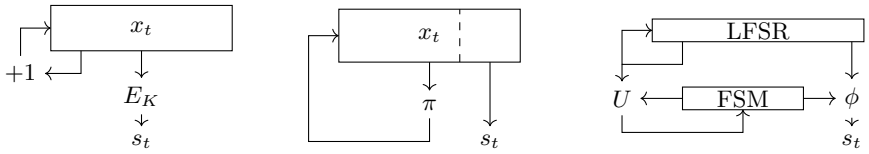



**Fig. 1:** The principle of transciphering, where $E$ is a symmetric cipher (with secret key $K_{\mathsf{sym}}$ sampled from the space $\mathcal{K}_{\mathsf{sym}}$), $H$ is a fully homomorphic cipher (with private key $K_{\mathsf{FHE}}$ sampled from the space $\mathcal{K}_{\mathsf{FHE}}$), $E^H$ is a homomorphic evaluation of $E$, $A$ corresponds to some arbitrary operations, and $A^H$ to their homomorphic evaluation.

However, we need the symmetric cipher to interact well with the FHE scheme, as its decryption circuit must be homomorphically evaluated on the server side. This induces a significant computational overhead with standard ciphers like AES, which, according to the latest record [11], achieves only 4 bits per second at an error probability of $2^{-40}$. As a result, the relevant performance metrics differ significantly from traditional ones like RAM consumption or throughput per area. Previous works [22,62] have highlighted that a synchronous stream cipher is far more suitable for this application than a block cipher in CBC mode, which is why we adopt this approach.

*Building TFHE-friendly Stream Ciphers.* In addition to having the fastest bootstrapping among FHE schemes, TFHE has the unique feature of enabling the free evaluation of an arbitrary function on the underlying plaintext. Since any function can be evaluated at the same computational cost, the *Programmable Bootstrapping* (PBS) becomes a powerful tool for introducing non-linearity into the scheme, allowing for free composition with arbitrary table lookups. Consequently, designing a TFHE-friendly scheme requires carefully balancing the number of cheap linear operations, which provide diffusion and increase noise, with the application of the more expensive PBS, which permits introducing non-linearity and reduce the noise.

Several ciphers from the literature have been designed specifically to address such requirements. `Elisabeth` [35] generates the keystream by first extracting digits from the master key in a pseudo-random manner, and then applying a complex filter function that relies on numerous non-linear PBSs. More recently, the designers of `FRAST` [32] opted for a block cipher operating in counter mode. Their design utilizes a generalized Feistel network, where the round function applies key-dependent S-boxes. While `FRAST` significantly outperforms `Elisabeth` in terms of throughput, the key-dependent S-box setup results




**(a)** Counter mode. **(b)** Permutation-based. **(c)** LFSR and FSM.

**Fig. 2:** Different types of stream cipher structures.

in a very time-consuming re-keying. Finally, `Kreyvium` [22], a stream cipher inspired by `Trivium` [39] with 128-bit security, turned out to have competitive performance when used for TFHE transciphering [8], despite being originally designed for use with BGV/FV schemes.

All stream ciphers share a common structure: they consist of an internal state, an update function applied to that state, and a filtering function that extracts some bits of the keystream at each clock cycle. Either the update function, filtering function, or the initialization of the internal state must be dependent on the key (and a nonce or IV). This very general view covers for example the case of a block cipher in counter mode, where the internal state is a block cipher input $x_t$, the update function is a simple addition, and the filter is a very complex key-dependent operation (the block cipher encryption $E_k$), as illustrated in Figure 2a. This model also covers sponge-based stream ciphers [72], where the filtering function is trivial (just copying bits from the outer part of the sponge), but the update function is highly sophisticated, involving a cryptographic permutation $\pi$, as shown in Figure 2b.

More classical constructions based on LFSRs, NLFSRs, and nonlinear *Finite State Machines (FSMs)* are also covered and turn out to be more promising for us. The design strategies that lead to secure and efficient block-cipher-based or permutation-based constructions typically apply non-linear functions (S-boxes) to the entire internal state. However, in the case of TFHE, these operations are by far the most expensive. On the other hand, an LFSR initialized with secret key material can be efficiently implemented in a TFHE-friendly manner. Indeed, we store the initial state as low-noise key material and we generate directly the $t$-th output as a linear combination of the key material, instead of updating the LFSR state at each clock. Thus, we obtain an almost noise-free digit. This allows the design of a TFHE-friendly stream cipher with large LFSRs to increase the state size, and a smaller FSM where all non-linear operations are concentrated, see Figure 2c. Ciphers like `Grain` [56] or `SNOW` [47,46] follow this construction.

As designers, we want to ensure that the stream cipher's output is as decorrelated from the initial key material as possible. This makes cryptanalysis significantly harder: the lower the correlation, the more difficult it becomes to extract information about the master key from the knowledge of the keystream. A natural approach for reaching this goal was proposed in [74,3], but has rarely been used since. It consists in designing the stream cipher so that there exists no correlation between its internal state and as many consecutive outputs as possible.

For instance, the authors of `Rocca` [71] prove that an attacker must derive linear equations through four AES rounds to (potentially) be able to recover information about the master key. Biryukov used a different technique in `LEX` [15], but obtained a similar result: two consecutive outputs are uncorrelated.

*Our Contributions.* We present `Transistor`, a stream cipher optimized for transciphering with TFHE. This design is the outcome of a careful study of the constraints and advantages specific to achieving efficient homomorphic evaluations with TFHE. In particular, we argue that operating on elements of $\mathbb{F}_p$, where $p$ is a small prime (4–5 bits), is a good choice for leveraging the full potential of TFHE's programmable bootstrapping: we chose $p = 17$. This choice is independent of the data format supported by the application running on the server, as changes of representations are easily feasible through bootstrapping [13].

The design of `Transistor` uses two LFSRs, one generating a sequence of subkeys (the "key-schedule LFSR"), and the other producing a sequence of masks (the "whitening LFSR"). The subkeys are fed into the round function of a finite state machine (FSM), which is updated using an AES-round-like transformation. At the same time, parts of the internal state of this component are leaked at each clock cycle and added to the masks to generate keystream blocks.

This structure offers multiple advantages. As discussed above, it minimizes both the noise in the subkeys, and the number of PBSs needed to update its state. At the same time, the use of a block-cipher round function enables us to construct robust security arguments that borrow techniques from block-cipher design. In fact, we establish information-theoretic arguments demonstrating that an attacker cannot obtain any information about the content of the key-schedule LFSR from the observation of three consecutive keystream blocks. The fact that at least four consecutive output blocks are needed then provides an upper bound on the bias of any linear relation between the keystream digits and the sequence produced by the key-schedule LFSR, derived from the use of traditional components. Operating on elements of $\mathbb{F}_p$ ($p > 2$) allows us to use nice features of TFHE. However, it required a careful analysis of linear biases and correlations in $\mathbb{F}_p$. Our results in this area may be of independent interest.

We present a careful analysis of the noise evolution throughout the homomorphic evaluation of `Transistor`, to fine-tune the TFHE parameters for optimal performance. Our homomorphic implementation of `Transistor` significantly outperforms the state of the art, achieving a throughput of over 60 bits/s on a standard CPU. This represents a factor 3 speedup compared to `FRAST` [32], the fastest previous method, while also achieving a considerably lower error probability and eliminating the need for an expensive initialization phase.

*Outline.* The rest of the article is structured as follows. Section 2 covers the preliminaries for the TFHE scheme. Section 3 discusses the design constraints for a stream cipher intended for use with TFHE, along with the design choices we made. The specification of `Transistor` and the reasoning behind its design is detailed in Section 4. Section 5 provides an extensive security analysis against

several classes of attacks. Finally, Section 6 details the homomorphic implementation of our scheme, providing performance metrics and benchmarks.

## 2 Preliminaries

### 2.1 Notation

Let $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ be the real torus, that is to say the additive group of real numbers modulo 1. In practice, torus elements are not represented with an infinite number of digits, but are discretized. We can define the discretized torus $\mathbb{T}_q = \{\frac{a}{q} \mid a \in \mathbb{Z}_q\}$, and identify it with the ring $\mathbb{Z}_q$. Thus, any element $\frac{a}{q}$ of $\mathbb{T}_q$ will be represented in machine by $a$ without losing any properties of the group $\mathbb{T}_q$. The operations of sum $+$ and external product $\cdot$ have to be understood modulo $q$. We also denote by $\mathbb{B}$ the set $\mathbb{B} = \{0, 1\}$, which is trivially a subset of any $\mathbb{Z}_q$. Moreover, for a natural integer $N$ and a given $q$, we will denote by $\mathbb{T}_{N,q}[X]$ the polynomial ring $\mathbb{T}_q[X]/(X^N + 1)$. The elements of this ring are polynomials of maximum degree $N - 1$ and with coefficients in $\mathbb{T}_q$. Like for the scalar version, this ring will be identified with the ring $\mathbb{Z}_{N,q}[X] = \mathbb{Z}_q[X]/(X^N + 1)$. In the following, $N$ is a power of two. For $x$ and $q \in \mathbb{Z}$ , $[x]_q$ denotes the reduction of $x$ modulo $q$. We denote by $x \xleftarrow{\$} \chi$ a random sampling according to a distribution $\chi$. We denote by $\mathbb{F}_p$ the finite field with $p$ elements, i.e., $\mathbb{Z}_p$ when $p$ is prime, and refer to its elements as *digits*.

### 2.2 Preliminaries on TFHE

TFHE [28,29,30] is a homomorphic encryption scheme, designed as a successor to FHEW [42]. Its security is based on the Learning With Errors (LWE) problem. Optimized for operations on low-precision data (typically less than 6 bits), TFHE offers a distinctive feature: *programmable bootstrapping*. This enables the evaluation of any univariate function on a ciphertext while simultaneously resetting its noise to a nominal level. In what follows, we introduce TFHE, discuss its underlying complexity assumptions, describe the encoding and encryption procedures, and provide an overview of the homomorphic operations it supports.

Like many popular FHE schemes, TFHE relies on the LWE problem [70]. More specifically, it relies on a slightly different version of this problem: the space considered is a discretized torus, and the secret is binary. We refer to Appendix B.1 for a rigorous definition of this problem.

**Plaintext Space and Encryption.** The plaintext space is the *discretized torus* $\mathbb{T}_p$, that we identify to the ring $\mathbb{Z}_p$, with $p \in \mathbb{N}$. Let us consider a mapping $\rho : \mathbb{Z}_p \to \mathbb{Z}_q$, defined as $\rho : m \mapsto \left\lfloor \frac{mq}{p} \right\rceil$. The image of this mapping only reaches $p$ elements in $\mathbb{Z}_q$, which take the form $\left\{ \frac{kq}{p} \mid k \in \mathbb{Z}_p \right\}$. These elements are evenly distributed across $\mathbb{Z}_q$ and form what we refer to as *sectors of* $\mathbb{Z}_q$, represented as:

$\left\{\left(\frac{(2k-1)q}{2p}, \frac{(2k+1)q}{2p}\right) \mid k \in \mathbb{Z}_p\right\}$. TFHE features two types of encryption that share similar structural patterns but operate within different mathematical spaces.

*LWE Encryption.* Let $m \in \mathbb{Z}_p$ be a message and let $sk = (s_1, \ldots, s_n)$ represent the secret key, sampled uniformly at random from $\mathbb{B}^n$. First, the message $m$ is encoded in the space $\mathbb{Z}_q$ by $\tilde{m} = \rho(m)$. A small random Gaussian noise $e \xleftarrow{\$} \chi_\sigma$ is then added. Since $e$ is small, the noisy message $\tilde{m} + e$ remains within the same sector as $\tilde{m}$. Next, we construct the LWE ciphertext as a vector $c = (a_1, \ldots, a_n, b)$, where the $a_i$'s are sampled uniformly at random from $\mathbb{Z}_q$, and $b$ is defined by $b = \sum_{i=1}^{n} a_i \cdot s_i + \tilde{m} + e$.

Decryption has two steps: first, we compute $\phi(c) = b - \sum_{i=1}^{n} a_i \cdot s_i$, referred to as the *phase*. Then we round it to the nearest plaintext value: $\tilde{m} = \left\lfloor \frac{p}{q}\phi(c) \right\rceil$. As long as $|e| < \frac{q}{2p}$, this rounding produces the right sector center.

*GLWE Encryption. General LWE* encryption mirrors the structure of LWE encryption but operates within polynomial rings. The secret key $SK$ is here represented as a vector $(S_1, \ldots, S_k)$, sampled uniformly at random from $\mathbb{B}_{N,q}[X]^k$. The message is encoded in a polynomial in $\mathbb{Z}_{N,q}[X]$. The noise is also a polynomial from the same ring, with coefficients drawn from $\chi_\sigma$. Similar to LWE encryption, the ciphertext takes the form $C = (A_1, \ldots, A_k, B)$ where $B = \sum_{i=1}^{k} A_i \cdot S_i + \tilde{M} + E$.

It is worth noting that LWE encryption can be viewed as a special case of GLWE encryption, where $N = 1$ and $k = n$.

**Homomorphic Operations.** TFHE is trivially linearly homomorphic, so we define the following operations that apply to both types of ciphertexts. In what follows, we refer to the variance of the Gaussian random variable associated to the noise as the *noise variance*. This quantity is a measure of the "noise level" and should stay as low as possible.

*Sum of ciphertexts.* Let $c_1$ and $c_2$ be two ciphertexts encrypting the messages $m_1$ and $m_2$, respectively, with noise variances $\sigma_1^2$ and $\sigma_2^2$. Performing a coordinate-wise sum of the two vectors results in a valid ciphertext $c'$, which encrypts $m_1+m_2$ with noise variance $\sigma_1^2+\sigma_2^2$. We denote this operation by $\texttt{SumTFHE}(c_1, c_2)$.

*Product with a cleartext.* Let $c$ be a ciphertext encrypting $m$ with noise variance $\sigma^2$. Multiplying each coordinate of $c$ by a constant $\lambda \in \mathbb{Z}$ produces a valid ciphertext $c'$, which encrypts $m' = \lambda \cdot m$ with noise variance $\lambda^2 \cdot \sigma^2$. We denote this operation as $\texttt{ClearMultTFHE}(c', \lambda)$.

These linear operations are extremely fast, particularly in comparison to bootstrapping (which we will introduce below). However, they increase the noise variance (noise level), which means that only a limited number of such operations can be performed before the correctness of the results is compromised.

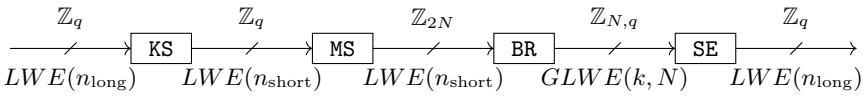



**Fig. 3:** Types and shapes of ciphertexts inside a PBS.

*Key Switching (KS).* TFHE also features a keyswitching algorithm, that allows the server to homomorphically transform a ciphertext $c_1$ encrypted under a key $s_1$ into a ciphertext $c_2$ encrypted under a key $s_2$. To do so, it requires a *keyswitching key* $KSK$, which is simply an encryption of $s_1$ under the key $s_2$. For more details, the reader is referred to [30]. This is particularly useful to temporarily reduce the size of a ciphertext by keyswitching it to a shorter key (but raising its noise), to enable some speed-ups in the bootstrapping algorithm.

*Programmable Bootstrapping (PBS).* Bootstrapping, introduced by Gentry in [49], is a generic technique that allows the noise of a ciphertext to be homomorphically reset to a nominal level. While this operation can theoretically be applied to any homomorphic encryption scheme, it is often deemed too slow for practical use. However, in TFHE, bootstrapping is relatively efficient compared to other fully homomorphic encryption (FHE) schemes, especially for low-precision messages, and it is implemented in a *programmable* manner. This means that while the noise is being reset, any arbitrary function can be evaluated on the ciphertext. We denote the evaluation of a PBS (Programmable Bootstrapping) on a ciphertext $c$ which evaluates the function $f$ as $\texttt{PBS\_TFHE}(c, f)$ and provide hereafter a high-level overview of how this process operates in practice.

Let $(a_1, \ldots, a_n, b)$ be the LWE encryption of a message with noise variance $\sigma^2$ and $(s_1, \ldots, s_n) \in \mathbb{B}^n$ represent the secret key. To reset the noise to a nominal level following Gentry's framework, the server must homomorphically evaluate $b - \sum_{i=1}^{n} a_i \cdot s_i$ and subsequently round the result to the nearest integer in $\mathbb{Z}_p$.

To facilitate this, the server is equipped with a *bootstrapping key*, which consists of encryptions $\text{Enc}(s_i)$ of each bit $s_i$ of the secret key. Using this, the computation of $\mu = b - \sum_{i=1}^{n} a_i \cdot \text{Enc}(s_i)$ is straightforward, leveraging the linear homomorphisms inherent to TFHE. However, since the $a_i$'s are sampled uniformly at random from $\mathbb{Z}_q$, they may have very large norms, leading to substantial noise growth. TFHE circumvents this issue with a technique known as *gadget decomposition*, which helps mitigate noise growth during multiplications with constants (see [30] for further details on gadget decomposition).

Once the linear part is computed, the server must homomorphically perform the rounding operation, a more challenging task, as follows:

1. Keyswitching(KS): The ciphertexts are keyswitched to a smaller key to accelerate the next steps.
2. ModSwitching(MS): The server switches the modulo of $\mu$ from $q$ to $2N$, producing $\tilde{\mu}$. This transition uses that the order of $X$ in $\mathbb{Z}_{N,q}[X]$ is $2N$.
3. It constructs an *accumulator polynomial* $acc(X)$, whose coefficients encode the outputs of the function evaluated alongside the PBS.

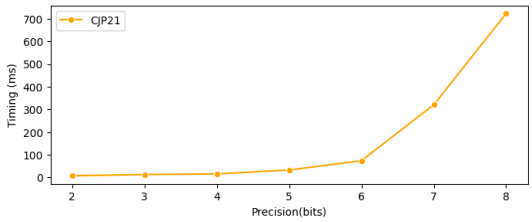



**Fig. 4:** Timing of a PBS with respect to the precision of the plaintext.

4. BlindRotate(BR): The server computes $X^{-\tilde{\mu}} \cdot acc(X)$ which rotates the polynomial and specifically moves its coefficient $v_{\tilde{\mu}} = m$ to the first position. If the LUT is properly encoded in the polynomial's coefficients, the first coefficient now contains an encryption of the LUT output.
5. SampleExtract(SE): The server then extracts this first coefficient into a new LWE ciphertext, which has significantly less noise than the original one. However, if $m > \frac{p}{2}$, the extracted coefficient will acquire an additional negative sign. This phenomenon is known as the *negacyclicity problem*. In Section 3.2, we discuss how we address this issue by using an odd value for $p$.

Figure 3 sums up the bootstrapping procedure, and clarifies the types of ciphertext used at each step. The PBS is more efficient than the bootstrapping processes of other FHE schemes, but it is still by far the most computationally expensive operation in TFHE, and its cost increases significantly with the modulus $p$ of the plaintext. Figure 4 illustrates its execution time on a laptop as a function of the input message's precision (number of bits).

## 3 Constraints for a TFHE-friendly Stream Cipher

### 3.1 State-of-the-Art

While transciphering can theoretically be instantiated with any symmetric cipher, traditional ciphers like AES were soon found to be suboptimal [50], which lead to the design of dedicated ciphers. Early proposals in this direction include the `LowMC` family of block ciphers [2], which minimizes multiplicative depth, and the `Kreyvium` stream cipher [22], a tweak of the well-known eSTREAM finalist `Trivium`. While `Trivium` itself was not originally designed for homomorphic encryption, both `Trivium` and `Kreyvium` showed competitive performance in TFHE-based transciphering scenarios [8].

In 2016, the `FLIP` stream cipher [62] introduced the concept of a filter permutator that randomly permutes key bits and applies a non-linear function on the result to generate a keystream bit. The direct application of non-linear filtering on low-noise key bits helps control the noise generated during homomorphic operations. Two variants were then proposed: `FiLIP` [61] and `Elisabeth` [35], which aimed for stronger security and improved performance. Most notably, `Elisabeth`

operates over arbitrary groups like $\mathbb{Z}_{2^4}$ to minimize costly field conversions in homomorphic evaluations, and it leverages negacyclic lookup tables to eliminate padding bits, thereby optimizing TFHE performance. However, in 2023, an algebraic attack successfully compromised `Elisabeth` [52], prompting the design of patched versions: `Elisabeth-b`, `Gabriel`, and `Margrethe` [58]. While these variants address the identified vulnerabilities, they introduce trade-offs in efficiency: either the evaluation cost or the communication overhead significantly increases compared to the original `Elisabeth`.

Another recent contribution is the construction of a PRF [40] based on the Learning With Rounding (LWR) problem [9]. Computing one PRF output requires only a single PBS in the TFHE context. Still, the main overhead stems from the transmission of the secret key in GGSW format,[1] which is significantly larger than traditional LWE ciphertexts.

Finally, the `FRAST` stream cipher [32] is built on top of a block cipher with a TFHE-friendly round function, and evaluates it efficiently using the double-blind rotation technique combined with WoP-PBS. This allows multiple S-box invocations to be processed at the cost of a single PBS. `FRAST` achieves substantial improvements in throughput while maintaining competitive noise growth. Its main trade-offs lie in a slight increase in communication cost and the need for an initial setup phase to convert GLWE ciphertexts into GGSW form [51].

Concrete performance figures for all the above-mentioned ciphers are provided in Section 6.6.

### 3.2 Constraints from TFHE

*TFHE Operations.* TFHE enables the evaluation of both linear functions and look-up tables on encrypted data, each offering complementary properties.

Linear operations in TFHE are highly efficient but contribute to an increase in ciphertext noise. Specifically, when performing a linear combination of ciphertexts $c_1, \ldots, c_n$ with constant coefficients $\alpha_1, \ldots, \alpha_n$, the noise variance increases in proportion to the squared $\ell_2$-norm of the coefficient vector, i.e., $\sum_{i=1}^{n} \alpha_i^2$. Therefore, to optimize efficiency and control the noise growth, a TFHE-friendly cipher can make greedy use of linear operations while minimizing the norm of the coefficient vectors to limit the resulting noise.

Conversely to linear operations, the programmable Bootstrapping (PBS) is a slow operation, but it allows the computation of any (small-precision) function chosen by the designer while reducing the noise in the ciphertext to a nominal level at the same time. Therefore, while we should minimize the number of these operations for the sake of efficiency, they are essential for introducing non-linearity into the cipher and limiting the noise growth throughout the execution. In practice, within our context, the use of PBS introduces further constraints which we address below.

[1] This technique is a generalization of the one introduced in [51].

*The arrangement of operations.* The PBS produces ciphertexts with a nominal noise level, which is typically lower than that of the input ciphertexts but still significantly higher than the noise in a fresh ciphertext. This implies that if the input bits are fresh encrypted data, they can undergo complex linear operations (specifically with potentially high $\ell_2$-norms). In contrast, the linear functions applied to the outputs of each PBS should involve somewhat limited linear operations in their resulting $\ell_2$-norms in order to limit the noise growth.

*The size of the plaintext space.* The choice of the plaintext space $\mathbb{Z}_p$ has a significant impact on the PBS. Indeed, execution time of the PBS grows exponentially with $p$, which is therefore usually limited to a few bits. Figure 4 illustrates the quick degradation of the PBS performance with respect to the bit-size of $p$. Although some recent works ([54,31,33,59]) introduce more sophisticated techniques for efficiently evaluating larger LUTs, their performance in terms of bits per second remains less favorable compared to using lower precision.

*The parity of the plaintext space.* A common choice for TFHE is using a small power of 2 for the modulus $p$ of the plaintext space, aligning with the format of (small-precision) binary numbers. However, selecting such an even modulus introduces an additional constraint: any function $f : \mathbb{Z}_p \to \mathbb{Z}_p$ used within a PBS must be *negacyclic*, meaning it must satisfy $f(x + p/2) = -f(x)$ for all $x \in \mathbb{Z}_p$. To circumvent this issue, a possible approach consists in keeping a bit of padding to 0, effectively embedding $\mathcal{P} = \mathbb{Z}_p$ into $\mathbb{Z}_{2p}$. However, this padding leads to an important overhead: linear operations are no longer virtually free, as frequent bootstrappings become necessary to maintain the padding bit cleared.

An alternative is to use the *Without Padding PBS* (WoP-PBS) proposed in [31], but the latter is slower than a standard PBS, which is not ideal from a performance standpoint. Another option in the context of TFHE-friendly transciphering is to design a cipher with non-linear functions (S-boxes) that are inherently negacyclic. This strategy, as exemplified in `Elisabeth` [35], imposes strict constraints on the design of the S-boxes, which may introduce weaknesses [52].

Another solution is to adopt an odd modulus $p$, which completely eliminates the negacyclicity problem. This approach, recently suggested in [17], requires only a minor modification of the bootstrapping algorithm, specifically in the construction of the accumulator polynomial, but allows for arbitrary PBS functions $f : \mathbb{Z}_p \to \mathbb{Z}_p$ without the need for a padding bit.

Using an odd modulus may seem unsuitable for manipulating bits or groups of bits. Indeed, it may lead to data expansion, as an element of $\mathbb{Z}_p$ cannot perfectly encode a group of bits. To mitigate this issue, one can select an odd $p$ that is slightly larger but close to $2^\ell$ for some $\ell$, allowing for the efficient embedding of $\ell$-bit chunks into elements of $\mathbb{Z}_p$.

**Our design choices.** We deduce the following guidelines for our design:

1. The plaintext space of the scheme will be reduced to a few bits to take advantage of the relative speed of the PBS at small precision. Specifically,

we chose $p = 17$ which meets our constraints as being odd (no negacyclicity) and the closest to a low power of 2 (thus well suited to encode nibbles of data). Besides, letting $p$ be a prime number eases the design and security analysis thanks to the field structure of $\mathbb{Z}_p = \mathbb{F}_p$.

Moreover, operating in $\mathbb{F}_{17}$ does not constrain the server-side application to this field. Once the server retrieves the homomorphic ciphertexts, they can be efficiently converted to any other space with a bootstrapping. We elaborate more on this point in Section 6.2.

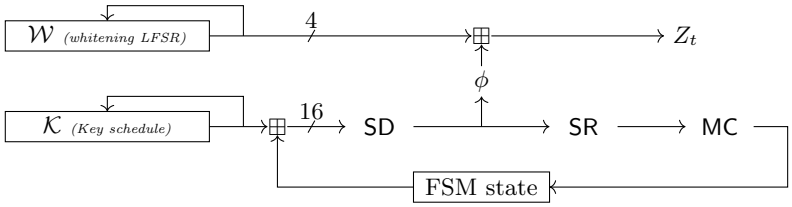
2. The non-linearity comes from a layer of S-boxes, each computing a function $\mathbb{F}_p \to \mathbb{F}_p$ giving rise to one PBS evaluation. Given our fixed choice of $p$, the number of PBS per element of the output stream represents the main performance metric which we search to minimize.
3. The initial key material (stored as fresh TFHE ciphertexts) can go through complex linear combinations before hitting the S-box layer.
4. Each S-box output should only go through lightweight linear operations (i.e., with low $\ell_2$-norms) before undergoing another PBS in order to make the noise in the input of the PBS sufficiently low to ensure correctness.
5. Each S-box output should only go through lightweight linear operations (i.e., with low $\ell_2$-norms) before being released. This way, the TFHE ciphertexts obtained after the stream-cipher decryption keep a noise level as close to nominal as possible.

# 4 Description of `Transistor`

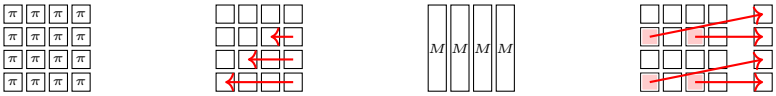
Bringing everything together, we designed the stream cipher `Transistor`. Its overall structure is presented in Section 4.1, its details are explained in Section 4.2, and the influence of noise is discussed in Section 4.4. A reference implementation can be found at https://github.com/CryptoExperts/Transistor/.

## 4.1 Overall Structure

*Usage.* `Transistor` is a stream cipher that generates a keystream consisting of elements from $\mathbb{F}_p = \mathbb{F}_{17}$, referred to as *digits*. It is intended for transciphering, i.e., for the type of protocol we summarized in the introduction(see Figure 1). More precisely, a 128-bit master key and an IV are used to initialize the internal state of `Transistor` using a PRF (namely, `SHAKE` [72]), as suggested in [12]. This initialization is only performed on the client side, and in particular is not evaluated homomorphically, meaning that its cost is negligible. On the other hand, it ensures for example that related IVs cannot be exploited. The entire resulting internal state is then encrypted using TFHE and sent alongside the ciphertext. This ciphertext is obtained by casting the plaintext message to a string of digits of $\mathbb{F}_p$, which is added digit by digit to the keystream produced by `Transistor` using the group law of $\mathbb{F}_p$.




**(a)** General structure (rectangles correspond to registers).




**(b)** SD. **(c)** SR. **(d)** MC. **(e)** $\phi$.

**Fig. 5:** A high level view of `Transistor`.

*Internal State.* The overall structure of `Transistor` is outlined in Figure 5.

The idea is to generate two pseudo-random sequences with a very long period using two distinct LFSRs. One of them generates whitening subkeys, while the other acts as a sort of key schedule. The output of the latter is fed into a Finite State Machine (FSM) with its own state, and which operates on it using non-linear operations. We thus have the following components:

- a register of 16 elements of $\mathbb{F}_p$ (the *FSM state*),
- an LFSR over $\mathbb{F}_p$ (the *key schedule* or *key-LFSR* $\mathcal{K}$) of length $|\mathcal{K}| = 64$,
- an LFSR over $\mathbb{F}_p$ (the *whitening LFSR* $\mathcal{W}$), of length $|\mathcal{W}| = 32$,
- a non-linear round function from $\mathbb{F}_p^{16}$ to itself (the *round function*), and
- a filter $\phi : \mathbb{F}_p^{16} \to \mathbb{F}_p^4$ that extracts 4 digits from the FSM.

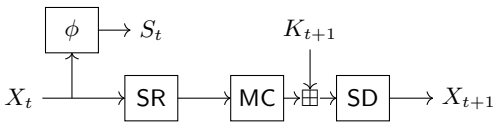
The FSM state is initialized to all zeros, and each LFSR is initialized using digits derived from the 128-bit master key and IV using SHAKE [72].

*Security Claim.* `Transistor` is a stream cipher providing 128 bits of security, meaning any attack should require at least $2^{128}$ elementary operations, assuming no more than $2^{31}$ digits (about 1 GB) are generated with each IV. We allow up to $2^{128}$ digits in total per key, corresponding to the multi-initial-state setting.

### 4.2 Detailed Description

Obviously taking inspiration from the AES, the state of the FSM is organized into a two-dimensional array of size $4 \times 4$, where each entry corresponds to a digit in $\mathbb{F}_p$. With this representation, the successive operations applied to the state can be defined as follows.

**SubDigits** (**SD**) is an S-box layer: the permutation $\pi$ is applied on each digit.
**MixColumns** (**MC**) applies to each column an MDS matrix $M$ over $\mathbb{F}_p$.




**(a)** Notation throughout clocks.

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

**(b)** Numbering in the FSM.

**Fig. 6:** Our notations. Note that the numbering of the digits differs from the one traditionally used for the AES.

**ShiftRows** (**SR**) rotates the $i$-th row by $i$ positions to the left.
**Filter** ($\phi$) takes 4 digits from the state and returns them.

In what follows, we provide a more detailed description of each step, using the notation summarized in Figure 6a. The keystream output at clock $t \geq 0$ consists of a tuple $Z_t \in \mathbb{F}_p^4$, called a block. The internal state of the FSM, just before the filter is applied, is denoted by $X_t$ (so that $S_t = \phi(X_t)$). As a consequence, $X_{t+1} = \mathsf{SD}\left(K_{t+1} + (\mathsf{MC} \circ \mathsf{SR}(X_t))\right)$, where $K_t$ is obtained by concatenating 16 successive digits generated by the key-schedule LFSR $\mathcal{K}$. The FSM is initialized with the all-zero value and its initial state is denoted by $X_{-1} := 0$.

*S-box Layer (SD).* We let $\pi$ be defined by its lookup table:

$$\pi = [1, 12, 6, 11, 14, 3, 15, 5, 10, 9, 13, 16, 7, 8, 0, 2, 4] \tag{1}$$

so that $\pi(0) = 1$, $\pi(1) = 12$, and so on. It has the following polynomial representation, and is thus of maximum degree:

$$\begin{aligned}\pi(x) \ &= 1 + 4x^1 + 13x^2 + 7x^3 + 16x^4 + 15x^5 + 5x^7 + 5x^8 \\ &\quad + 11x^9 + 13x^{10} + 12x^{11} + 13x^{12} + 15x^{14} + x^{15} \ .\end{aligned}$$

It was chosen by enumerating all APN permutations of $\mathbb{F}_{17}$, i.e., all permutations $A$ such that the equation $A(x + a) = A(x) + b$ has at most 2 solutions $x$ for all $a \neq 0$ and all $b$. Then, we selected $\pi$ among those that offer a good balance between minimizing the number of pairs $(a, b)$ for which the previous equation has exactly two solutions, and minimizing the maximum modulus of the Walsh spectrum (see Definition 1).

*Linear Layer (MC).* We opted for a $4 \times 4$ Maximum Distance Separable (MDS) matrix to ensure optimal diffusion. The matrix we chose is

$$M = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & -1 & 1 & -2 \\ 1 & 1 & -2 & -1 \\ 1 & -2 & -1 & 1 \end{bmatrix} \ . \tag{2}$$

We verified that there is no MDS matrix in $\mathbb{F}_{17}$ with coefficients in $\{-1, 1\}$ by exhaustively testing all such matrices. As we were interested in MDS matrices with minimal $\ell_2$-norm and we were able to find during the initial experiments

matrices with a squared $\ell_2$-norm of 7, it was evident from the definition of the $\ell_2$-norm that matrices with minimal $\ell_2$-norm could not have coefficients $x$ with $|x| > 2$. Thus, by testing all matrices with coefficients in $\{-2, -1, 1, 2\}$, we found a total of 30 720 MDS matrices with an $\ell_2$-norm of 7. We selected $M$ for its symmetries, particularly because it is its own transpose.

*Filter.* The filter function $\phi$ maps $\mathbb{F}_p^{16}$ (i.e., the full FSM state) to a tuple $(a, b, c, d)$ in $\mathbb{F}_p^4$. As summarized in Figure 5e, we have that $a, b, c$ and $d$ correspond to the digits of the FSM state with indices 4, 6, 12, and 14 respectively (using the numbering from Figure 6b).

*LFSRs.* The whitening LFSR $\mathcal{W}$ and the key schedule LFSR $\mathcal{K}$ are simply LFSRs over $\mathbb{F}_p$ of maximum period, and have length 32 and 64 respectively. We obtain a maximum-period LFSR over $\mathbb{F}_p^w$ using the coefficients of a primitive polynomial as the taps. More precisely, we used the `SageMath` implementation of the finite field $\mathbb{F}_{p^w}$, which resulted in a pseudo-Conway polynomial. The output of the LFSR is taken from its last cell.

More precisely, an LFSRs of length $\ell$ at time $t$ is a list of digits $x_0^t, ..., x_{\ell-1}^t$ that is clocked as follows:

1. $x_0^{t+1} \leftarrow -\sum_{i=0}^{\ell-1} x_i^t c_i$,
2. $x_i^{t+1} \leftarrow x_{i-1}^t$ for $0 < i < \ell$,
3. the output is $x_{\ell-1}^t$,

where $C = (c_i)_{0 \le i < \ell}$ is the list of its taps, each being a digit of $\mathbb{F}_{17}$. We define $\mathsf{clock}_C$ to be the function applying the operations above to a list $x_0^t, ..., x_{\ell-1}^t$ to update it, and returning $x_{\ell-1}^t$.

For the key schedule $\mathcal{K}$, we use the following taps:

$$\begin{aligned} C(\mathcal{K}) \;=\; \{&9, 4, 6, 4, 8, 6, 6, 16, 3, 9, 15, 12, 8, 12, 11, 4, 4, 8, 1, 8, 8, 9, 4, 6, 6, 7, 6, 3, \\ &16, 14, 14, 6, 10, 15, 14, 13, 10, 1, 1, 10, 13, 11, 14, 10, 7, 4, 15, 8, 16, 3, 13, \\ &14, 15, 16, 3, 16, 9, 3, 6, 12, 15, 9, 12, 3\}\ , \end{aligned}$$

and for the whitening LFSR $\mathcal{W}$ we use

$$\begin{aligned} C(\mathcal{W}) \;=\; \{&8, 14, 14, 14, 1, 6, 12, 10, 14, 14, 14, 5, 2, 5, 6, 13, 6, 15, 14, 3, \\ &13, 16, 1, 13, 9, 1, 7, 15, 13, 6, 14, 3\}\ . \end{aligned}$$

*Master Key Processing.* We generate the digits in $\mathcal{K}$ first, and then those in $\mathcal{W}$. To generate them, we concatenate the 128-bit long master key with an IV and then a byte set to 1. The result is fed into SHAKE128, and the output byte stream of this primitive is used to generate digits of $\mathbb{F}_{17}$ using rejection sampling: if a byte $x$ is equal to 255, we discard it; otherwise, we generate the digit $\lfloor x/15 \rfloor$. Since $15 \times 17 = 255$, this results in an unbiased transformation.

### 4.3 Running `Transistor`

Using `Transistor` to generate a keystream is done as follows.

1. Use the procedure described above to fill the key schedule and whitening LFSRs.
2. Set the FSM state to be all zero.
3. Then, while keystream blocks are needed, we repeat the following:
   (a) Clock the key schedule 16 times and add its content (modulo 17) into the FSM in the order specified in Figure 6b.
   (b) (SubDigits) Apply the S-box $\pi$ (see Equation (1)) to each element of the FSM.
   (c) ($\phi$) Take the elements with indices in $\{4, 6, 12, 14\}$, clock the whitening LFSR four times, and add its successive outputs to these elements. The result forms a keystream tuple of four digits.
   (d) (ShiftRows) Shift the elements of the $i$-th row of the FSM by $i$ positions to the left.
   (e) (MixColumns) Apply the matrix $M$ (see Equation (2)) to each column of the FSM.

### 4.4 Controlling the Noise Evolution

We first detail the implementation of each building block of the scheme using TFHE, as this is essential to justify our design choices and to understand the evolution of the noise throughout the cipher. We then use this discussion to explain how the noise influences the overall security and efficiency of `Transistor`.

*LFSR.* A naive approach for implementing an LFSR homomorphically would be to maintain an encrypted state, and update it by computing a linear combination with the feedback coefficients. However, this method would cause the noise in the state to accumulate over time, necessitating periodic use of PBS operations to refresh and control the noise growth. For this reason we introduce the principle of the *silent LFSR*. Every output of an LFSR is a linear combination of the digits in its initial state. By computing on the fly the coefficients of these linear combinations in clear, we can evaluate the output of the LFSR at every clock cycle without updating an encrypted version of the internal state. This way, the noise variance in the output of the silent LFSR remains stable over time. This principle is comparable to the approach of `FLIP` [62] and follow-up works, whereby a key state is queried without being updated.

To bound the noise variance in the output of the silent LFSR, we consider the worst-case scenario in which all the coefficients in the linear combinations are of maximal absolute value, i.e., $\frac{p-1}{2}$.[2] The resulting noise variance is thus equal to the original noise variance multiplied by the worst-case squared $\ell_2$-norm.

[2] Constant coefficients of $\mathbb{F}_p$ are encoded as integers of the interval $[-\frac{p-1}{2}, \frac{p-1}{2}]$ to minimize their absolute value and hence their impact on the noise.

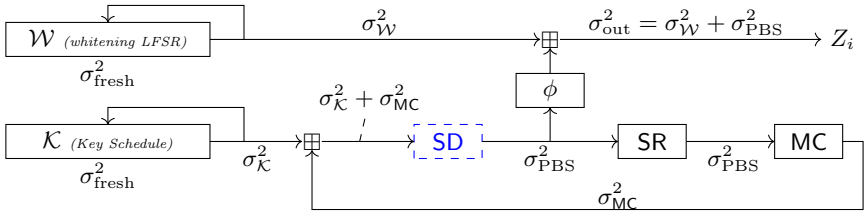



**Fig. 7:** Evolution of the noise variance in a homomorphic evaluation of `Transistor`. Operations involving PBSs are in blue and dashed.

Specifically, in the output of the key schedule LFSR $\mathcal{K}$ and of the whitening LFSR $\mathcal{W}$, the noise variances $\sigma^2_{\mathcal{K}}$ and $\sigma^2_{\mathcal{W}}$ satisfy

$$\sigma^2_{\mathcal{K}} \leq |\mathcal{K}| \cdot \left(\frac{p-1}{2}\right)^2 \cdot \sigma^2_{\text{fresh}} \quad \text{and} \quad \sigma^2_{\mathcal{W}} \leq |\mathcal{W}| \cdot \left(\frac{p-1}{2}\right)^2 \cdot \sigma^2_{\text{fresh}}, \tag{3}$$

where $\sigma^2_{\text{fresh}}$ is the noise variance of the encrypted key material in the LFSRs.

*SubDigits.* Each digit of the state of the FSM goes through a PBS that evaluates the permutation $\pi$. All PBSs can be evaluated in parallel for higher speed. We denote by $\sigma^2_{\text{PBS}}$ the noise variance at the output of SubDigits for encrypted digits.

*ShiftRows.* Since each digit in the state is encrypted in a separate ciphertext digit, this step involves simply rearranging the ciphertext digits within the state. Consequently, it incurs no additional noise growth and no performance impact.

*MixColumns.* This operation involves a straightforward linear combination of the digits of the state. The matrix $M$ has been specifically constructed to minimize the $\ell_2$-norm, considering coefficients in $\mathbb{F}_{17}$. From the homomorphic perspective, this choice is crucial, as the variance of the noise increases proportionally with the square of this $\ell_2$-norm, that we denote $L_{\text{MC}}$. Namely, the noise variance $\sigma^2_{\text{MC}}$ after MixColumns satisfies $\sigma^2_{\text{MC}} = L^2_{\text{MC}} \cdot \sigma^2_{\text{PBS}}$.

*Sums.* The output of MixColumns is then added to the next output of the LFSR to be injected again into SubDigits. The noise variance after the addition step corresponds to the sum of both noise variances. Similarly, the noise variance at the output of the scheme, referred to as $\sigma^2_{\text{out}}$, is equal to the sum $\sigma^2_{\mathcal{W}} + \sigma^2_{\text{PBS}}$.

Figure 7 illustrates the evolution of the noise variance throughout the operations of `Transistor`. Building on the previous equations, the main constraints influencing the design of `Transistor` are related to the noise variance at both the input of the PBS and the output of the scheme. Specifically, the noise variance $\sigma^2_{\mathcal{K}} + \sigma^2_{\text{MC}}$ at the input of the PBS (i.e., at input of SubDigits) must remain sufficiently low, otherwise it could lead to a high probability of PBS failure. Additionally, the noise variance $\sigma^2_{\text{out}} = \sigma^2_{\mathcal{W}} + \sigma^2_{\text{PBS}}$ at the output stream must remain low enough for subsequent applications, ideally as close as possible to

the nominal noise variance at the PBS output $\sigma^2_{\text{PBS}}$. In practical settings, we have $\sigma^2_{\text{PBS}} \gg \sigma^2_{\text{fresh}}$, the noise variances from both LFSRs are negligible compared to $\sigma^2_{\text{PBS}}$. For example in our implementation, the noise magnitude of $\sigma_{\text{fresh}}$ is around $2^{14}$, while the noise magnitude of $\sigma_{\text{PBS}}$ is around $2^{52}$. Consequently, $\sigma^2_{\text{out}} \approx \sigma^2_{\text{PBS}}$ which validates the second constraint. Similarly, the noise variance at the input of the PBS is close to that at the output of MixColumns. The latter additionally remains low due to the minimized $\ell_2$-norm of the coefficients of the MDS matrix $M$, thereby validating the first constraint.

To wrap up, the design of `Transistor` allows to control the evolution of the noise in the FSM while getting a very low number of PBS per element. To complete our noise analysis, we need to set the parameters of the TFHE scheme to ensure the correctness of the PBS. Concretely, the noise $\sigma^2_{\mathcal{K}} + \sigma^2_{\mathsf{MC}}$ at the input of SubDigits should be low enough to fail with a negligible probability. Of course, these parameters must ensure that the PBS operates as fast as possible while maintaining the security of the scheme. In Section 6.4, we detail our method for selecting the parameters.

## 5 Security Analysis

In this section we analyze the resistance of `Transistor` against classical attacks and derive lower bounds on the complexity required for these attacks to succeed. The internal parameters, especially the LFSR dimensions, were chosen based on the security analysis results presented here. Many attacks discussed in this section assume that the adversary has access to the sequence $(S_t)_{t\geq 0}$ *prior to its addition with the whitening LFSR*. The corresponding attacks against `Transistor` have therefore a higher complexity since they need to be adapted in order to remove the influence of the whitening LFSR, either by guessing part of its internal state (which increases time complexity), or by cancelling its outputs thanks to a parity-check equation (which increases data complexity).

We first describe Time-Memory-Data trade-offs and Guess-and-Determine in Section 5.1. Then, we develop our arguments against correlation-based key recovery attacks and the existence of distinguishers in two steps. First, we establish an important property of `Transistor`: three consecutive outputs are statistically independent from the content of the LFSRs (Section 5.2). We then analyze the correlation of linear trails, and thus argue that (fast) correlation attacks are not a threat for a properly used instance of `Transistor` (Section 5.4). In Section 5.5, we argue that a multi-key approach is similarly inefficient.

Several attack directions are discussed in the appendix, namely algebraic attacks (Appendix D.1), and the relationship between the security of `Transistor` and `LEX` [15] (Appendix D.2).

### 5.1 Dimensioning the Internal State based on Generic Attacks

**Time-Memory-Data Trade-Offs.** Let $P, M, T, D$ denote the respective precomputation, memory, time and data complexities needed for recovering the internal state of a stream cipher. As independently introduced by Babbage [5] and

Golic [53], generic Time-Memory-Data Trade-Offs aim at leveraging a more interesting balance between the four metrics than the extreme cases obtained with an exhaustive search ($T = N$, $D = 1$) or a full code-book attack ($P = M = N$, $T = 1$), where $N$ is the number of possible internal states.

To do so, a table is first built and stored offline. This table contains pairs $(X, F(X))$ (indexed by the second coordinate) where $F$ is the function which maps an initial state $X$ to the first $n$ elements of the output sequence where $n$ is chosen such that $\mathrm{Im}(F)$ has size $N$. Then, during the online phase, the attacker hopes to find a collision between the images stored in the table and the ones observed online. If the attacker observes $D$ sequences of length $n$, the standard birthday-paradox argument states that $MD \approx N$ is the condition for such a collision to occur. Taking $M = D = P = T = \sqrt{N}$ gives the classical trade-off.

In the case of `Transistor` without the whitening LFSR $\mathcal{W}$, such an attack can be mounted in two ways. First, we can choose $F$ as the function which maps the LFSR state $\mathcal{K}$ *and* the state of the FSM to the first outputs of $\phi$. In this case, $n = |\mathcal{K}| + 16$, providing the first bound:

$$p^{|\mathcal{K}|+16} \geq 2^{\lambda} D \ ,$$

where $\lambda$ is the security level and $p = 17$ for `Transistor`. In that case, the number of observed sequences $D$ can actually be replaced by the number of observed successive output digits. Indeed, by observing $d \gg n$ output digits, one can build $d - n + 1 \approx d$ sequences of $n$ digits.

Yet, at any clock $t > 0$, the FSM state only depends on the initial state $\mathcal{K}$. We can therefore consider $F$ as the function which maps the initial LFSR state $\mathcal{K}$ (and the all-zero FSM state) to the first $k/4$ output blocks $S_0, S_1 \dots S_{k/4-1}$, where $k$ denotes the size of $\mathcal{K}$ in digits. In this case, $n = |\mathcal{K}|$ and therefore:

$$p^{|\mathcal{K}|} \geq 2^{\lambda} D \ .$$

However, this attack must be launched in the multi-key setting because each observed output sequence must be obtained from an all-zero initial FSM state. Then, $D$ is also the number of distinct keyed primitives attacked.

Both attacks can be applied to the full `Transistor`, that is, with the whitening LFSR $\mathcal{W}$. To do so, we denote by $P_{\mathcal{W}} = X^{|\mathcal{W}|} - \sum_{i=1}^{|\mathcal{W}|} c_i X^{|\mathcal{W}|-i}$ the characteristic polynomial of $\mathcal{W}$, and by $(w_t)_{t\in\mathbb{N}}, (s_t)_{t\in\mathbb{N}}, (z_t)_{t\in\mathbb{N}}$ the sequences of digits generated by $\mathcal{W}, \phi$ and `Transistor` respectively, so that $z_t = w_t + s_t$. Therefore, by linearity, we immediately deduce that,

$$\forall t \geq 0, \quad z_{|\mathcal{W}|+t} - \sum_{i=1}^{|\mathcal{W}|} c_i z_{|\mathcal{W}|-i+t} \ = \ s_{|\mathcal{W}|+t} - \sum_{i=1}^{|\mathcal{W}|} c_i s_{|\mathcal{W}|-i+t}.$$

The same attack as before can therefore be mounted by observing the sequence $(s_{|\mathcal{W}|+t} - \sum_{i=1}^{|\mathcal{W}|} c_i s_{|\mathcal{W}|-i+t})_{t\in\mathbb{N}}$, instead of $(s_t)_{t\in\mathbb{N}}$. Therefore, with the parameters used in `Transistor`, the length of the keystream generated from the same key is limited to $2^{31}$ digits. As a result, TMDTO attacks have a time complexity of $2^{296}$ in the single-IV setting, which drops to $2^{130}$ when keystreams generated from $2^{130}$ IVs are available to the attacker.

**Guess and Determine.** We explain a basic Guess-and-Determine attack, where the attacker links the FSM state $X_t$ (initialized as $X_{-1} = 0$) to the filter output $S_t$ by guessing digits of the key-schedule sequence $K_t$.

Without the whitening LFSR, the attacker observes at each clock $t \geq 0$

$$S_t = \phi(X_t) = \mathsf{SD}\big(K_t + (\mathsf{MC} \circ \mathsf{SR}(X_{t-1}))\big)[4, 6, 12, 14] \ .$$

If $X_{t-1}$ is known, he deduces

$$K_t[4, 6, 12, 14] = \mathsf{SD}^{-1}(S_t) - (\mathsf{MC} \circ \mathsf{SR}(X_{t-1}))\,[4, 6, 12, 14] \ .$$

After guessing the 12 missing digits $K_t[0, 1, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15]$, he computes the full $X_t = \mathsf{SD}\,(K_t + (\mathsf{MC} \circ \mathsf{SR}(X_{t-1})))$.

Starting from clock $t = |\mathcal{K}|/16$, the key-schedule digits are linearly dependent on the previous ones, and the attacker can verify that the output $S_t$ is correct without making new guesses. Therefore, in total he has to guess $\frac{12}{16}|\mathcal{K}|$ digits, leading to a complexity $p^{\frac{3}{4}|\mathcal{K}|} \approx 2^{196}$. When taking into account the whitening LFSR, the attacker first has to guess it, leading to an attack with complexity $p^{\frac{3}{4}|\mathcal{K}|+|\mathcal{W}|} \approx 2^{294}$.

### 5.2 Three consecutive outputs are statistically independent of the secret key

The basic strategy in (fast) correlation attacks against stream ciphers [74,64] consists in recovering some information about (a part of) the initial state of the cipher from the knowledge of the keystream. In the following, we investigate this type of attacks *without the whitening LFSR* and aim at recovering the internal state of the key-LFSR $\mathcal{K}$. To this end, we consider the so-called *augmented function* with $n$ outputs, which generates $n$ consecutive output blocks of $(S_t)_{t\in\mathbb{N}}$ from the internal state of the FSM and from $(n-1)$ consecutive 16-digit blocks of the key-sequence:

$$\begin{aligned} F^{(n)} : \quad \mathbb{F}_p^{16} \times \mathbb{F}_p^{16(n-1)} \quad &\to \mathbb{F}_p^{4n} \\ (X, K_1, \ldots, K_{n-1}) &\mapsto (S_0, S_1, \ldots, S_{n-1}) \ . \end{aligned} \tag{4}$$

We here focus on the case where $n \leq 5$, otherwise $F^{(n)}$ depends on 64 elements $K_1, \ldots, K_{64}$ only since all $K_t$ for $t > 64$ are derived from these elements. In this case ($n \leq 5$), it is obvious that $F^{(n)}$ is balanced, *i.e.*, all preimages by $F^{(n)}$ have size $p^{12n}$. In this context, an important quantity is the smallest length of output sequence $(S_t)_{t\in\mathbb{N}}$ that can provide information on the sequence produced by the key-LFSR [3].

In the following, we show that this minimal length is at least 4. In other words, we prove that the output of $F^{(3)}$ is statistically independent from (i.e., not correlated to) the key-sequence. This property is equivalent to the following theorem.

**Theorem 1.** *For any $K_1, K_2 \in \mathbb{F}_p^{16}$, the function $X_0 \mapsto F^{(3)}(X_0, K_1, K_2)$ is balanced.*

*Proof.* We consider an arbitrary image $(S_0, S_1, S_2) \in \mathbb{F}_p^{4\times 3}$, and we show that there are exactly $p^4$ solutions $X_0$ to the equation

$$F^{(3)}(X_0, K_1, K_2) = (S_0, S_1, S_2). \tag{5}$$

Instead of directly solving for the initial state $X_0$, we write the output in terms of the state $X_1$ after one round. Using the numbering of the digits as in Fig 6b, we obtain:

$$\begin{aligned}
\phi(X_0) = S_0 &\iff \phi\left(\mathsf{SR}^{-1} \circ \mathsf{MC}^{-1}\Big(\big(\mathsf{SD}^{-1}(X_1) - K_1\big)\Big)\right) = S_0 \\
&\iff \begin{cases} M^{-1} \times \begin{bmatrix} \pi^{-1}(X_1[1]) - K_1[1] \\ \pi^{-1}(X_1[5]) - K_1[5] \\ \pi^{-1}(X_1[9]) - K_1[9] \\ \pi^{-1}(X_1[13]) - K_1[13] \end{bmatrix} = \begin{bmatrix} * \\ S_0[1] \\ * \\ S_0[2] \end{bmatrix} \\ M^{-1} \times \begin{bmatrix} \pi^{-1}(X_1[3]) - K_1[3] \\ \pi^{-1}(X_1[7]) - K_1[7] \\ \pi^{-1}(X_1[11]) - K_1[11] \\ \pi^{-1}(X_1[15]) - K_1[15] \end{bmatrix} = \begin{bmatrix} * \\ S_0[0] \\ * \\ S_0[3] \end{bmatrix} \end{cases} \qquad (6) \\
\phi(X_1) = S_1 &\iff X_1[4, 6, 12, 14] = S_1[0, 1, 2, 3] \qquad (7) \\
\phi(X_2) = S_2 &\iff \phi\Big(\mathsf{SD}\big(K_2 + (\mathsf{MC} \circ \mathsf{SR}(X_1))\big)\Big) = S_2 \\
&\iff \begin{cases} M \times \begin{bmatrix} X_1[0] \\ X_1[5] \\ X_1[10] \\ X_1[15] \end{bmatrix} = \begin{bmatrix} * \\ \pi^{-1}(S_2[0]) - K_2[4] \\ * \\ \pi^{-1}(S_2[2]) - K_2[12] \end{bmatrix} \\ M \times \begin{bmatrix} X_1[2] \\ X_1[7] \\ X_1[8] \\ X_1[13] \end{bmatrix} = \begin{bmatrix} * \\ \pi^{-1}(S_2[1]) - K_2[6] \\ * \\ \pi^{-1}(S_2[3]) - K_2[14] \end{bmatrix} \end{cases} \qquad (8)
\end{aligned}$$

We solve the system with the following steps:

1. Pick arbitrary values for $X_1[0, 2, 5, 7]$.
2. Set $X_1[4, 6, 12, 14] = S_1[0, 1, 2, 3]$ following (7).
3. Deduce $X_1[8, 10, 13, 15]$ from (8) using linear algebra.
4. Deduce $X_1[1, 3, 9, 11]$ from (6) using linear algebra.

An important observation is that there is a single solution to the linear systems being solved at steps 3 and 4, because $M$ is an MDS matrix (see Property 1 below). Therefore there are exactly $p^4$ solutions to Equation (5). □

*Property 1.* [60, Page 319] Let $M$ be an MDS matrix of size $n \times n$, and $X$ and $Y$ vectors of length $n$. If we fix $n - t$ coordinates of $X$, and $t$ coordinates of $Y$, then the equation $M \times X = Y$ has a unique solution.

The fact that the knowledge of three consecutive keystream blocks does not provide any information about the content of the key-LFSR is an interesting

security argument. As a comparison, the number of consecutive outputs of the keystream generator for which there exists a biased linear relation with the content of the LFSR is two in SNOW 2.0 [77], and three in SNOW-V [73]. It is also four in Rocca [71, Section 4.5] but this property has been derived from the automatic search method introduced in [45]. Contrary to these other ciphers, which are also based on the AES round function and use an MDS matrix, the structure of Transistor enables us to derive this argument in a very simple way from the MDS property of MixColumns.

### 5.3 Biased Linear Relations Involving Four Consecutive Outputs

We want to estimate the minimal data complexity required to recover the internal state of the key-register from the knowledge of the output sequence $(S_t)_{t\in\mathbb{N}}$, given that at least four consecutive outputs $(S_t, S_{t+1}, S_{t+2}, S_{t+3})$ need to be considered together. The so-called Xiao-Massey lemma [78,21] shows that, if the output of the augmented function $F^{(n)}$ is correlated to its key-input, then there exists a biased linear relation between the key-inputs and the outputs of $F^{(n)}$. It follows that, as soon as the key-LFSR and the considered segment of the output sequence are not statistically independent, there exists a biased linear relation between the digits of these two sequences. In other words, there exists a pair of masks $\alpha \in \mathbb{F}_p^{16(n-1)}$ and $\beta \in \mathbb{F}_p^{4n}$, $\beta \neq 0$, such that the output distribution of

$$\begin{aligned} F^{(n)}_{\alpha,\beta} : \quad & \mathbb{F}_p^{16} \times \mathbb{F}_p^{16(n-1)} \to \mathbb{F}_p \\ & (X, K_1, \ldots, K_{n-1}) \mapsto \beta \cdot F^{(n)}(X, K_1, \ldots, K_{n-1}) - \textstyle\sum_{i=1}^{n-1} \alpha_i K_i \end{aligned}$$

differs from the uniform distribution. Unlike in the block-cipher scenario, the attacker has no control over the input $X$, thereby requiring the associated input mask to be zero in the linear relations we consider. Obviously, if $F^{(n)}_{\alpha,\beta}$ is biased, it also holds for the entire span of $(\alpha_1, \ldots, \alpha_{n-1}, \beta)$, i.e., for all $(\lambda\alpha_1, \ldots, \lambda\alpha_{n-1}, \lambda\beta)$ with $\lambda \in \mathbb{F}_p^*$.

There might exist some other relations between these two sequences, of higher degree, whose probability distribution is farther from the uniform distribution. However, it seems much more difficult to exploit nonlinear relations in an attack for two reasons: in practice, what is known to the attacker is the sum between $(S_t)_{t\in\mathbb{N}}$ and the output of the whitening LFSR. Any relation involving the digits of $(S_t)_{t\in\mathbb{N}}$ in a nonlinear manner would involve the outputs of the whitening LFSR in a nonlinear manner, and would then probably require an exhaustive search for its initial state. A second motivation for focusing on linear relations is that recovering the internal state of the key-LFSR faster than exhaustive search is much easier if the known biased relations are linear.

In the following, we use the generalization of linear cryptanalysis to $\mathbb{F}_p$, as described in [7,14]. An important tool is the Walsh transform over $\mathbb{F}_p$.

**Definition 1 (Walsh transform over $\mathbb{F}_p$).** *Let $F : \mathbb{F}_p^n \to \mathbb{F}_p^k$ and $\omega$ be a $p$-th root of unity in $\mathbb{C}$. The Walsh transform of $F$ is the function $\widehat{F} : \mathbb{F}_p^n \times \mathbb{F}_p^k \to \mathbb{C}$*

*defined for any* $a \in \mathbb{F}_p^n$ *and* $b \in \mathbb{F}_p^k$ *by:*

$$\widehat{F}(a,b) = \sum_{x \in \mathbb{F}_p^n} \omega^{b \cdot F(x) - a \cdot x} \ .$$

It follows that the correlation of a linear trail $(\alpha, \beta)$ with $\alpha \in \mathbb{F}_p^{16(n-1)}$ and $\beta \in \mathbb{F}_p^{4n}$ over $n$ rounds is

$$C^{(n)}(\alpha,\beta) = p^{-16n} \widehat{F}^{(n)}(0,\alpha,\beta) \ .$$

Each pair $(\alpha, \beta)$ defines a linear trail where all intermediate masks are fixed. The squared modulus of its correlation is then upper-bounded by the product of the squared moduli of the normalized Walsh coefficients of all active S-boxes in the trail, i.e.,

$$\left|C^{(n)}(\alpha,\beta)\right|^2 \leq \left(\frac{\mathcal{L}(\pi)}{p}\right)^{2w_n} \tag{9}$$

where $\mathcal{L}(\pi)$ is the maximal modulus of the Walsh coefficients of $\pi$, and $w_n = \sum_{i=1}^{n-1} wt(\alpha_i)$ where $wt(\cdot)$ denotes the number of nonzero digits of a vector with coordinates in $\mathbb{F}_p$. The proof is given in Appendix A.1 for the sake of completeness.

As detailed in Appendix A.2 , we have established a lower bound on the minimal number of active S-boxes over $n$ rounds, $w_n$, using well-known MILP-based techniques. We found that $w_4 \geq 13$, $w_5 \geq 20$, $w_6 \geq 25$, and $w_n \geq 26$ for $n \geq 7$, with the truncated trail examples given on Figure 11. These bounds might not be tight as we have not tried to instantiate these paths, but a lower bound is sufficient. The S-box has been chosen to minimize the maximal modulus of its Fourier coefficients, which is $\mathcal{L}(\pi) = 6.5135 \approx 2^{2.70}$. We then deduce the following bounds on the correlations of 4-round linear trails.

**Theorem 2.** *The correlation of any* 4*-round linear trail of* `Transistor` *satisfies*

$$\left|C^{(4)}(\alpha,\beta)\right|^2 \leq 2^{-35.98} \ ,$$

*where these bounds may*[3] *only be tight for some linear trails with* 13 *active S-boxes. Moreover, there are at most* $2^8$ *different* 4*-round trails of* `Transistor` *that share the same output mask* $\beta$ *and achieve this bound.*

*Proof.* The first statement is a direct consequence of (9), combined with the values $w_4$ (obtained by MILP) and $\mathcal{L}(\pi)$ (computed directly from the look-up table). The last statement comes from the fact that there exists exactly four 4-round truncated trails with 13 active S-boxes (obtained using MILP). These truncated trails are depicted on Figure 11 in Appendix A.2 and correspond to the four possibilities for the activity pattern of $(\beta_2, \beta_3)$. Each output mask then corresponds to a single truncated trail, for example the one depicted below.

[3] The bound may not be tight as we have not tried to instantiate these trails.

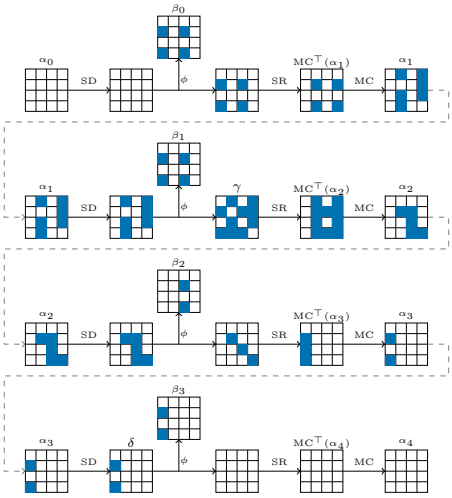



**Fig. 8:** One of the four activity patterns for 4-round trails.

We then have to determine the maximum number of possible values for $(\alpha_1, \alpha_2, \alpha_3)$ for a fixed output mask $\beta$ that follow this activity pattern. The mask $\delta$ at the output of SD at Round 4 is equal to $\beta_3$. The corresponding input mask $\alpha_3$ has to be such that $\mathsf{MixColumns}^T(\alpha_3)$ has the required activity pattern. Since $\alpha_3$ has a single active column $\alpha_{3,0}$, and that 3 digits are fixed in $(\alpha_{3,0}, M^T(\alpha_{3,0}))$, we deduce that $\alpha_3$ can take only $(p-1)$ possible values because $M$ is MDS (see Property 1). Similarly, $\alpha_2$ must be such that $\mathsf{MixColumns}^T(\alpha_2)$ has the required activity pattern. Moreover, the digits at positions $4, 6, 12, 14$ of the mask $\gamma$ in Figure 8 are equal to the digits of $\beta_1$ at the same positions. Therefore, each of the three active columns of $\mathsf{MixColumns}^T(\alpha_2)$ has two digits that are fixed. Thus, only the third column of $\alpha_2$ is not fixed, and can take $(p-1)$ different values. Since the value of $\alpha_1$ is entirely determined by $\beta_0$, it follows that the number of possible masks $\alpha$ following this activity pattern is at most $(p-1)^2$. □

### 5.4 Initial-State Recovery based on (Fast) Correlation Attacks

(Fast) correlation attacks [64,23,27,76] can be seen as linear attacks in the particular setting of stream ciphers. The minimal length $N$ of keystream required for recovering the initial state of the underlying LFSR (of length $|\mathcal{K}|$), using that the output distribution of $F^{(n)}_{\alpha,\beta}$ is biased, is determined by its *capacity* [16], or *squared imbalance* [6], $\Delta(F^{(n)}_{\alpha,\beta})$ defined in the next proposition:

$$N = \frac{|\mathcal{K}| \ln p}{\Delta(F^{(n)}_{\alpha,\beta})} \,. \tag{10}$$

The proof of this result is provided in Appendix A.3 in the more general case where the relation is of the form $-g(K_t, \ldots, K_{t+n-1}) + h(S_t, \ldots, S_{t+n-1})$ for any (possibly nonlinear) functions $g$ and $h$.

**Definition 2 (Capacity of the output distribution of a function [6,57]).** *Let $F$ be a function from $\mathbb{F}_p^n$ to $\mathbb{F}_p$ with output distribution*

$$\Pr_{X \xleftarrow{\$} \mathbb{F}_p^n}[F(X) = y] = \frac{1}{p} + \varepsilon_y \text{ with } |\varepsilon_y| \ll 1, \text{ for all } y \in \mathbb{F}_p \, .$$

*The* capacity *(aka* squared imbalance*) of the output distribution of $F$ is*

$$\Delta(F) := p \sum_{y \in \mathbb{F}_p} \varepsilon_y^2 \, .$$

For an attack exploiting the output distributions of $F_{\alpha,\beta}^{(n)}$ for several $(\alpha, \beta)$ together, the relevant parameter is the sum of all capacities of the output distributions of $F_{\alpha,\beta}^{(n)}$ with $(\alpha, \beta)$ in the span of the considered relations [16,57, Def. 2].

The following result is a consequence of Plancherel's formula and has been generalized to any finite Abelian group in [7] (see Appendix A.3 for a proof).

**Proposition 1 ([7]).** *Let $F$ be a function from $\mathbb{F}_p^n$ to $\mathbb{F}_p$. Then, the capacity of its output distribution is*

$$\Delta(F) = p^{-2n} \sum_{b \in \mathbb{F}_p^*} \left| \widehat{F}(0, b) \right|^2 \, .$$

It follows that the capacity $\Delta(F_{\alpha,\beta}^{(n)})$ is nothing else than the sum of the squared moduli of the correlations of all trails corresponding to the masks $(\lambda\alpha, \lambda\beta)$, $\lambda \in \mathbb{F}_p^*$, i.e.,

$$\Delta(F_{\alpha,\beta}^{(n)}) = \sum_{\lambda \in \mathbb{F}_p^*} \left| C^{(n)}(\lambda\alpha, \lambda\beta) \right|^2 \, .$$

We deduce from Theorem 2 that any correlation attack based on the span of a single linear trail $(\alpha, \beta)$ requires at least $64 \times \ln(17) \times 2^{31.97} = 2^{39.5}$ 4-digit blocks of keystream, i.e., $2^{41.5}$ digits of the output sequence. It is worth noting that this bound is tight when the initial state of the key-LFSR is recovered by performing an exhaustive search over all its possible values by a maximum-likelihood statistical test. The corresponding time complexity then exceeds the cost defined by the target security level. Other algorithms can be used, for instance algorithms based on low-weight parity-check relations, but the price to pay is a significant increase of the required length $N$ of output sequence [64,23,27].

More interestingly, it has been shown by Todo *et al.* [76] that the use of multiple linear approximations with the same output mask may significantly improve the attack. The minimal keystream length required by this attack is again derived from Equation (10) where the capacity of the only output distribution that is considered, *i.e.* $\Delta(F_{\alpha,\beta}^{(n)})$, is replaced by the capacity of the *set* of all considered

linear approximations. Based on Theorem 2, it can be proved that an attack using all optimal linear trails together requires at least $64 \times \ln(17) \times 2^{23.97} = 2^{31.5}$ blocks of keystream ($2^{33.5}$ digits). Here, we assume that the use of additional linear relations with a lower capacity does not improve the attack. It is important to note that this is an information-theoretic bound and the attack described in [76] requires a longer keystream segment. Moreover, the previous analysis assumes that the output of the whitening LFSR is known to the attacker. This can be achieved by an exhaustive search for its initial state, implying that the time complexity of our attack will be multiplied by a factor $p^{|\mathcal{W}|} \simeq 2^{131}$. However, we can also get rid of the whitening LFSR by choosing some coefficients $\beta_0, \ldots, \beta_{n-1}$ corresponding to a recurring relation satisfied by the sequence generated by $\mathcal{W}$, such that $\sum_{i=0}^{n-1} \beta_i \cdot Z_{t+i} = \sum_{i=0}^{n-1} \beta_i \cdot S_{t+i}$. By definition, such a recurring relation corresponds to a multiple of the feedback polynomial of $\mathcal{W}$. It has therefore degree $d \geq 32$, implying that it involves $d$ digits and therefore 4-digit blocks of $(S_t)_{t\in\mathbb{N}}$ at distance $\lceil d/4 \rceil \geq 8$. As mentioned, we have $w_n \geq 26$ for $n \geq 7$, which implies that exploiting any linear approximation compatible with a recurring relation for the whitening LFSR requires the knowledge of at least $64 \times \ln(17) \times 2^{67} \approx 2^{75}$ blocks (where $2^{-67}$ is the capacity upper bound deduced from (9) for $w_n = 26$), i.e. at least $2^{77}$ digits of the output sequence.

### 5.5 Linear Distinguishers on the Keystream

Linear approximations can also lead to linear distinguishing attacks [55] (aka linear masking attacks [77]). These attacks do not recover the initial state of the key-register, but the counterpart is that they can use together several keystream segments produced from multiple initial states. They consist in exhibiting a biased linear relation among the keystream digits of the form

$$\sum_{i\in\mathbb{T}_z} \beta_i \cdot Z_{t+i} \tag{11}$$

where $\mathbb{T}_z$ defines a set of positions in the keystream. Obviously, the maximal value in $\mathbb{T}_z$ must be smaller than the maximal keystream length that can be generated from the same key/IV pair, i.e., $2^{31}$ digits. Such a relation is typically derived from a *parity-check equation* for the key-LFSR, i.e., $\sum_{i=0}^{\ell-1} \gamma_i K_{t+i} = 0, \forall t \geq 0$ . Any multiple of the LFSR feedback polynomial equivalently defines a parity-check equation, but the capacity of the resulting linear approximation of the keystream highly increases with the number $w$ of monomials in this multiple. Here, the feedback polynomial of the key-LFSR is a primitive polynomial of degree $|\mathcal{K}| = 64$ over $\mathbb{F}_p$. Since this polynomial is dense and does not have any specific property, the number of its multiples of weight $w$, of degree at most $d$ and constant coefficient equal to 1 can be approximated by

$$\binom{d}{w-1}(p-1)^{w-1}p^{-|\mathcal{K}|} .$$

Thus, the expected degree such that there exists a multiple of weight $w$ is

$$[(w-1)!]^{\frac{1}{w-1}}\, p^{\frac{|\mathcal{K}|}{w-1}-1} .$$

This degree is then smaller than $2^{31}$ only for $w \geq 9$. Using such a parity-check equation, a biased linear relation on the keystream as (11) will be the result of the combination of at least 9 linear approximations like the ones studied in Theorem 2. Estimating the resulting capacity is difficult (see e.g. [68]) but we can safely assume that it is much lower than $\max_{\alpha,\beta}(\Delta(F_{\alpha,\beta}))^4 \simeq 2^{-128}$. Therefore, the corresponding attack would be more expensive than an exhaustive search for the key. It is worth noting that the previous analysis does not take into account the whitening LFSR. For an attack against the full `Transistor`, we have to use a parity-check equation that holds for the key-LFSR and for the whitening LFSR simultaneously, i.e., a multiple of the lcm of the two feedback polynomials $P_{\mathcal{K}}$ and $P_{\mathcal{W}}$. However, the expected degree for such a multiple cannot be estimated by replacing $|\mathcal{K}|$ by $|\mathcal{K}| + |\mathcal{W}|$ in the previous formula, because $P_{\mathcal{W}}$ defines a subfield of the field defined by $P_{\mathcal{K}}$.

## 6 Performances of Transciphering with `Transistor`

This section focuses on the performances of transciphering with `Transistor`. We first address the wrapping of a (`Transistor`) symmetric key as a compact set of TFHE ciphertexts for which we additionally introduce a trade-off between bandwidth and computation. Next, we explain how to manage different data representations to be able to fit with the input format of the server application. We then provide a detailed description of the homomorphic evaluation of `Transistor`. We finally give some implementation benchmarks and comparison to the state of the art.

### 6.1 Key Wrapping and Bandwidth in TFHE Transciphering

Assume one wants to generate a fresh TFHE ciphertext vector $(c_1, \ldots, c_t)$ for a plaintext vector $(m_1, \ldots, m_t) \in \mathbb{Z}_p^t$, where $c_i = (a_{i,1}, \ldots, a_{i,n}, b_i)$, for every $i \in [1, t]$. Since the $a_{i,j}$'s are uniformly sampled at random over $\mathbb{Z}_q$, a folklore trick is to generate them pseudorandomly from a seed. We get the following compressed encryption procedure (where $\lambda$ denotes the security level in bits):

`CompressEncrypt`$(s, m_1, \ldots, m_t)$

1. Sample $\mathsf{seed} \leftarrow \{0,1\}^\lambda$
2. Expand $((a_{i,j})_{1 \leq j \leq n})_{1 \leq i \leq t} \leftarrow \mathsf{PRG}(\mathsf{seed})$
3. $\forall\, i \in [1, t]$: $b_i \leftarrow \sum_{j=1}^{n} a_{i,j} \cdot s_j + \tilde{m}_i + e_i$ with $e_i \leftarrow \chi_\sigma$
4. Return $(\mathsf{seed}, b_1, \ldots, b_t)$

Recovering standard TFHE ciphertexts $c_1, \ldots, c_t$ from the compressed form $(\mathsf{seed}, b_1, \ldots, b_t)$ is simply done by expanding the $a_{i,j}$'s from $\mathsf{seed}$. The size of the obtained compressed ciphertext vector is $\lambda + t \cdot \log_2(q)$ against $t \cdot (n+1) \cdot \log_2(q)$ for a standard TFHE encryption, meaning a compression by a factor about $(n+1)$.

**Table 1:** Bandwidth of homomorphic ciphertexts (in bits).

| **Approach used** | **Naive** | **Compressed** | `Transistor` |
|---|---|---|---|
| Fixed cost | 0 | $\lambda$ | $\lambda + (\lvert\mathcal{K}\rvert + \lvert\mathcal{W}\rvert) \cdot \log_2(q)$ |
| Per message in $\mathbb{Z}_p$ | $(n+1) \cdot \log_2(q)$ | $\log_2(q)$ | $\log_2(p)$ |
| Expansion factor | $(n+1) \cdot \log_2(q)/\log_2(p)$ | $\log_2(q)/\log_2(p)$ | 1 |

This compressed TFHE encryption method can be applied directly to transmit homomorphically encrypted data from the user to the server. Alternatively, it can be combined with transciphering to encrypt a symmetric key. The resulting bandwidth requirements and the corresponding plaintext-to-ciphertext expansion factor are summarized in Table 1, where they are further compared with the naive (uncompressed) TFHE encryption. In particular, for `Transistor`, a wrapped key is of size $\lambda + (|\mathcal{K}| + |\mathcal{W}|) \cdot \log_2(q)$, (which in our case gives 784 bytes) for a security of $\lambda = 128$ bits (target security of `Transistor`), the standard choice of $q = 2^{64}$ (which we use in our implementation) and $|\mathcal{K}| + |\mathcal{W}| = 96$ per the specification of `Transistor` (see Section 4). This fixed cost is hence very quickly amortized while the amount of data to encrypt grows. Moreover, this approach can be applied to the server keys as well, which are actually encryptions of the secret key's bits. We took this optimization into account in our estimations of the server key sizes in Table 4.

*Compressing further.* In Appendix C, we introduce an additional tweak to compress a TFHE encryption further by truncating its noisy bits.

### 6.2 Transciphering vs. Data Representation

Managing data representation is a common challenge when working with TFHE. Since this scheme is only efficient at very low precision, an abstraction layer is required to construct practical data types (e.g., 8-, 32-, or 64-bit integers) from smaller encrypted chunks. Common constructions include radix-based decompositions and Chinese Remainder Theorem (CRT) representations, leading to different efficiency trade-offs. Carry propagation in radix-based representations is notoriously slow due to the large number of required bootstrappings, while CRT representations impose constraints on feasible operations. These constructions have been studied in [13].

As a result, there is no universal representation that is optimal for all homomorphic operations. Thankfully, the representation of data in the transciphering algorithm can be chosen independently of that of the homomorphic application running on the server. If the representation in $\mathbb{Z}_p$ does not suit the application, the server can convert the ciphertexts to the desired representation before running the application. We stress that this additional step of conversion would be necessary for any transciphering algorithm, as the data format desired in output of transciphering is completely application-dependent.

As a concrete example, assume that the data to be encrypted (i.e., the input to the homomorphic computation) consists of elements from $\mathbb{Z}_{16}$. The overall

transciphering process unfolds as follows. On the client side, the plaintext is first embedded from $\mathbb{Z}_{16}$ into $\mathbb{Z}_{17}$ before being encrypted using `Transistor`. On the server side, the keystream is homomorphically generated and then used to homomorphically decrypt the ciphertext. This results in a TFHE encryption of the original plaintext, now embedded in $\mathbb{Z}_{17}$, meaning that the plaintext space for the TFHE encryption is $\mathbb{Z}_{17}$. A programmable bootstrapping (PBS) operation is then applied to switch the plaintext space from $\mathbb{Z}_{17}$ back to $\mathbb{Z}_{16}$.

In terms of computation, this process adds one PBS per $\mathbb{Z}_{16}$-element of the original plaintext, in addition to the four PBS per element required for keystream generation with `Transistor`. Moreover, embedding $\mathbb{Z}_{16}$ into $\mathbb{Z}_{17}$ increases the size of the encrypted data by a factor of $1 + 1/16 = 1.0625$.

This approach can be generalized to address other plaintext representations. In particular, for larger chunks of bits, the bootstrapping operation would allow to merge several elements of $\mathbb{Z}_{16}$ (embedded into $\mathbb{Z}_{17}$) into one element of $\mathbb{Z}_{2^\ell}$ with $\ell > 4$. On the other hand, one may split an element of $\mathbb{Z}_{16}$ (or its $\mathbb{Z}_{17}$ embedding) into 4 elements of $\mathbb{Z}_2$ using a PBS with multiple look-up tables ("PBSmanyLUT") as proposed in [31].

### 6.3 Detailed Homomorphic Implementations

In the following, we provide a more detailed way of how we implemented the homomorphic version of `Transistor`.

*Homomorphic evaluation of LFSRs.* The `Transistor` design involves two LFSRs operating on elements of $\mathbb{F}_{17}$. The standard way to implement an LFSR is to evaluate the linear feedback function on the state at each clock cycle, thus producing a new element that enters the state, while the state is shifted to output an element. We suggest the *silent LFSR* approach for the homomorphic evaluation of LFSRs. In this approach, the encrypted LFSR state is immutable to avoid any noise growth in the underlying ciphertexts (hence keeping the LFSR "silent"). We use the fact that every output element of the LFSR can be expressed as a linear combination of the initial state. So, at each clock cycle, we compute *in the clear* the coefficients of this linear combination and homomorphically evaluate it on the immutable encrypted state. This process is depicted in Algorithm 1.

*Homomorphic evaluation of* `Transistor`. The complete homomorphic evaluation of a round of on clock cycle of `Transistor` is depicted in Algorithm 2, using $\mathcal{K}$.`clock` and $\mathcal{W}$.`clock` as subroutines (i.e., Algorithm 1 evaluated on the key schedule and whitening LFSRs). The most computation intensive part of the algorithm is by far the evaluation of the PBS in SubDigits which can be fully parallelized to reduce the latency.

### 6.4 TFHE Parameters

We discuss hereafter the selection of the different TFHE parameters involved in the homomorphic implementation of `Transistor`. Here is the list of the different parameters to be considered:

**Algorithm 1** `LFSR.clock` - Produce a pseudo random element of the state.

**State:** $\begin{cases} \ell : \text{ Size of the state of the LFSR.} \\ (u_1, \ldots, u_\ell) : \text{ Encrypted initial state of the LFSR.} \\ (\lambda_1^{(0)}, \ldots, \lambda_\ell^{(0)}) : \text{ Coefficients of retroaction in the definition of the LFSR.} \\ (\lambda_1^{(i)}, \ldots, \lambda_\ell^{(i)}) : \text{ Previous coefficients used in the linear combination.} \end{cases}$

**Result:** $\begin{cases} o^{(i)} : \text{ Encryption of the } i\text{-th pseudorandom element of } \mathbb{F}_{17}. \\ (\lambda_1^{(i+1)}, \ldots, \lambda_\ell^{(i+1)}) : \text{ Updated coefficients of the linear combination.} \end{cases}$

$o^{(i)} \leftarrow 0$
`/* Evaluation of the linear combination */`
**for** $k \in \{1, \ldots, \ell\}$ **do**
  $o^{(i)} \leftarrow \texttt{SumTFHE}(o^{(i)}, \texttt{ClearMultTFHE}(u_k, \lambda_k^{(i)}))$
**end**
`/* Update of the next coefficients */`
**for** $k \in \{2, \ldots, \ell\}$ **do**
  $\lambda_k^{(i+1)} \leftarrow \lambda_{k-1}^{(i)} + \lambda_\ell^{(i)} \cdot \lambda_k^{(0)}$
**end**
$\lambda_1^{(i+1)} \leftarrow \lambda_\ell^{(i)} \cdot \lambda_1^{(0)}$
**return** $o^{(i)}$

- $q$: the modulo used for the ciphertexts,
- $n_{\text{short}}$: the dimension of the LWE ciphertexts used within the PBS algorithm,
- $k$: the dimension of GLWE ciphertexts,
- $N$: the degree of the polynomials in GLWE ciphertexts,
- $n_{\text{long}} = k \cdot N$: the dimension of the LWE ciphertexts extracted from GLWE ciphertexts,
- $\sigma_{\text{short}}, \sigma_{\text{long}}$: the standard deviation of the Gaussian distribution of fresh noise in the ciphertexts of dimension respectively $n_{\text{short}}$ and $n_{\text{long}}$,
- $B_{KS}$ and $B_{BR}$: the bases used in the gadget decompositions occurring in respectively the Keyswitching and the BlindRotate phases,
- $\ell_{KS}$ and $\ell_{BR}$: the levels of those decompositions.

Throughout the homomorphic evaluation of `Transistor`, the manipulated ciphertexts can be of three different types: LWE ciphertexts of dimension $n_{\text{short}}$, LWE ciphertexts of dimension $n_{\text{long}}$, or GLWE ciphertexts of dimension $k$ and polynomial degree $N$. Figure 9 shows the ciphertext format at the different steps of the homomorphic evaluation.

*Optimization of the TFHE parameters.* To generate a set of parameters, we use the method developed in [13]. Given the negligible noise contribution from the LFSRs, the FSM can be modeled using the *atomic pattern* introduced in [13] (specifically the instance referred to as $\mathcal{A}^{(\text{CJP21})}$), which is a pattern of homomorphic operators taking a set of ciphertexts in input, computing linear combinations of those ciphertexts and applying a programmable bootstrapping to each of them. The FSM round in `Transistor` which composes a multiplication by a

**Algorithm 2** `Transistor.clock` - Produce $r$ encypted elements of the key stream

**State:** $\begin{cases} \mathcal{K} : \text{the LFSR used for the pseudo-keyschedule and its state (cf Algorithm 1).} \\ \mathcal{W} : \text{the LFSR used for the whitening.} \\ X = \begin{pmatrix} x_{1,1} & \dots & x_{1,\sqrt{m}} \\ \dots & \dots & \dots \\ x_{\sqrt{m},1} & \dots & x_{\sqrt{m},\sqrt{m}} \end{pmatrix} : \text{ Encrypted state of the FSM} \end{cases}$

**Result:** $\{ Y = (y_1, \dots, y_r) :$ Encryption of $r$ elements of the key stream

```
/* Compute the pseudo-key schedule and adds it to the FSM                    */
for i ∈ [1, √m] do
    for j ∈ [1, √m] do
        k_{i,j} ← K.clock()
        x_{i,j} ← SumTFHE(x_{ij}, k_{i,j})
    end
end
/* Compute SubDigits with a layer of PBS                                     */
for i ∈ [1, √m] do
    for j ∈ [1, √m] do
        x_{i,j} ← PBS_TFHE(x_{i,j}, S)
    end
end
/* Extract the output bits and whiten them                                   */
(y_1, ..., y_r) ← φ(X)
 for i ∈ [1, r] do
    w_i ← W.clock()
    y_i ← SumTFHE(y_i, w_i)
end
/* Compute ShiftRows, (same as in clear)                                     */
X ← SR(X)
/* Compute MixColumns                                                        */
for i ∈ [1, √m] do
    for j ∈ [1, √m] do
        z_{i,j} ← 0
        for k ∈ [1, √m] do
            z_{i,j} ← SumTFHE(z_{i,j}, ClearMultTFHE(x_{k,j}, MC_{i,k}))
        end
    end
end
return Y
```

constant matrix (MixColumns), followed by a bootstrapping step (SubDigits) is precisely an instance of such an atomic pattern. The framework proposed in [13] generates parameters that guarantee a specified security level $\lambda$ for the LWE encryption and target error probability $p_{\text{err}}$, while optimizing the PBS to be as

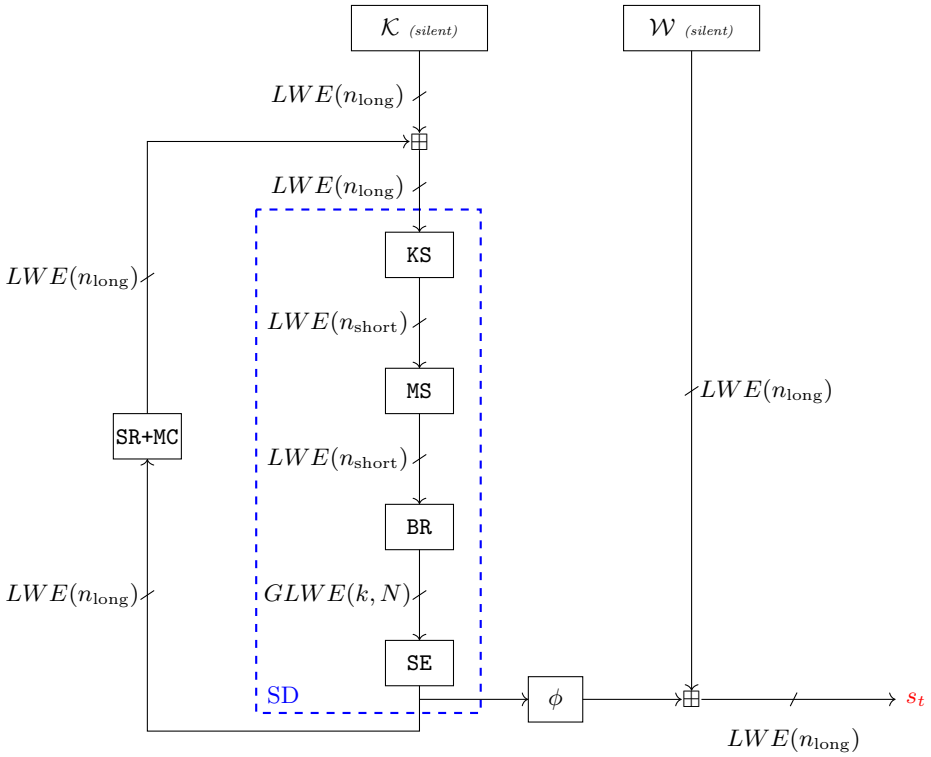



**Fig. 9:** Types and shapes of ciphertexts in homomorphic `Transistor`. The SubDigits is broken down into its elementary components

**Table 2:** TFHE Parameters used in our experiments

| $p_{\text{err}}$ | $q$ | $n_{\text{short}}$ | $k$ | $N$ | $\sigma_{\text{short}}$ | $\sigma_{\text{long}}$ | $B_{BR}$ | $\ell_{\text{BR}}$ | $B_{KS}$ | $\ell_{\text{KS}}$ | $\lambda_{\text{short}}$ | $\lambda_{\text{long}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^{-40}$ | $2^{64}$ | 788 | 2 | 1024 | $2^{47}$ | $2^{14}$ | $2^{23}$ | 1 | $2^{4}$ | 3 | 131.8 | 128.9 |
| $2^{-128}$ | $2^{64}$ | 774 | 1 | 2048 | $2^{47}$ | $2^{14}$ | $2^{23}$ | 1 | $2^{3}$ | 5 | 131.8 | 128.9 |

fast as possible. Table 2 shows the parameters used for our experiments, all ensuring 128 bits of security. The obtained security levels $\lambda_{\text{short}}$ et $\lambda_{\text{long}}$ have been estimated using the `lattice estimator` [1].

*Encryption security.* The security level (in bits) of the LWE ciphertexts is a function of the modulus $q$, the dimension $n$ and the noise standard deviation $\sigma$. While no explicit formula exists for this function, the *lattice estimator* tool allows to produce an estimation of this function by simulating the main attacks of the literature [1], which we denote $\mathcal{O}$ (for *security oracle*). The selected TFHE parame-

ters are constrained to satisfy $\mathcal{O}(q, n, \sigma) \geq \lambda$ for both $(q, n, \sigma) = (q, n_{\text{short}}, \sigma_{\text{short}})$ and $(q, n, \sigma) = (q, n_{\text{long}}, \sigma_{\text{long}})$.

*Correctness of the PBS.* To compute the error probability $p_{\text{err}}$, we have to evaluate the variances occurring inside the programmable bootstrapping of the SubDigits layer. We provide such an analysis in Appendix B.2. In the following, we denote the maximum error probability of the bootstrapping by $p_{\text{err}} := 2^{-\kappa}$. We aim to choose parameters such that the PBS outputs a correct ciphertext with probability at least $1 - p_{\text{err}}$. This translates to the following constraint:

$$\sigma^2_{\text{in-BR}} \leq C(\kappa) \cdot \left(\frac{1}{4p}\right)^2 \quad \text{with} \quad C(\kappa) := \left(\frac{1}{\sqrt{2} \cdot \mathsf{erfc}^{-1}(2^{-\kappa})}\right)^2 .$$

Under the Gaussian assumption, the noise in input of the BlindRotate is lower than $\mathsf{erfc}^{-1}(2^{-\kappa}) \cdot \sqrt{\sigma^2_{\text{in-BR}}}$ with probability $1 - 2^{-\kappa}$. The above constraint thus implies that, with probability $1-2^{-\kappa}$, the noise is lower than $1/4p$ which ensures the correctness of the PBS.

As $|\mathcal{W}| \leq |\mathcal{K}|$, we can check that we always have $\sigma^2_{\text{out}} \leq \sigma^2_{\text{in-PBS}}$ which implies that the output is always bootstrappable with correctness probability at least $1 - p_{\text{err}}$ in the subsequent bootstrapping.

### 6.5 Performances

We provide hereafter some benchmarks of our implementation of `Transistor` for two sets of parameters tailored for two different error probabilities. We first consider $p_{\text{err}} = 2^{-40}$, which is a common choice in the literature to benchmark homomorphic implementations. Our results for this error probability allow a fair comparison with the state of the art. While such an error probability theoretically allows transciphering to be error-free with a large amount of data with good probability, some recent works have shown that non-negligible error probabilities could be exploited by an adversary in some contexts [24,25]. Thus, we also provide another set of parameters and associated benchmark for $p_{\text{err}} = 2^{-128}$.

Our implementation relies on a customized version of `tfhe-rs` [79] which has been adapted to support odd $p$ (size of the plaintext space) as described in [17]. The experiments were carried on a processor 12th Gen Intel(R) Core(TM) i5-1245U with 4.4 GHz. Table 3 summarizes the obtained timings for the two sets of parameters. The throughput is computed assuming that $\log_2(17)$ bits are encoded on one element of $\mathbb{F}_{17}$. Encoding 4 bits on each element would scale the throughput by a factor $4/\log_2(17) \approx 0.98$.

Although our current implementation does not leverage the inherent parallelism of `Transistor`, it is important to note that it can be easily parallelized across 16 threads. Specifically, during the SubDigits steps, which dominate the overall runtime, the 16 PBS operations can be executed concurrently. This parallelization would result in nearly a 16-fold reduction in total execution time.

Without taking into account the server key(whose sizes are shown in Table 4), and using compressed encryption (Section 6.1), transciphering 1 KB of plain

**Table 3:** Performances of our two instances of `Transistor`.

| $p_{\text{err}}$ | Time for one PBS | Latency (one round) | Throughput |
|---|---|---|---|
| $2^{-40}$ | 11.9 ms | 195 ms | 83.84 bits/s |
| $2^{-128}$ | 15.28 ms | 251 ms | 65.10 bits/s |

**Table 4:** Size of the server keys for the two considered sets of parameters.

| | Theoretical sizes | Sizes for $p_{\text{err}} = 2^{-40}$ | Sizes for $p_{\text{err}} = 2^{-128}$ |
|---|---|---|---|
| KSK | $n_{\text{long}} \cdot l_{\text{KS}} \cdot \log_2 q$ | 49 KB | 82 KB |
| BSK | $n_{\text{short}} \cdot l_{\text{BS}} \cdot \log_2 q \cdot N \cdot (k+1)$ | 6.5 MB | 12.7 MB |

data requires 1.78 KB of data to be sent, instead of 64 KB. For larger amounts of message, the volume of the encrypted symmetric key becomes negligible with respect to the message: for 1 MB of plain data, we use 1.0008 MB, and for 1 GB, this goes down to 1.000001 GB. The two sets of parameters yield the same bandwidth consumption, but not the same running time as shown in Table 3.

By applying the "free truncation optimization" introduced in Appendix C, we can reduce the volume of the encrypted symmetric key by a factor 0.7. This is particularly useful when transciphering a small volume of data (the volume of the encrypted key being preponderant). For example, to transcipher 1 KB of data, using this technique decreases the volume from 1.78 KB to 1.54 KB.

in Table 4 we provide the sizes for the server keys, namely the *key-switching key* (KSK) and the *bootstrapping key* (BSK) while using the ciphertext compression technique described in Section 6.1. Those keys are only generated and communicated to the server once (during some user enrollment step).

### 6.6 Comparisons to the State of the Art

**Comparisons with other TFHE-friendly ciphers.** In what follows, we compare `Transistor` with several of the most competitive state-of-the-art schemes. Our results are summarized in Table 5. Although such comparisons must be interpreted with care — due to differences in libraries, hardware platforms, and bootstrapping error probabilities — they still offer valuable insights into the relative efficiency and trade-offs of these approaches.

In [8], `Trivium` and `Kreyvium` were evaluated on a powerful AWS instance, which makes direct comparison with our local experiments impractical. However, an important distinction is that `Transistor` requires no setup phase, unlike these ciphers. Also, the implementation optimizes the running time by switching between two sets of parameters, doubling the size of the evaluation keys.

`Margrethe` [4] has low noise ratios ($2^{-15.3}$ or $2^{-21.9}$), compared to around $2^{-12}$ for `Transistor`. Its authors also report low latencies of 27 or 54 ms, and high throughputs of 147 or 73 bits/s (depending on the configuration). However, these come at the cost of much larger sizes for the encryptions of the symmetric key. Indeed, the use of Vertical Packing mandates that symmetric keys be

**Table 5:** Performance of state-of-the-art TFHE-friendly ciphers (single-threaded when applicable). Communication cost accounts for both the encrypted symmetric key and the evaluation keys.

| Cipher | Setup | Latency | Throughput | Communication Cost[a] | $p_{\text{err}}$ |
|---|---|---|---|---|---|
| `Trivium` [8] (128 thr.) | 2259 ms | 121 ms | 529 bits/s | 640 B + 35.6 MB † | $2^{-40}$ |
| `Kreyvium` [8] (128 thr.) | 2883 ms | 150 ms | 427 bits/s | 1024 B + 35.6 MB † | $2^{-40}$ |
| `Margrethe` [4] | No | 27.2 ms | 147.06 bits/s | 64 MB * | $< 2^{-1000}$ |
| | No | 54.2 ms | 73.8 bits/s | 128 MB * | $< 2^{-1000}$ |
| PRF-based construction [40] | No | 5.675 ms | 881 bits/s | 32.8 MB = 8.9 MB + 23.9 MB | $2^{-64}$ |
| `FRAST` [32] | 25 s (8 thr.) | 6.2 s | 20.66 bits/s | 34.05 MB = 148 KB + 33.91 MB | $2^{-80}$ |
| `Transistor` | No | 251 ms | 65.10 bits/s | 13.54 MB = 780 B + 12.78 MB | $2^{-128}$ |

* In `Margrethe`, no keyswitching nor bootstrapping keys are required.
† Values recomputed from the data of the papers. For consistency's sake, we applied the compression of ciphertexts of Section 6.1 to estimate the communication cost.

encrypted under GGSW form, resulting in hundreds of megabytes. Although an alternative (lifting from LWE to GGSW using circuit bootstrapping) could reduce the transmission size, it would significantly degrade performance.

Similarly, Deo et al. proposed [40] a pseudorandom function whose security relies on the hardness of the LWR problem [9]. It enables a stream cipher–like transciphering scheme, where each pseudorandom element in $\mathbb{Z}_q$ (with $q = 2^5$) is produced using a single bootstrapping. Their implementation reaches up to 881 bits/s on their hardware, surpassing `Transistor` in throughput. However, as with `Margrethe`, this efficiency comes at the cost of a significant increase in key size: their PRF requires 500 to 1000 elements encrypted in GGSW form, which is much larger than the 96 elements in LWE form for `Transistor`.

The authors of `FRAST` [32] implemented it with `tfhe-rs`, like `Transistor`. It targets 128-bit security with an error probability of $2^{-80}$. On the same platform, our instance of `Transistor` (with a tighter error bound of $2^{-128}$) achieves three times higher throughput, significantly lower latency, and does not require any setup phase—unlike `FRAST`, which involves a 25-second setup. In addition, `FRAST` requires substantially larger evaluation key material, due to its use of multiple derivatives of the PBS algorithm.

**Comparisons with other homomorphic schemes.** TFHE yields very different trade-offs between the various performance metrics (latency, throughput, bandwith consumption, ...), compared to other FHE schemes. In scenarios where throughput is a priority, TFHE—and by extension, `Transistor`—is generally not the most suitable choice. However, TFHE shines in use cases that require low-latency homomorphic computations and efficient evaluation of look-up tables (LUTs), thanks to its native support for fast programmable bootstrapping. In the following, we provide a brief survey of some transciphering approaches built upon alternative homomorphic encryption schemes.

For CKKS-based transciphering, we look at the AES evaluation using the so-called `XBOOT` optimization proposed in [67]. While the reported amortized throughput significantly outperforms that of `Transistor`, it comes at the cost of a much higher latency—153s and 236s using 64 threads, compared to 251ms for `Transistor` running on a single thread. We observe similar results for BGV-based transciphering. In this case we instead consider the optimized evaluation of `RASTA` [41] presented in [67]. Using the same `XBOOT` optimization as the CKKS variant, it also yields high latencies: 303s, again with 64-thread parallelism.

Lastly, `FINAL` [18] is another, more recent homomorphic encryption scheme based on NTRU. Its bootstrapping is approximately 30% faster than TFHE's, but it is not programmable, and therefore does not support LUT evaluation. The works [63,34] implement the stream cipher `Filip` using `FINAL`, relying on the Improved Filter Permutator paradigm, without LUTs' evaluation. The competitive performances of these implementations (resp. 159 bits/s and 381 bits/s) comes with a trade-off in key size: encrypting the symmetric key requires resp. 215 MB and 200 MB, versus 4.8 MB for `Transistor` (uncompressed). This is mainly due to the size of the key register in `Filip` ($2^{14}$ bits), while `Transistor` only requires the upload of 96 elements in $\mathbb{F}_{17}$ (around 400 bits).

## 7 Conclusion

`Transistor` is a new stream cipher design tailored to TFHE transciphering, that significantly outperforms the state-of-the-art of TFHE-friendly stream ciphers. After analyzing the constraints of the TFHE setting in the context of a symmetric cipher, we designed `Transistor` by combining an LFSR-based key schedule, an LFSR-based whitening, and a non-linear FSM with an AES-like structure. We report implementation results of `Transistor` using state-of-the-art TFHE, using a trick to implement *silent* LFSRs. This general structure can be easily adapted to other contexts, and we believe it will find applications beyond TFHE.

Our initial cryptanalysis of `Transistor` provides a solid theoretical background for this type of stream ciphers. We introduce arguments based on the minimum number of rounds needed to have a non-zero correlation between ciphertext and master key, an inherent property of the round function of the FSM which we consider to be of independent interest. We expect that future work will explore in more details how this property is related to the design of the round function, and how to optimize it without increasing the cost of the cipher.

## Acknowledgements

This work was supported by the French Agence Nationale de la Recherche through the SWAP project under Contract ANR-21-CE39-0012. Léo Perrin is supported by the European Research Council (ERC *ReSCALE*, grant agreement no. 101041545 "ReSCALE") and Jules Baudrin through ERC project 101096871 (*BRIDGE*). This work was also supported by project *Cryptanalyse* from PEPR Cybersécurité (22-PECY-0010). The authors thank Cyprien Gauthier for his

help with the generation of the S-box of `Transistor`, as well as Tim Beyne and the anonymous reviewers for their insightful comments which enhanced the technical quality of the paper.

## References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015). https://doi.org/10.1515/jmc-2015-0016
2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 430–454. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_17
3. Anderson, R.J.: Searching for the optimum correlation attack. In: Preneel, B. (ed.) FSE'94. LNCS, vol. 1008, pp. 137–143. Springer, Berlin, Heidelberg (Dec 1995). https://doi.org/10.1007/3-540-60590-8_11
4. Aranha, D.F., Guimarães, A., Hoffmann, C., Méaux, P.: Secure and efficient transciphering for FHE-based MPC. Cryptology ePrint Archive, Paper 2024/1702 (2024), https://eprint.iacr.org/2024/1702
5. Babbage, S.: A space/time tradeoff in exhaustive search attacks on stream ciphers. European Convention on Security and Detection, IEE Conference Publication No. 408 (May 1995)
6. Baignères, T., Junod, P., Vaudenay, S.: How far can we go beyond linear cryptanalysis? In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 432–450. Springer, Berlin, Heidelberg (Dec 2004). https://doi.org/10.1007/978-3-540-30539-2_31
7. Baignères, T., Stern, J., Vaudenay, S.: Linear cryptanalysis of non binary ciphers. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) SAC 2007. LNCS, vol. 4876, pp. 184–211. Springer, Berlin, Heidelberg (Aug 2007). https://doi.org/10.1007/978-3-540-77360-3_13
8. Balenbois, T., Orfila, J., Smart, N.P.: Trivial transciphering with Trivium and TFHE. In: Brenner, M., Costache, A., Rohloff, K. (eds.) 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Copenhagen, Denmark, 26 November 2023. pp. 69–78. ACM (2023). https://doi.org/10.1145/3605759.3625255
9. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 719–737. Springer, Berlin, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_42
10. Bariant, A., Boeuf, A., Lemoine, A., Ayala, I.M., Øygarden, M., Perrin, L., Raddum, H.: The algebraic FreeLunch: Efficient Gröbner basis attacks against arithmetization-oriented primitives. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part IV. LNCS, vol. 14923, pp. 139–173. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68385-5_5
11. Belaïd, S., Bon, N., Boudguiga, A., Sirdey, R., Trama, D., Ye, N.: Further improvements in AES execution over TFHE: Towards breaking the 1 sec barrier. Cryptology ePrint Archive, Paper 2025/075 (2025), https://eprint.iacr.org/2025/075
12. Berbain, C., Gilbert, H.: On the security of IV dependent stream ciphers. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 254–273. Springer, Berlin, Heidelberg (Mar 2007). https://doi.org/10.1007/978-3-540-74619-5_17

13. Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J., Tap, S.: Parameter optimization and larger precision for (T)FHE. J. Cryptol. **36**(3), 28 (2023). https://doi.org/10.1007/S00145-023-09463-5
14. Beyne, T.: A geometric approach to linear cryptanalysis. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part I. LNCS, vol. 13090, pp. 36–66. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92062-3_2
15. Biryukov, A.: The design of a stream cipher LEX. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 67–75. Springer, Berlin, Heidelberg (Aug 2007). https://doi.org/10.1007/978-3-540-74462-7_6
16. Biryukov, A., De Cannière, C., Quisquater, M.: On multiple linear approximations. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 1–22. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28628-8_1
17. Bon, N., Pointcheval, D., Rivain, M.: Optimized Homomorphic Evaluation of Boolean Functions. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(3), 302–341 (Jul 2024). https://doi.org/10.46586/tches.v2024.i3.302-341
18. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: FINAL: Faster FHE instantiated with NTRU and LWE. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 188–215. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22966-4_7
19. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012. pp. 309–325. ACM (Jan 2012). https://doi.org/10.1145/2090236.2090262
20. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3), 13:1–13:36 (2014). https://doi.org/10.1145/2633600, https://doi.org/10.1145/2633600
21. Brynielsson, L.: A short proof of the Xiao-Massey lemma. IEEE Trans. Inf. Theory **35**(6), 1344 (1989)
22. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 313–333. Springer, Berlin, Heidelberg (Mar 2016). https://doi.org/10.1007/978-3-662-52993-5_16
23. Canteaut, A., Trabbia, M.: Improved fast correlation attacks using parity-check equations of weight 4 and 5. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 573–588. Springer, Berlin, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_40
24. Checri, M., Sirdey, R., Boudguiga, A., Bultel, J.P.: On the practical CPAD security of "exact" and threshold FHE schemes and libraries. Cryptology ePrint Archive, Report 2024/116 (2024), https://eprint.iacr.org/2024/116
25. Cheon, J.H., Choe, H., Passelègue, A., Stehlé, D., Suvanto, E.: Attacks against the INDCPA-D security of exact FHE schemes. Cryptology ePrint Archive, Report 2024/127 (2024), https://eprint.iacr.org/2024/127
26. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
27. Chepyzhov, V.V., Johansson, T., Smeets, B.J.M.: A simple algorithm for fast correlation attacks on stream ciphers. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 181–195. Springer, Berlin, Heidelberg (Apr 2001). https://doi.org/10.1007/3-540-44706-7_13

28. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 3–33. Springer, Berlin, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_1
29. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 377–408. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_14
30. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (Jan 2020). https://doi.org/10.1007/s00145-019-09319-x
31. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 670–699. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_23
32. Cho, M., Chung, W., Ha, J., Lee, J., Oh, E., Son, M.: FRAST: TFHE-friendly cipher based on random S-Boxes. IACR Trans. Symmetric Cryptol. **2024**(3), 1–43 (2024). https://doi.org/10.46586/TOSC.V2024.I3.1-43
33. Clet, P.E., Zuber, M., Boudguiga, A., Sirdey, R., Gouy-Pailler, C.: Putting up the swiss army knife of homomorphic calculations by means of TFHE functional bootstrapping. Cryptology ePrint Archive, Report 2022/149 (2022), https://eprint.iacr.org/2022/149
34. Cong, K., Das, D., Park, J., Pereira, H.V.L.: SortingHat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 563–577. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560702
35. Cosseron, O., Hoffmann, C., Méaux, P., Standaert, F.X.: Towards case-optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part III. LNCS, vol. 13793, pp. 32–67. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22969-5_2
36. Courtois, N.: Fast algebraic attacks on stream ciphers with linear feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 176–194. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_11
37. Courtois, N., Meier, W.: Algebraic attacks on stream ciphers with linear feedback. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 345–359. Springer, Berlin, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_21
38. Cover, T.M., Thomas, J.A.: Elements of information theory (2. ed.). Wiley (2006), http://www.elementsofinformationtheory.com/
39. De Cannière, C.: Trivium: A stream cipher construction inspired by block cipher design principles. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 171–186. Springer, Berlin, Heidelberg (Aug / Sep 2006). https://doi.org/10.1007/11836810_13
40. Deo, A., Joye, M., Libert, B., Curtis, B.R., de Bellabre, M.: Homomorphic evaluation of LWR-based PRFs and application to transciphering. Cryptology ePrint Archive, Report 2024/665 (2024), https://eprint.iacr.org/2024/665
41. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A cipher with low anddepth and few ands per bit. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara,

CA, USA, August 19-23, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10991, pp. 662–692. Springer (2018). https://doi.org/10.1007/978-3-319-96884-1_22, https://doi.org/10.1007/978-3-319-96884-1_22
42. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: EUROCRYPT (1). pp. 617–640. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_24
43. Dunkelman, O., Keller, N.: A new attack on the LEX stream cipher. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 539–556. Springer, Berlin, Heidelberg (Dec 2008). https://doi.org/10.1007/978-3-540-89255-7_33
44. Duval, S., Lallemand, V., Rotella, Y.: Cryptanalysis of the FLIP family of stream ciphers. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 457–475. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_17
45. Eichlseder, M., Nageler, M., Primas, R.: Analyzing the linear keystream biases in AEGIS. IACR Trans. Symm. Cryptol. **2019**(4), 348–368 (2019). https://doi.org/10.13154/tosc.v2019.i4.348-368
46. Ekdahl, P., Johansson, T.: A new version of the stream cipher SNOW. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 47–61. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/3-540-36492-7_5
47. Ekdhal, P., Johansson, T.: SNOW - a new stream cipher. In: Proceedings of the first Nessie workshop (2000)
48. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. p. 144 (2012), http://eprint.iacr.org/2012/144
49. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) STOC 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440
50. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 850–867. Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_49
51. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_5
52. Gilbert, H., Boissier, R.H., Jean, J., Reinhard, J.R.: Cryptanalysis of elisabeth-4. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part III. LNCS, vol. 14440, pp. 256–284. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8727-6_9
53. Golic, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 239–255. Springer, Berlin, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_17
54. Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in TFHE. IACR Transactions on Cryptographic Hardware and Embedded Systems p. 229–253 (Feb 2021). https://doi.org/10.46586/tches.v2021.i2.229-253, https://tches.iacr.org/index.php/TCHES/article/view/8793
55. Hell, M., Johansson, T.: Advanced linear Cryptanalysis of block and stream ciphers, chap. Linear attacks on stream ciphers. IOS Press (2011)
56. Hell, M., Johansson, T., Maximov, A., Meier, W.: The grain family of stream ciphers. In: Robshaw, M.J.B., Billet, O. (eds.) New Stream Cipher Designs - The eSTREAM Finalists, Lecture Notes in Computer Science, vol. 4986, pp. 179–190. Springer (2008). https://doi.org/10.1007/978-3-540-68351-3_14

57. Hermelin, M., Nyberg, K.: Multidimensional linear distinguishing attacks and boolean functions. Cryptogr. Commun. **4**(1), 47–64 (2012). https://doi.org/10.1007/S12095-011-0053-3, https://doi.org/10.1007/s12095-011-0053-3
58. Hoffmann, C., Méaux, P., Standaert, F.X.: The patching landscape of elisabeth-4 and the mixed filter permutator paradigm. In: Chattopadhyay, A., Bhasin, S., Picek, S., Rebeiro, C. (eds.) INDOCRYPT 2023, Part I. LNCS, vol. 14459, pp. 134–156. Springer, Cham (Dec 2023). https://doi.org/10.1007/978-3-031-56232-7_7
59. Kluczniak, K., Schild, L.: Fdfb: Full domain functional bootstrapping towards practical fully homomorphic encryption. IACR Transactions on Cryptographic Hardware and Embedded Systems p. 501–537 (Nov 2022). https://doi.org/10.46586/tches.v2023.i1.501-537
60. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes. North-Holland Publishing Company (1977)
61. Méaux, P., Carlet, C., Journault, A., Standaert, F.X.: Improved filter permutators for efficient FHE: Better instances and implementations. In: Hao, F., Ruj, S., Sen Gupta, S. (eds.) INDOCRYPT 2019. LNCS, vol. 11898, pp. 68–91. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-35423-7_4
62. Méaux, P., Journault, A., Standaert, F.X., Carlet, C.: Towards stream ciphers for efficient FHE with low-noise ciphertexts. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 311–343. Springer, Berlin, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49890-3_13
63. Méaux, P., Park, J., Pereira, H.V.L.: Towards practical transciphering for FHE with setup independent of the plaintext space. CiC **1**(1), 20 (2024). https://doi.org/10.62056/anxrxrxqi
64. Meier, W., Staffelbach, O.: Fast correlation attacks on stream ciphers (extended abstract). In: Günther, C.G. (ed.) EUROCRYPT'88. LNCS, vol. 330, pp. 301–314. Springer, Berlin, Heidelberg (May 1988). https://doi.org/10.1007/3-540-45961-8_28
65. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C., Yung, M., Lin, D. (eds.) Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. Lecture Notes in Computer Science, vol. 7537, pp. 57–76. Springer (2011). https://doi.org/10.1007/978-3-642-34704-7_5, https://doi.org/10.1007/978-3-642-34704-7_5
66. Naehrig, M., Lauter, K.E., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Cachin, C., Ristenpart, T. (eds.) Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011. pp. 113–124. ACM (2011), https://dl.acm.org/citation.cfm?id=2046682
67. Niu, C., Huang, Z., Yang, Z., Chen, Y., Kong, L., Hong, C., Wei, T.: XBOOT: Free-XOR gates for CKKS with applications to transciphering. Cryptology ePrint Archive, Paper 2025/074 (2025), https://eprint.iacr.org/2025/074
68. Nyberg, K., Wallén, J.: Improved linear distinguishers for SNOW 2.0. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 144–162. Springer, Berlin, Heidelberg (Mar 2006). https://doi.org/10.1007/11799313_10
69. Perrin, L.: Security analysis of XHASH8/12. Cryptology ePrint Archive, Report 2024/605 (2024), https://eprint.iacr.org/2024/605
70. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) Proceedings of the 37th Annual ACM

Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005. pp. 84–93. ACM (2005). https://doi.org/10.1145/1060590.1060603
71. Sakamoto, K., Liu, F., Nakano, Y., Kiyomoto, S., Isobe, T.: Rocca: An efficient AES-based encryption scheme for beyond 5g. IACR Trans. Symm. Cryptol. **2021**(2), 1–30 (2021). https://doi.org/10.46586/tosc.v2021.i2.1-30
72. SHA-3 standard: Permutation-based hash and extendable-output functions. National Institute of Standards and Technology, NIST FIPS PUB 202, U.S. Department of Commerce (Aug 2015). https://doi.org/10.6028/NIST.FIPS.202
73. Shi, Z., Jin, C., Zhang, J., Cui, T., Ding, L., Jin, Y.: A correlation attack on full SNOW-V and SNOW-vi. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 34–56. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_2
74. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Trans. Computers **34**(1), 81–85 (1985). https://doi.org/10.1109/TC.1985.1676518
75. Tap, S.: Constructing new tools for efficient homomorphic encryption. Theses, Université de Rennes (Dec 2023), https://theses.hal.science/tel-04587370
76. Todo, Y., Isobe, T., Meier, W., Aoki, K., Zhang, B.: Fast correlation attack revisited - cryptanalysis on full Grain-128a, Grain-128, and Grain-v1. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 129–159. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_5
77. Watanabe, D., Biryukov, A., De Cannière, C.: A distinguishing attack of SNOW 2.0 with linear masking method. In: Matsui, M., Zuccherato, R.J. (eds.) SAC 2003. LNCS, vol. 3006, pp. 222–233. Springer, Berlin, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-24654-1_16
78. Xiao, G., Massey, J.L.: A spectral characterization of correlation-immune combining functions. IEEE Trans. Inf. Theory **34**(3), 569–571 (1988). https://doi.org/10.1109/18.6037
79. Zama: TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data (2022), https://github.com/zama-ai/tfhe-rs

# A Complexity of Correlation Attacks over $\mathbb{F}_p$

## A.1 Bound on the Correlation of Linear Trails

**Proposition 2.** *Let us consider an $n$-round linear trail of* `Transistor`*, for $n \geq 4$, defined by masks $\alpha \in \mathbb{F}_p^{16(n-1)}$ and $\beta \in \mathbb{F}_p^{4n}$, $\beta \neq 0$. Then, its correlation, defined by*

$$C^{(n)}(\alpha, \beta) = p^{-16n} \widehat{F}^{(n)}(0, \alpha, \beta),$$

*satisfies*

$$\left| C^{(n)}(\alpha, \beta) \right|^2 \leq \left( \frac{\mathcal{L}(\pi)}{p} \right)^{2w_n},$$

*where $\mathcal{L}(\pi)$ is the maximal modulus of the Walsh coefficients of $\pi$, and $w_n = \sum_{i=1}^{n-1} wt(\alpha_i)$ where $wt(\cdot)$ denotes the number of nonzero digits of a vector with coordinates in $\mathbb{F}_p$.*

*Proof.* For $n$ rounds, the Walsh coefficient of the augmented function, $\widehat{F}^{(n)}(0, \alpha, \beta)$, corresponds to a Walsh coefficient of the following function

$$\begin{aligned} H^{(n)} : \quad \mathbb{F}_p^{16} \times \mathbb{F}_p^{16(n-1)} &\to \mathbb{F}_p^{4(n-1)} \times \mathbb{F}_p^{16} \\ (X_0, K_1, \ldots, K_{n-1}) &\mapsto (S_0, \ldots, S_{n-2}, X_{n-1}) \ . \end{aligned}$$

In other words, the output of $H^{(n)}$ corresponds (up to a reordering of the 16 last digits) to the concatenation of the output of the augmented function $F^{(n)}$ and of the remaining 12 internal digits that are not output by $\phi$. Then, $H^{(n+1)}$ can be seen as the composition of $H^{(n)}$ with the round function

$$\begin{aligned} &\mathbb{F}_p^{16} \times \mathbb{F}_p^{16} \to \mathbb{F}_p^4 \times \mathbb{F}_p^{16} \\ &(X_{n-1}, K_n) \mapsto (\phi(X_{n-1}), \mathsf{SD}(L(X_{n-1}) + K_n)) \ . \end{aligned}$$

Let $G$ be any function of the form

$$\begin{aligned} G : \mathbb{F}_p^{\ell} &\to \mathbb{F}_p^{k} \times \mathbb{F}_p^{16} \\ U &\mapsto (G_1(U), G_2(U)) \ . \end{aligned}$$

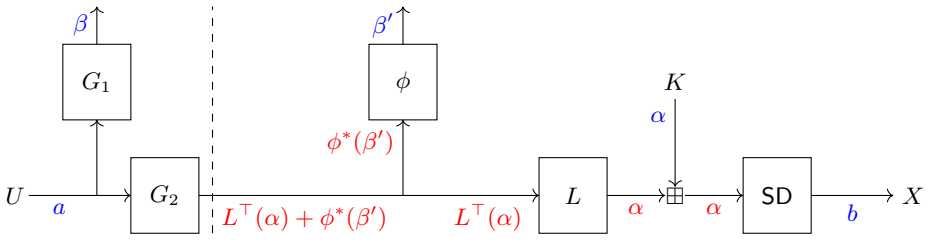



**Fig. 10:** Masks propagation through a composition with `Transistor`'s round function. Input and output masks are written in blue while inner masks whose values are deduced are written in red.

Then, the Walsh transform of the composition

$$F : (U, K) \mapsto (G_1(U), \phi(G_2(U)), \mathsf{SD}(L(G_2(U)) + K))$$

can be easily derived from the Walsh transform of $G$, as shown on Figure 10. Indeed, the detailed computation of the Walsh coefficient

$$\mathcal{I} := \widehat{F}(a, \alpha; \beta, \beta', b)$$

for the input mask $(a, \alpha)$ and output mask $(\beta, \beta', b)$ is as follows.

$$\begin{aligned}
\mathcal{I} &= \sum_{U \in \mathbb{F}_p^\ell, K \in \mathbb{F}_p^{16}} \omega^{\beta \cdot G_1(U) + \beta' \cdot \phi(G_2(U)) + b \cdot \mathsf{SD}(L(G_2(U)) + K) - \alpha \cdot K - a \cdot U} \\
&= \sum_{U \in \mathbb{F}_p^\ell} \omega^{\beta \cdot G_1(U) + \beta' \cdot \phi(G_2(U)) - a \cdot U} \left( \sum_{Z \in \mathbb{F}_p^{16}} \omega^{b \cdot \mathsf{SD}(Z) - \alpha \cdot Z + \alpha \cdot L(G_2(U))} \right)
\end{aligned}$$

where we set $Z = L(G_2(U)) + K$. We then deduce

$$\begin{aligned}
\mathcal{I} &= \sum_{U \in \mathbb{F}_p^\ell} \omega^{\beta \cdot G_1(U) + \beta' \cdot \phi(G_2(U)) - a \cdot U + \alpha \cdot L(G_2(U))} \widehat{\mathsf{SD}}(\alpha, b) \\
&= \widehat{\mathsf{SD}}(\alpha, b) \sum_{U \in \mathbb{F}_p^\ell} \omega^{\beta \cdot G_1(U) + \phi^*(\beta') \cdot G_2(U) - a \cdot U + L^T(\alpha) \cdot G_2(U)} \\
&= \widehat{\mathsf{SD}}(\alpha, b) \widehat{G}(a; \beta, L^T(\alpha) + \phi^*(\beta'))
\end{aligned}$$

where $\phi^* : \mathbb{F}_p^4 \to \mathbb{F}_p^{16}$ is the function outputting an internal state whose digits are all zero, expect the digits affected by $\phi$, which are equal to the inputs. Finally, we observe that $H^{(1)}$ is the identity function implying that $\widehat{H}^{(1)}(a, b) = p^{16}$ if $a = b$ and 0 otherwise. It follows that the Walsh coefficients of $H^{(n)}$ are either zero, or given by

$$\widehat{H}^{(n)}(a, \alpha_1, \ldots \alpha_{n-1}; \beta_0, \ldots, \beta_{n-2}, b) = p^{16} \prod_{i=1}^{n-1} \widehat{\mathsf{SD}}(\alpha_i, b'_i) \ ,$$

for some $b'_1, \ldots, b'_{n-1}$. Therefore,

$$p^{-16n} \widehat{H}^{(n)}(a, \alpha_1, \ldots, \alpha_{n-1}; \beta_0, \ldots, \beta_{n-2}, b) = \prod_{i=1}^{n-1} \frac{\widehat{\mathsf{SD}}(\alpha_i, b'_i)}{p^{16}} \ .$$

The result then directly follows by observing that $\widehat{\mathsf{SD}}(\alpha_i, b'_i)$ is the product of the Walsh coefficients of the 16 S-boxes composing $\mathsf{SD}$. □

### A.2 Truncated Linear Trails from MILP

In order to find a lower bound for the number of S-boxes active in a linear trail over 4 rounds of `Transistor`, we apply the approach introduced by Mouha *et*

*al.* [65] in the most direct way. As we are interested in the (in)activity of the Sboxes throughout the rounds, we only need to assign binary variables to output digits, but also to internal digits of the state before the Sbox layer, and before the MixColumn operation. For each round, we then have $4+16+16 = 36$ binary variables that are related one to others by the following constraints.

**3-fork constraint.** Any output digit corresponds to a digit of the internal state after the Sbox layer, so it is naturally related to the activity of a digit before MixColumn, by taking care of the reorganization of the digits through ShiftRows. But because the activity pattern is unchanged through the Sbox layer, it is also related to a digit before the Sbox layer. The constraint between three such variables is that if one is active, then at least two of them are. This corresponds to a 3-fork constraint that can be modeled as in [65, Sec 2.2].

**MDS constraint.** A given column before and after MixColumns are related by the following MDS constraint: if a digit is active, then at least 5 of them are. This can be modeled in a manner similar to the 3-fork. In our case, the binary variables associated to the state after MixColumns at round $r$ are the one corresponding to the state before the Sbox layer at round $r+1$.

**Border constraints.** We also add some border constraints to ensure that the initial inner state is fully inactive, and that the same holds for the final inner state. This way, we make sure that the considered linear equations do not depend on the unknown FSM state, but only on the key and output digits. We also impose that at least a digit among the ones output at the first round, and at least one among the ones of the last round must be active.

Finally, with the described constraints, the objective is to minimize the number of active Sboxes, that is, the number of active digits before the Sbox layer. Note that any solution to this problem is actually a worst-case scenario in our case: a returned activation pattern is not guaranteed to be actually instantiable. We solved this simple MILP model using the SageMath interface for Mixed Integer Linear Programing solving within seconds on a standard laptop. Our code is available online.[4]

Most notably, we have found that

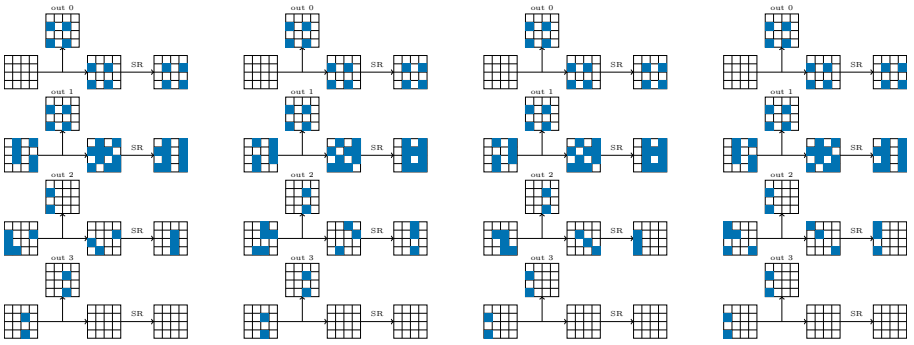
$$w_4 \geq 13,\ w_5 \geq 20 \text{ and } w_6 \geq 25$$

with the potential trail examples depicted on Fig. 11. We also verified that $w_n \geq 26$ for $n \in \{7, 8, \dots 26\}$. For larger values of $n$, either a trail has at least one active Sbox per round, or it splits into two smaller trails with at least 13 active Sboxes. Therefore, we deduce that $w_n \geq 26$ for all $n \geq 7$.

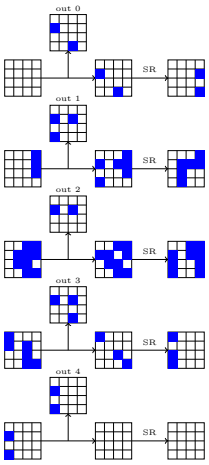
### A.3 Data Complexity of Fast Correlation Attacks

It is well-known that the capacity of the output distribution of a linear relation determines the minimal length of the sequence obtained by a given linear combination of the digits of $(S_t)_{t\in\mathbb{N}}$ that is required for recovering the initial state

[4] https://github.com/CryptoExperts/Transistor/




**(a)** 4-round trail. **(b)** 4-round trail. **(c)** 4-round trail. **(d)** 4-round trail.




**(e)** 5-round trail.

**Fig. 11:** Activity patterns for linear trails over 4 and 5 rounds.

of the key-LFSR from the linear relation

$$\sum_{i=1}^{n-1} \alpha_i \cdot K_{t+i} + \sum_{i=0}^{n-1} \beta_i \cdot S_{t+i}, \forall t \geq 0 \; .$$

However, the same result holds even when the relation is not linear as stated in the following theorem.

**Theorem 3.** *Let $F$ be a function from $\mathbb{F}_p^\kappa \times \mathbb{F}_p^m$ to $\mathbb{F}_p^n$. Let $g : \mathbb{F}_p^\kappa \to \mathbb{F}_p$ and $h : \mathbb{F}_p^n \to \mathbb{F}_p$ such that the probability distribution of*

$$(U, V) \mapsto h(F(U, V)) - g(U)$$

*is close to the uniform distribution,* i.e.*, for all $z \in \mathbb{F}_p$,*

$$\Pr_{(U,V) \xleftarrow{\$} \mathbb{F}_p^\kappa \times \mathbb{F}_p^m} [h(F(U, V)) - g(U) = z] = \frac{1}{p} + \varepsilon_z \text{ with } \varepsilon_z \ll 1 \; .$$

*Let $(U_t)_{t\in\mathbb{N}}$ be a sequence of elements in $\mathbb{F}_p^\kappa$ defined by $U_{t+1} = \Phi(U_t)$, where $\Phi$ is a function from $\mathbb{F}_p^\kappa$ to itself. Let $(V_t)_{t\in\mathbb{N}}$ be a sequence of elements in $\mathbb{F}_p^m$. Then, the minimal length $N$ of the sequence $(B_t)_{t\in\mathbb{N}} := (h(F(U_t, V_t))_{t\in\mathbb{N}}$ required for recovering $U_0$ is*

$$N = \frac{\kappa \ln p}{\Delta}$$

*with*

$$\Delta = p \sum_{y\in\mathbb{F}_p} \varepsilon_y^2 = \sum_{a\in\mathbb{F}_p^*} \left| p^{-\kappa} \sum_{U\in\mathbb{F}_p^\kappa, V\in\mathbb{F}_p^m} \omega^{a(h(F(U,V))-g(U))} \right|^2 .$$

*Proof.* Let

$$\mathcal{C} = \{(g(\Phi^t(U_0)))_{0\le t<N}, U_0 \in \mathbb{F}_p^\kappa\} .$$

This set is a (non-linear) code over $\mathbb{F}_p$ of length $N$ and size $p^\kappa$. As originally observed by Meier and Staffelbach [64], recovering $U_0$ from $(B_0, \ldots, B_{N-1})$ boils down to decoding this code. Indeed, $(B_0, \ldots, B_{N-1})$ can be seen as the result of the transmission of the $N$-digit word $(g(U_0), \ldots, g(\Phi^{N-1}(U_0)))$ through a noisy transmission channel. This transmission channel is memoryless, since each digit is affected in the same way:

$$B_t = g(\Phi^t(X_0)) + \eta_t$$

where the probability distribution of all $\eta_t$ is given by

$$\pi_z := \Pr[\eta_t = z] = \Pr_{(U,V)\xleftarrow{\$}\mathbb{F}_p^\kappa\times\mathbb{F}_p^m}[h(F(U,V)) - g(U) = z] = \frac{1}{p} + \varepsilon_z .$$

This transmission channel is called symmetric in the sense that its transition matrix, formed by the probabilities that an input value $i$ is transformed to $j$, is a circulant matrix whose rows correspond to the probability distribution $(\pi_0, \ldots, \pi_{p-1})$. Therefore, Shannon's channel-coding theorem [38, Section 7.7] implies that this code can be decoded with a non-negligeable success probability if and only if its rate, *i.e.*, the ratio between the logarithm of its size and its length, is smaller than the channel capacity:

$$\frac{\kappa}{N} \le C_{\text{channel}} .$$

The capacity of any $p$-ary symmetric channel is given by [38, Th. 7.2.1]

$$C_{\text{channel}} = 1 - H_p(\pi_0, \pi_1, \ldots, \pi_{p-1})$$

where $H_p$ denotes the $p$-ary entropy, i.e.,

$$H_p(\pi_0, \pi_1, \ldots, \pi_{p-1}) := -\sum_{z\in\mathbb{F}_p} \pi_z \log_p(\pi_z) .$$

We use that, for $\varepsilon \ll 1$,

$$\ln\left(\varepsilon + \frac{1}{p}\right) = \ln(1 + p\varepsilon) - \ln p = p\varepsilon - \ln p + o(\varepsilon) .$$

It follows that

$$\sum_{z\in\mathbb{F}_p}\left(\varepsilon_z+\frac{1}{p}\right)\ln\left(\varepsilon_z+\frac{1}{p}\right)\simeq p\sum_{z\in\mathbb{F}_p}\varepsilon_z^2-\ln p\sum_{z\in\mathbb{F}_p}\left(\varepsilon_z+\frac{1}{p}\right)\ .$$

We then deduce that

$$C_{\text{channel}}=1+\frac{1}{\ln p}\left(p\sum_{z\in\mathbb{F}_p}\varepsilon_z^2-\ln p\right)=\frac{1}{\ln p}\left(p\sum_{z\in\mathbb{F}_p}\varepsilon_z^2\right)\ .$$

It is well-known (see Prop. 1) that the same quantity can also be derived from the Walsh transform of the function

$$f:(U,V)\mapsto h(F(U,V))-g(U)\ .$$

Indeed, if $\pi'$ denotes the function from $\mathbb{F}_p$ to $\mathbb{R}$ defined by $\pi'(x)=\pi_x-\frac{1}{p}=\varepsilon_x$, we have

$$\Delta=p\sum_{y\in\mathbb{F}_p}|\pi'(y)|^2=\sum_{a\in\mathbb{F}_p}|\widehat{\pi}'(a)|^2$$

where the last equality corresponds to Plancherel's formula. Moreover, the Walsh transform of $\pi'$ an be computed as follows: if $a\neq 0$,

$$\begin{aligned}\widehat{\pi}'(a)&=\sum_{x\in\mathbb{F}_p}\pi'(x)\omega^{-ax}=\sum_{x\in\mathbb{F}_p}\pi_x\omega^{-ax}-p^{-1}\sum_{x\in\mathbb{F}_p}\omega^{-ax}\\&=p^{-(\kappa+m)}\sum_{x\in\mathbb{F}_p}\#f^{-1}(x)\omega^{-ax}=p^{-(\kappa+m)}\sum_{U,V\in\mathbb{F}_p^\kappa\times\mathbb{F}_p^m}\omega^{-af(U,V)}\\&=p^{-(\kappa+m)}\widehat{f}(0,-a)\ ,\end{aligned}$$

and $\widehat{\pi}'(0)=0$. It follows that

$$\Delta=\sum_{a\in\mathbb{F}_p^*}\left|p^{-(\kappa+m)}\widehat{f}(0,a)\right|^2\ .$$

□

# B More Resources About TFHE

## B.1 Complexity assumptions

Here, we define the assumptions on which the security of TFHE relies.

**Definition 3.** *(LWE problem over the discretized torus). Let $q,n\in\mathbb{N}$ and let $\boldsymbol{s}=(s_1,\ldots,s_n)\xleftarrow{\$}\mathbb{B}^n$. Let $\chi$ be an error distribution over $\mathbb{Z}_q$. The* decisional

Learning With Errors over discretized torus problem *is to distinguish between samples drawn from the distributions:*

$$\mathcal{D}_0 = \{(\boldsymbol{a}, r) \mid \boldsymbol{a} \xleftarrow{\$} \mathbb{T}_q^n, r \xleftarrow{\$} \mathbb{T}_q\}$$

*and:*

$$\mathcal{D}_1 = \{(\boldsymbol{a}, b) \mid \boldsymbol{a} = (a_1, \ldots, a_n) \xleftarrow{\$} \mathbb{T}_q^n, e \xleftarrow{\$} \chi, b = \sum_{j=1}^{n} a_j \cdot s_j + e\}.$$

*The* search *version of the problem is to recover* $\boldsymbol{s}$ *from samples of* $\mathcal{D}_1$*.*

Both the search and decisional problems are reducible to each other [70], and their average case is as hard as the worst-case lattice problems.

TFHE also relies on the generalized version of LWE over rings, introduced in [19], known as GLWE.

**Definition 4.** *(GLWE problem over the discretized torus). Let* $N, q, k \in \mathbb{N}$ *with* $N$ *a power of two and let* $\boldsymbol{s} = (s_1, \ldots, s_k) \xleftarrow{\$} \mathbb{B}_N[X]^k$*. Let* $\chi$ *be an error distribution over* $\mathbb{Z}_{N,q}[X]$*. The* General decisional Learning With Errors over discretized torus problem *is to distinguish between samples drawn from the following distributions*

$$\mathcal{D}_0 = \{(\boldsymbol{a}, r) \mid \boldsymbol{a} \xleftarrow{\$} \mathbb{T}_{N,q}[X]^k, r \xleftarrow{\$} \mathbb{T}_{N,q}[X]\}$$

*and:*

$$\mathcal{D}_1 = \{(\boldsymbol{a}, b) \mid \boldsymbol{a} = (a_1, \ldots, a_k) \xleftarrow{\$} \mathbb{T}_{N,q}[X]^k, e \xleftarrow{\$} \chi, b = \sum_{j=1}^{k} a_j \cdot s_j + e\}.$$

*The* search *version is analogous to the LWE one.*

The complexity analysis is analogous to the LWE version. In practice, the error distribution $\chi$ is a centered Gaussian distribution parametrized by its standard deviation $\sigma$.

### B.2 Analysis of the variances inside a PBS

We recall that the PBS is composed of four successive operations: Keyswitch, ModulusSwitch, BlindRotate and SampleExtract.

The critical variance is the variance in input of the BlindRotate (BR). If the noise at this point of the algorithm is too high, the PBS will output the encryption of a wrong value with high probability. This critical variance satisfies:

$$\sigma^2_{\text{in-BR}} = \sigma^2_{\text{in-PBS}} + \sigma^2_{\text{KS}} + \sigma^2_{\text{MS}}$$

where $\sigma^2_{\text{in-PBS}}$ is the variance in input of the PBS, which according to Section 4.4 satisfies:

$$\sigma^2_{\text{in-PBS}} = \sigma^2_{\mathcal{K}} + \sigma^2_{\text{MC}} = |\mathcal{K}| \cdot \left(\frac{p-1}{2}\right)^2 \cdot \sigma^2_{\text{long}} + L^2_{\text{MC}} \cdot \sigma^2_{\text{PBS}} \ ,$$

and where $\sigma^2_{\mathsf{KS}}$ and $\sigma^2_{\mathsf{MS}}$ are additive terms introduced by the KeySwitch and the ModSwitch respectively and which depend on the internal parameters of the PBS. The former is defined by:

$$\sigma^2_{\mathsf{KS}} = n_{\text{long}} \cdot \left( \frac{q^2}{12 B_{\text{KS}}^{2\ell_{\text{KS}}}} - \frac{1}{12} \right) \cdot \left( \text{Var}(s_i) + \mathbb{E}^2(s_i) \right) + \frac{n_{\text{long}}}{4} \cdot \text{Var}(s_i) + n_{\text{long}} \cdot \ell_{\mathsf{KS}} \cdot \sigma^2_{\mathsf{KSK}} \cdot \frac{B_{\text{KS}} + 2}{12}$$

where $s_i$ refers to the secret key's bits used for encryption, and $\sigma_{\mathsf{KSK}}$ is the noise introduced in the key-switching keys (so $\sigma_{\text{short}}$ in our case).

For $\sigma_{\mathsf{MS}}$, we have:

$$\sigma^2_{\mathsf{MS}} = \frac{q^2}{4N^2} \cdot \left( \frac{1}{12} - \frac{4N^2}{12q^2} + \frac{n_{\text{short}}}{24} + \frac{n_{\text{short}} \cdot 4N^2}{48q^2} \right)$$

The full noise analysis leading to these formulas can be found in [75].

## C Further Compressing the TFHE Ciphertexts

We introduce hereafter a tweak to compress a TFHE encryption further than the folklore compression. By definition of the TFHE encryption process, the least significant bits of the body $b_i = \sum_{j=1}^{n} a_{i,j} \cdot s_j + \tilde{m}_i + e_i$ are randomized by the error $e_i$ and can hence be discarded without loss of information. We can thus tweak the above compressed encryption process by returning $(\mathsf{seed}, \text{Tr}_\ell(b_1), \ldots, \text{Tr}_\ell(b_t))$ where $\text{Tr}_\ell(\cdot)$ denotes the truncation of the $\ell$ least significant bits. To decompress such ciphertexts, besides pseudorandomly generating the masks from the seed, one just needs to pad the truncated bodies with $\ell$ bits to 0. By the randomness of the mask, the effect of this truncation plus 0-padding is to add a uniform random error of $\ell$ bits to the body, namely an error of standard deviation:

$$\sigma_0^2 = \frac{(2^\ell - 1)^2}{12} \approx \frac{2^{2\ell}}{12} \approx 0.08 \cdot 2^{2\ell} \ .$$

This optimization comes in two flavors:

1. *The "free" variant.* The number of truncated bits $\ell$ is selected to have a small impact on the noise distribution. For instance in `Transistor`, the noise of the fresh ciphertexts is summed with the noise coming from the FSM. Thus, we can compute $\ell$ to keep $\sigma^2_{\mathcal{W}} < \sigma^2_{\text{PBS}}$. Our experiments shows $\sigma^2_{\text{PBS}} = 2^{52}$, so running the numbers we find that we can truncate up to $\ell = 19$ bits, allowing to reduce the volume of the TFHE ciphertexts to send by a factor $1 - \frac{19}{64} \approx 0.7$.
2. *The communication-computation trade-off.* In this variant, one selects a high value of $\ell$. The truncated body should at least contain $\log_2(p)$ bits to keep the plaintext information, plus a margin of a few bits in order to remain bootstrappable. Denoting this margin $\delta$, the truncated body should be of at

least $\log_2(p) + \delta$ bits and $\ell$ can be up to $\log_2(q) - (\log_2(p) + \delta)$. Taking the maximum level of truncation, inducing the maximum level of bootstrappable noise, implies some adaptation of the underlying homomorphic computation. Specifically, it should start with applying a noise-reduction bootstrapping to the decompressed ciphertexts before performing the original evaluation. We hence obtain a trade-off with reduced bandwidth against additional bootstrappings.

In the context of transciphering with `Transistor`, the trade-off provided by the second option gives rise to an initialization procedure which consists in decompressing and bootstrapping the wrapped key.

# D Additionnal Cryptanalysis

## D.1 Algebraic Analysis

Algebraic cryptanalysis consists of formulating nonlinear equations that an attacker can derive from the information observed in terms of the secret key material. Several techniques, such as Gröbner basis methods or linearization, exist for solving such systems of equations, and we discuss them in this section. Based on this analysis, we claim that `Transistor` is resistant to algebraic attacks and their improvements.

*Gröbner Basis.* Such an attack consists of four main steps: formulating the equations that model the intended cryptanalysis, computing their Gröbner basis, applying a "change of monomial ordering" to transform the Gröbner basis into a more useful form, and finally solving the result using univariate techniques. The complexity of the first and last steps is usually negligible, meaning that we should evaluate the time complexity of at least one of the other two steps.

However, as recently shown in [10], it is possible to write the equations in such a way as to entirely bypass the computation of the Gröbner basis. This is achieved by choosing a custom *monomial ordering* that ensures the equations, as formulated, immediately form a Gröbner basis. This approach can be applied here by assigning increasing weights to the successive outputs of the key schedule, so that the leading monomials in each equation involve only a single key variable. This method is effective for at least the first four clock cycles, as the clock outputs are independent.

At this stage, we have 16 independent equations of degree 15 (the degree of $\pi$). Adding the equations corresponding to the next 20 clocks, we get as many equations as unknowns, which, in particular, should lead to a 0-dimensional ideal. As conjectured in [69], the ideal degree of this system can be lower-bounded by $15^{16} \approx 2^{62.5}$. This bound would be exact if the 80 remaining equations somehow failed to contribute to an increase in this quantity, or if the ideal degree was in some way decreased by one of these equations (despite the 0-dimension). Given that change of order algorithms are at least quadratic in the ideal degree, we can safely claim security against Gröbner-basis-based algebraic attacks.

*Linearization.* Such attacks may, a priori, pose a threat, as they have led to the downfall of two FHE-friendly stream ciphers, namely `FLIP` [62] and `Elisabeth` [35], which were broken in [44] and [52], respectively. The structure of these ciphers made such attacks an inherent risk: in both cases, a low-degree function is applied to a constant key register to generate keystream words. As a result, in these ciphers, the nonlinear equations derived from the keystream sequence have a constant degree. Considering all monomials (or linear combinations of monomials) as new independent variables in this representation enables powerful attacks [44,52]. These attacks are the result of two fundamental weaknesses: the constant degree of the system and low diffusion, as the registers are never updated, only a bit-permutation is applied at each clock cycle.

These issues are directly addressed in the design of `Transistor`. First, the round function of the FSM applies the S-box $\pi$ to every digit. This S-box has a univariate representation in $\mathbb{F}_p$ that is both dense and of degree 15. Furthermore, the content of the FSM accumulates high-degree equations within the key-LFSR. As a result, the multivariate polynomial representation of the keystream digits will not have a constant degree; instead, it will be very dense and of high degree.

*Using Annihilators.* Another powerful technique is to directly use an annihilator of the filtering function, where this annihilator has a lower degree than the original function [37,36]. This allows the attacker to collect and solve a system of equations with a smaller degree than the original one. That is, similar to Gröbner basis-like attacks, this approach operates on the ideal generated by the polynomials. In the case of `Transistor`, we can argue and defend the role of the whitening LFSR $\mathcal{W}$. Indeed, this LFSR has a length of 32 digits, meaning that an annihilator at the output must consider the sum of 8 outputs by $\phi$ and cancel them with a polynomial. Therefore, without additional information, such an annihilator would need to multiply approximately 32 digits, leading to an increase in the degree. This strategy must also account for the degree increase in successive outputs of $\phi$, as discussed above.

*Other Techniques.* Last but not least, algebraic attacks can be improved in several ways using the Guess-and-Determine strategy or the so-called Hybrid approach in Gröbner basis computations. In our case, one could guess key-register cells, whitening-key cells, or cells in the FSM. Although this is a valid approach, the remaining equations (depending on the guessing strategy) would either have an increased degree or require guessing too many cells, making the attack impractical.

### D.2 Comparison With `LEX`

`LEX` [15] is a stream cipher designed by Biryukov in 2006 and selected to the third phase of the eSTREAM competition. `LEX` employed a rather unusual design for a stream cipher, based on the `AES` block cipher and a technique called *leak extraction*. The idea of the leak extraction is to produce the key stream by extracting parts of the underlying block cipher state.

The description of LEX is very simple and elegant. It is based on a slightly tweaked version of the AES where the AddRoundKey operation before the first round is omitted and where the MixColumns of the last round is not. For simplicity, we will still refer to this tweaked version as AES. First, the publicly known $IV$ is encrypted by the AES under the secret key $K$ to produce an initial state $S = \texttt{AES}_K(IV)$. Then, the state $S$ is repeatedly encrypted using the OFB mode and the same secret key $K$. At each round of encryption, four words of the internal state are extracted to compose the key stream produced by LEX. The positions of the extracted words depend on the round number and are depicted in Figure 12.

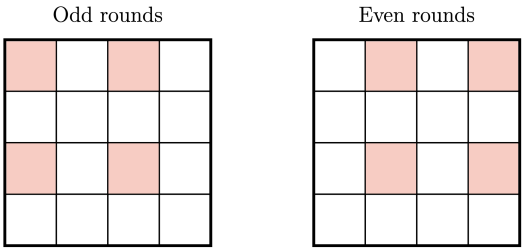



**Fig. 12:** Extraction of internal state words for odd and even rounds of LEX. The extracted words are the ones being colored.

In 2008, Dunkelman and Keller presented an attack against LEX [43] able to recover the 128-bit secret key with $2^{36.3}$ bytes of key-stream produced by the same key and a time complexity of $2^{112}$ simple operations. This attack worked by exploiting a particular difference pattern of probability $2^{-64}$ in the AES internal state that could be detected by observing a 32-bit condition in the output key stream. The attack can be decomposed in the three following steps:

1. The attacker observes the output stream for a specific 32-bit pattern to occur (the four output words at a certain round should all be zero). This potentially indicates a special difference pattern (8 particular words with zero-difference) in the internal state of AES.
2. Once this difference pattern is detected, the attacker recovers the values of 16 bytes of the internal state in both AES encryptions. This is achieved by guessing the difference in eight additional state words and by exploiting simple properties of the MixColumns and the S-box.
3. Finally, using the recovered 16 bytes, the attacker proceeds with a guess-and-determine approach to retrieve the secret key. The key step here is exploiting relations derived from the AES-128 key schedule that link bytes from three consecutive subkeys, reducing the number of required guesses to just two subkey bytes. This limits the guessing process to only 10 bytes (80 bits) in total.

`Transistor`'s structure ressembles `LEX` in several aspects, the most notable being the extraction of four words at each iteration. Additionally, `Transistor`'s round function is inspired by the `AES` round function. For these reasons, it is natural to question whether the attack described in [43] could be adapted to `Transistor`. However, as we will argue next, `Transistor` differs from `LEX` in some crucial design choices, making the Dunkelman and Keller attack very difficult to apply:

- The output of the FSM at every round is masked by the whitening LFSR $\mathcal{W}$ making it hard to directly recover the values of the internal state as done in the attack of `LEX`.
- The LFSR $\mathcal{K}$ playing the role of the key schedule, produces uncorrelated outputs, making it hard to find simple relations between key values in consecutive rounds. A crucial element in the success of the attack against `LEX` was exactly the fact that by guessing only two key bytes, the attacker was able to recover many more key values by exploiting such relations holding over several rounds. As we showed in the previous sections, it is not possible for an attacker to extract any information on the secret key by observing the output of the FSM over 3 or 4 consecutive rounds.