# How to Compress Homomorphic Ciphertexts

Anne Canteaut[1], Sergiu Carpov[2], Caroline Fontaine[3*], Tancrède Lepoint[4**], María Naya-Plasencia[1], Pascal Paillier[4**], and Renaud Sirdey[2]

[1] Inria, France, {anne.canteaut,maria.naya_plasencia}@inria.fr
[2] CEA LIST, France, {sergiu.carpov,renaud.sirdey}@cea.fr
[3] CNRS/Lab-STICC and Telecom Bretagne and UEB, caroline.Fontaine@telecom-bretagne.eu
[4] CryptoExperts, France, {tancrede.lepoint,pascal.paillier}@cryptoexperts.com

**Abstract.** In typical applications of homomorphic encryption, the first step consists for Alice to encrypt some plaintext $m$ under Bob's public key $\mathsf{pk}$ and to send the ciphertext $c = \mathsf{HE}_{\mathsf{pk}}(m)$ to some third-party evaluator Charlie. This paper specifically considers that first step, *i.e.* the problem of transmitting $c$ as efficiently as possible from Alice to Charlie. As previously noted, a form of compression is achieved using hybrid encryption. Given a symmetric encryption scheme $\mathsf{E}$, Alice picks a random key $k$ and sends a much smaller ciphertext $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ that Charlie decompresses homomorphically into the original $c$ using a decryption circuit $\mathcal{C}_{\mathsf{E}^{-1}}$.

In this paper, we revisit that paradigm in light of its concrete implementation constraints; in particular $\mathsf{E}$ is chosen to be an additive IV-based stream cipher. We propose 2 new designs such that $\mathcal{C}_{\mathsf{E}^{-1}}$ has very small multiplicative depth, typically between 8 and 12 for 128-bit security. Our first construction of depth 12 is inspired by Trivium and reportedly the current fastest option. Our second construction, based on exponentiation in binary fields, is impractical but sets the lowest depth record to 8 for 128-bit security, and more generally to a neat $\lceil \log \kappa \rceil + 1$ for $\kappa$-bit security.

## 1 Introduction

Since the breakthrough result of Gentry [Gen09] achieving fully homomorphic encryption (FHE), many works have been published on simpler and more efficient schemes based on homomorphic encryption. Because they allow arbitrary computations on encrypted data, FHE schemes suddenly opened the way to exciting new applications, in particular cloud-based services in several areas (see *e.g.* [NLV11,GLN12,LLN14]).

**Compressed encryption.** In these cloud applications, it is often assumed that some data is sent encrypted under a homomorphic encryption (HE) scheme to the cloud to be processed in a way or another. It is thus typical to consider, in the first step of these applications, that a user (Alice) encrypts some data $m$ under some other user's public key $\mathsf{pk}$ (Bob) and sends some homomorphic ciphertext $c = \mathsf{HE}_{\mathsf{pk}}(m)$ to a third-party evaluator in the Cloud (Charlie). The roles of Alice and Bob are clearly distinct, even though they might be played by the same entity in some applications.

However, all HE schemes proposed so far suffer from a very large ciphertext expansion; the transmission of $c$ between Alice and Charlie is therefore a very significant bottleneck in practice. The problem of reducing the size of $c$ as efficiently as possible has first been considered in [NLV11] wherein $m$ is encrypted with a symmetric encryption scheme $\mathsf{E}$ under some key $k$ randomly chosen

[*] This work has received a French governmental support granted to the COMIN Labs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.

[**] This work has been supported in part by the European Union's H2020 Programme under grant agreement number ICT-644209.

by Alice, who then sends a much smaller ciphertext $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ to Charlie. Given $c'$, Charlie then exploits the homomorphic property of $\mathsf{HE}$ and recovers the original

$$c = \mathsf{HE}_{\mathsf{pk}}(m) = \mathcal{C}_{\mathsf{E}^{-1}}\left(\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m)\right)$$

by homomorphically evaluating the decryption circuit $\mathcal{C}_{\mathsf{E}^{-1}}$. This can be assimilated to a *compression method* for homomorphic ciphertexts, $c'$ being the result of applying a *compressed encryption scheme* to the plaintext $m$ and $c$ being recovered from $c'$ using a *ciphertext decompression procedure*. In that approach obviously, the new encryption rate $|c'|/|m|$ becomes asymptotically close to 1 for long messages, which leaves no significant margin for improvement. However, the paradigm of ciphertext compression leaves totally open the question of how to choose $\mathsf{E}$ in a way that minimizes the decompression overhead, while preserving the same security level as originally intended.

**Prior art.** The cost of a homomorphic evaluation of several symmetric primitives has been investigated, including several optimized implementations of AES [GHS12,CCK$^+$13,DHS14], and of the lightweight block ciphers Simon [LN14] and Prince [DSES14]. Usually very simple, lightweight block ciphers seem natural candidates for efficient evaluations in the encrypted domain. However, they may also lead to *much worse* performances than a homomorphic evaluation of, say, AES. Indeed, contemporary HE schemes use *noisy* ciphertexts, where a fresh ciphertext includes a noise component which grows along with homomorphic operations. Usually a homomorphic multiplication increases the noise by much larger proportions than a homomorphic addition. The maximum allowable level of noise (determined by the system parameters) then depends mostly on the multiplicative depth of the circuit. Many lightweight block ciphers balance out their simplicity by a large number of rounds, *e.g.* KATAN and KTANTAN [CDK09], with the effect of considerably increasing their multiplicative depth. This type of design is therefore prohibitive in a HE context. Still Prince appears to be a much more suitable block cipher for homomorphic evaluation than AES (and than Simon), because it specifically targets applications that require a low latency; it is designed to minimize the cost of an unrolled implementation [BCG$^+$12] rather than being designed to optimize *e.g.* silicon area.

As recently observed by Albrecht, Rechberger, Schneider, Tiessen and Zohner in [ARS$^+$15], the usual criteria that rule out the design of lightweight block ciphers are not appropriate when designing a symmetric encryption scheme with a low-cost homomorphic evaluation. Indeed, both the number of rounds and the number of binary multiplications required to evaluate a Sbox have to be taken into account. Minimizing the number of rounds is a crucial issue for low-latency ciphers like Prince, while minimizing the number of multiplications is a requirement when designing a block cipher for efficient masked implementations (see *e.g.* [GLSV14]).

These two criteria have been considered together for the first time by Albrecht *et al.* in the recent design of a family of block ciphers called LowMC [ARS$^+$15] with very small multiplicative size and depth.[5]

**Our contributions.** We emphasize that beyond the task of designing a HE-friendly block cipher, revisiting the whole compressed encryption scheme (in particular its internal mode of operation) is what is really needed in order to take these concrete HE-related implementation constraints into account.

[5] It is worth noting that in a HE context, reducing the multiplicative size of a symmetric primitive might not be the first concern (although it is critical in a multiparty computation context, which also motivated the work of Albrecht *et al.* [ARS$^+$15]), whereas minimizing the multiplicative depth is of prime importance.

First, we identify that homomorphic decompression is subject to an *offline phase* and an *online phase*. The offline phase is plaintext-independent and therefore can be performed in advance, whereas the online phase completes decompression upon reception of the plaintext-dependent part of the compressed ciphertext. Making the online phase as quick as technically doable leads us to choose an additive IV-based stream cipher to implement $\mathsf{E}$. However, we note that the use of a lightweight block cipher as the building-block of that stream cipher usually provides a security level limited to $2^{n/2}$ where $n$ is the block size [Rog11], thus limiting the number of encrypted blocks to (typically) less than $2^{32}$ (*i.e.* 32GB for 64-bit blocks).

As a result, we propose our own candidates for $\mathsf{E}$. Our first candidate is the keystream generator Trivium [CP08], which belongs to the eSTREAM portfolio of recommended stream ciphers, and a new proposal called *Kreyvium*, which shares the same internal structure. The main advantage of Kreyvium over Trivium is that it provides 128-bit security (instead of 80-bit) with the same multiplicative depth, and inherits the same security arguments. Beside a higher security level, it also accomodates longer IVs, so that it can encrypt up to $46 \cdot 2^{128}$ plaintext bits under the same key, with multiplicative depth only 12. We implemented our construction and instantiated it with Trivium, Kreyvium and LowMC in CTR-mode. Our results show that instantiating our construction with Trivium and Kreyvium is really competitive in term of latency compared to a LowMC-based instantiation (25% faster), but has a throughput 1.5 to 4 times smaller. In other words, we show that the promising performances attained by the HE-dedicated block cipher LowMC can also be achieved with well-known primitives whose security has been firmly established for over a decade.

Our second candidate for $\mathsf{E}$ relies on a completely different technique based on the observation that multiplication in binary fields is $\mathbb{F}_2$-bilinear, making it possible to homomorphically exponentiate field elements with a log-log-depth circuit. We also report a random oracle based proof that compressed ciphertexts are semantically secure under an appropriate complexity assumption. We show, however, that this second approach remains disappointingly impractical.

**Organization of the paper.** We introduce a general model and a generic construction to compress homomorphic ciphertexts in Section 2. Our first construction using Trivium and Kreyvium is described in Section 3. Subsequent experimental results and a comparison with LowMC are presented in Section 4. Section 5 presents and discusses our second construction based on discrete logs on binary fields.

## 2 A Generic Design for Efficient Decompression

In this section, we describe our model and generic construction to transmit compressed homomorphic ciphertexts between Alice and Charlie. We use the same notation as in the introduction: Alice wants to send some plaintext $m$, encrypted under Bob's public key $\mathsf{pk}$ (of an homomorphic encryption scheme $\mathsf{HE}$) to a third party evaluator Charlie.

### 2.1 Offline/Online Phases in Ciphertext Decompression

Most practical scenarios would likely find it important to distinguish between three distinct phases within the homomorphic evaluation of $\mathcal{C}_{\mathsf{E}^{-1}}$:

1. an *offline key-setup* phase which only depends on Bob's public key and can be performed once and for all before Charlie starts receiving compressed ciphertexts encrypted under Bob's key;

2. an *offline decompression* phase which can be performed only based on some plaintext-independent material found in the compressed ciphertext;
3. an *online decompression* phase which aggregates the result of the offline phase with the plaintext-dependent part of the compressed ciphertext and (possibly very quickly) recovers the decompressed ciphertext $c$.

As such, our general-purpose formulation $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ does not allow to make a clear distinction between these three phases. In our context, it is much more relevant to reformulate the encryption scheme as an IV-based encryption scheme where the encryption and decryption process are both deterministic but depend on an IV:

$$\mathsf{E}_k(m) \stackrel{\text{def}}{=} \left(IV, \mathsf{E}'_{k,IV}(m)\right) .$$

Since the IV has a limited length, it can be either transmitted during an offline preprocessing phase, or may alternately correspond to a state which is maintained by the server. Now, to minimize the latency of homomorphic decompression for Charlie, the online phase should be reduced to a minimum. The most appropriate choice in this respect consists in using an additive IV-based stream cipher $Z$ so that

$$\mathsf{E}'_{k,IV}(m) = Z(k, IV) \oplus m .$$

In this reformulation, the decompression process is clearly divided into a offline precomputation stage which only depends on $\mathsf{pk}, k$ and $IV$, and an online phase which is plaintext-dependent. The online phase is thus reduced to a mere XOR between the plaintext-dependent part of the ciphertext $\mathsf{E}'_{k,IV}(m)$ and the HE-encrypted keystream $\mathsf{HE}(Z(k, IV))$, which comes essentially for free in terms of noise growth in HE ciphertexts. All expensive operations (*i.e.* homomorphic multiplications) are performed during the offline decompression phase where $\mathsf{HE}(Z(k, IV))$ is computed from $\mathsf{HE}(k)$ and $IV$.

## 2.2 Our Generic Construction

We devise a generic construction based on a homomorphic encryption scheme $\mathsf{HE}$ with plaintext space $\{0, 1\}$, an expansion function $G$ mapping $\ell_{IV}$-bit strings to strings of arbitrary size, and a fixed-size parametrized function $F$ with input size $\ell_x$, parameter size $\ell_k$ and output size $N$. The construction is depicted on Fig. 1.

**Compressed encryption.** Given an $\ell_m$-bit plaintext $m$, Bob's public key $\mathsf{pk}$ and $IV \in \{0, 1\}^{\ell_{IV}}$, the compressed ciphertext $c'$ is computed as follows:

1. Set $t = \lceil \ell_m / N \rceil$,
2. Set $(x_1, \ldots, x_t) = G(IV; t\ell_x)$,
3. Randomly pick $k \leftarrow \{0, 1\}^{\ell_k}$,
4. For $1 \leq i \leq t$, compute $z_i = F_k(x_i)$,
5. Set $\mathsf{keystream}$ to the $\ell_m$ leftmost bits of $z_1 \,||\, \ldots \,||\, z_t$,
6. Output $c' = (\mathsf{HE}_{\mathsf{pk}}(k), m \oplus \mathsf{keystream})$.

**Ciphertext decompression.** Given $c'$ as above, Bob's public key $\mathsf{pk}$ and $IV \in \{0, 1\}^{\ell_{IV}}$, the ciphertext decompression is performed as follows:

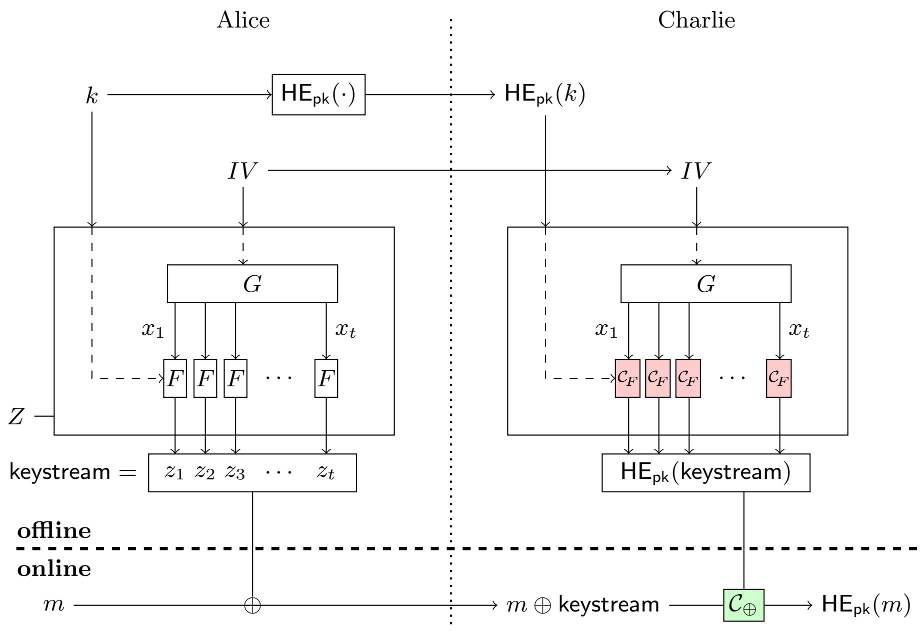


**Fig. 1.** Our generic construction. The multiplicative depth of the circuit is equal to the depth of $\mathcal{C}_F$. This will be the bottleneck in our protocol and we want the multiplicative depth of $F$ to be as small as possible. With current HE schemes, the circuit $\mathcal{C}_\oplus$ is usually very fast (addition of ciphertexts) and has a negligible impact on the noise in the ciphertext.

1. Set $t = \lceil \ell_m / N \rceil$,
2. Set $(x_1, \ldots, x_t) = G(IV; t\ell_x)$,
3. For $1 \le i \le t$, compute $\mathsf{HE_{pk}}(z_i) = \mathcal{C}_F\left(\mathsf{HE_{pk}}(k), x_i\right)$ with some circuit $\mathcal{C}_F$,
4. Deduce $\mathsf{HE_{pk}}(\mathsf{keystream})$ from $\mathsf{HE_{pk}}(z_1), \ldots, \mathsf{HE_{pk}}(z_t)$,
5. Compute $c = \mathsf{HE_{pk}}(m) = \mathcal{C}_\oplus\left(\mathsf{HE_{pk}}(\mathsf{keystream}), m \oplus \mathsf{keystream}\right)$.

The circuit $\mathcal{C}_\oplus$ computes $\mathsf{HE}(a \oplus b)$ given $\mathsf{HE}(a)$ and $b$ where $a$ and $b$ are bit-strings of the same size. In our construction, the cost of decompression per plaintext block is *fixed* and roughly equals one single evaluation of the circuit $\mathcal{C}_F$; most importantly, the multiplicative depth of the decompression circuit is also fixed, and set to the depth of $\mathcal{C}_F$.

**How secure are compressed ciphertexts?** From a high-level perspective, compressed homomorphic encryption is just hybrid encryption and relates to the generic KEM-DEM construct. However it just cannot inherit from the general security results attached to the KEM-DEM framework [AGKS05,HK07] since taking some HE scheme to implement the KEM part does not even fulfill the basic requirements that the KEM be IND-CCA or even IND-CCCA. It is usual that HE schemes succeed in achieving CPA security but often grossly fail to realize any form of CCA1 security, to the point of admitting simple key recovery attacks [CT14]. Therefore common KEM-DEM results just do not apply here.

On the other hand, CPA security is arguably strong enough for compressed homomorphic encryption, given that in practice Alice may always provide a signature $\sigma(c')$ together with $c'$ to Charlie to ensure origin and data authenticity. Thus, the right level of security requirement on the compressed encryption scheme itself seems to be just IND-CPA for concrete use. However, it is not known what minimal security assumptions to require from a homomorphic KEM and a general-purpose DEM to yield a KEM-DEM scheme that is provably IND-CPA. As a result of that, evidence that CPA security is reached may only be provided on a case-by-case basis given a specific embodiment.

**Instantiating the paradigm.** The rest of the paper focuses on how to choose the expansion function $G$ and function $F$ so that the homomorphic evaluation of $\mathcal{C}_F$ is as fast (and its multiplicative depth as low) as possible. More specifically, we investigate two different operating modes and embodiments of our generic compressed encryption scheme:

***Approach 1.*** Here, the value of $IV$ is assumed to be shared between Alice and Charlie and needs not be transmitted along with the compressed ciphertext. For instance, $IV$ is chosen to be an absolute constant such as $IV = 0^\ell$ where $\ell = \ell_{IV} = \ell_x$. Another example is to take for $IV \in \{0,1\}^\ell$ a synchronized state that is updated between transmissions. Also, the expansion function $G$ is chosen to implement a counter in the sense of the NIST description of the CTR mode [Nat01], for instance

$$G(IV; t\ell) = (IV, IV \boxplus 1, \ldots, IV \boxplus (t-1)) \quad \text{where} \quad a \boxplus b = (a+b) \bmod 2^\ell \ .$$

Finally, $F$ is chosen to follow a specific design to ensure both an appropriate security level and a low multiplicative depth. We focus in Section 3 on the keystream generator corresponding to Trivium, and on a new variant, called *Kreyvium*.

Interestingly, the output of an iterated PRF used in counter mode is computationally indistinguishable from random [BDJR97, Th. 13]. Hence, under the assumption that Trivium or Kreyvium is a PRF[6], the keystream $z_1 \mid\mid \ldots \mid\mid z_t$ produced by our construction is also indistinguishable. However, this is insufficient to prove that the compressed encryption scheme is semantically secure (IND-CPA), because the adversary also sees $\mathsf{HE}_{\mathsf{pk}}(k)$ during the IND-CPA game, which cannot be proven not to make the keystream distinguishable. Although the security of Approach 1 is empiric, Section 3 provides a strong rationale for the Kreyvium design and makes it the solution with the smallest homomorphic evaluation latency known so far.

*Why not using a block cipher for $F$?* Although not specifically in these terms, the use of lightweight block ciphers like PRINCE and SIMON has been proposed in the context of compressed homomorphic ciphertexts *e.g.* [LN14,DSES14]. However a complete encryption scheme based on the ciphers has not been defined. This is a major issue since the security provided by all classical modes of operation (including all variants of CBC, CTR, CFB, OFB, GCM, OCB...) is inherently limited to $2^{n/2}$ where $n$ is the block size [Rog11] (this is also emphasized in *e.g.* [KL14, p. 95]). Only a very few modes providing *beyond-birthday* security have been proposed [Iwa06,Yas11,LST12] but they induce a much higher implementation cost and their security is usually upper-bounded by $2^{2n/3}$.

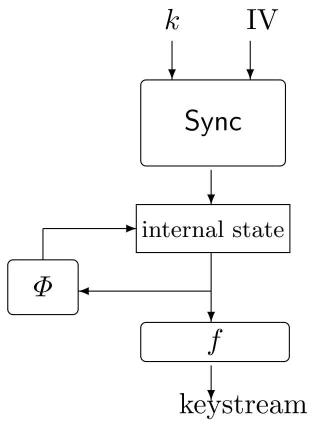
In other words, the use of a block cipher operating on 64-bit blocks like PRINCE or SIMON-32/64 implies that the number of blocks encrypted under the same key should be significantly less that $2^{32}$ (*i.e.* 32GB for 64-bit blocks). Therefore, only block ciphers with a large enough block size, like the LowMC instantiation with a 256-bit block proposed in [ARS+15], are suitable in applications which may require the encryption of more than $2^{32}$ bits under the same key.

[6] Note that this equivalent to say that Kreyvium instantiated with a random key and mapping the IV's to the keystream is secure [BG07, Sec. 3.2].

***Approach 2.*** Our second approach is an attempt to achieve provable security while ensuring a low-depth circuit $\mathcal{C}_F$. For this, we require $G$ to be a PRNG and $IV$ to be chosen at random at encryption time and transmitted within $c'$. This allows us to prove that $c'$ is semantically secure under a well-defined complexity assumption. Simultaneously, we use exponentiation in a binary field to instantiate $F$, which yields a circuit $\mathcal{C}_F$ of depth $\lceil \log \ell_k \rceil$. Performance estimations, however, show that Approach 2 is rather impractical.

## 3 Approach 1: Kreyvium, A Low-Depth Stream Cipher

Since an additive stream cipher is the optimal choice, we now focus on the keystream generation, and on its homomorphic evaluation. An IV-based keystream generator is decomposed into:



- a resynchronization function, Sync, which takes as input the IV and the key (possibly expanded by some precomputation phase), and outputs some $n$-bit initial state;
- a transition function $\Phi$ which computes the next state of the generator;
- a filtering function $f$ which computes a keystream segment from the current internal state.

Since generating $N$ keystream bits may require a circuit of depth up to

$$(\mathrm{depth}(\mathsf{Sync}) + N\,\mathrm{depth}(\Phi) + \mathrm{depth}(f))\,,$$

the best design strategy for minimizing this value consists in choosing a transition function with a small depth. The extreme option is to choose for $\Phi$ a linear function as in the CTR mode where the counter is implemented by an LFSR. An alternative strategy that we will investigate consists in choosing a nonlinear transition whose depth does not increase too fast when it is iterated. In Appendix A, we discuss the influence of Sync on the multiplicative depth of the circuit depending on which quantity should be encrypted under the HE scheme.

*Size of the internal state.* A major specificity of our context is that a large internal state can be easily handled. Indeed, in most classical stream ciphers, the internal-state size usually appears as a bottleneck because the overall size of the quantities to be stored highly influences the number of gates in the implementation. This is not the case in our context. It might seem, a priori, that increasing the size of the internal state automatically increases the number of nonlinear operations (because the number of inputs of $\Phi$ increases). But, this is not the case if a part of this larger internal state is used, for instance, for storing the secret key. This strategy can be used for increasing the security at no implementation cost. Indeed, the complexity of all generic attacks aiming at recovering the internal state of the generator is $\mathcal{O}(2^{n/2})$ where $n$ is the size of the secret part of the internal state even if some part is not updated during the keystream generation. For instance, the time-memory-data-tradeoff attacks in [Bab95,Gol97,BS00] aim at inverting the function which maps the internal state of the generator to the first keystream bits. But precomputing some values of this function must be feasible by the attacker, which is not the case if the filtering or transition function depends on some secret material. On the other hand, the size $n'$ of the non-constant secret part of the internal state determines the data complexity for finding a collision on the internal state: the length of the keystream produced from the same key is limited to $2^{n'/2}$. But, if the transition function or the filtering function depends on the IV, this limitation corresponds to the maximal keystream length produced from the same key/IV pair. It is worth noticing that many attacks require a very long keystream generated from the same key/IV pair and do not apply in our context since the keystream length is strictly limited by the multiplicative depth of the circuit.

### 3.1 Trivium in the HE setting

Trivium [CP08] is one of the seven stream ciphers recommended by the eSTREAM project after a 5-year international competition [ECR05]. Due to the small number of nonlinear operations in its transition function, it appears as a natural candidate in our context.

**Description.** Trivium is a synchronous stream cipher with a key and an IV of 80 bits each. Its internal state is composed of three registers of sizes 93, 84 and 111 bits, having an internal state size of 288 bits in total. Here, we use for the internal state the notation introduced by the designers: the leftmost bit of the 93-bit register is $s_1$, and its rightmost one is $s_{93}$; the leftmost bit of the register of size 84 is $s_{94}$ and the rightmost $s_{177}$; the leftmost bit of register of size 111 is $s_{178}$ and the rightmost $s_{288}$. The initialization and the generation of an $N$-bit Keystream are described below.

```
$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_0, \ldots, K_{79}, 0, \ldots, 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_0, \ldots, IV_{79}, 0, \ldots, 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
for $i = 1$ to $1152 + N$ do
    $t_1 \leftarrow s_{66} + s_{93}$
    $t_2 \leftarrow s_{162} + s_{177}$
    $t_3 \leftarrow s_{243} + s_{288}$
        if $i > 1152$ do
            output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
        end if
    $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
    $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
    $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
    $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
    $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
    $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
end for
```

No attack better than an exhaustive key search is known so far on the full Trivium. It can therefore be considered as a secure cipher. The family of attacks that seems to provide the best result on round-reduced versions is the cube attack and its variants [DS09,ADMS09,FV13]. They recover some key bits (resp. provide a distinguisher on the keystream) if the number of initialization rounds is reduced to 799 (resp. 885) rounds out of 1152. The highest number of initialization rounds that can be attacked is 961: in this case, a distinguisher exists for a class of weak keys [KMN11].

**Multiplicative depth.** It is easy to see that the multiplicative depth grows quite slowly with the number of iterations. An important observation is that, in the internal state, only the first

80 bits in Register 1 (the keybits) are initially encrypted under the HE and that, as a consequence, performing hybrid clear and encrypted data calculations is possible (this is done by means of the following simple rules: $0 \cdot [x] = 0$, $1 \cdot [x] = [x]$, $0 + [x] = [x]$ and $1 + [x] = [1] + [x]$, where the square brackets denote encrypted bits and where in all but the latter case, an homomorphic operation is avoided which is specially desirable for multiplications). This optimization allows for instance to increase the number of bits which can be generated (after the 1152 blank rounds) at depth 12 from 42 to 57 (*i.e.*, a 35% increase). Then, the relevant quantity in our context is the multiplicative depth of the circuit which computes $N$ keystream bits from the 80-bit key. The proof of the following proposition is given in Appendix B.

**Proposition 1.** *In Trivium, the keystream length $N(d)$ which can be produced from the 80-bit key with a circuit of multiplicative depth $d$, $d \geq 4$, is given by*

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 81 & \textit{if } d \equiv 0 \bmod 3 \\ 160 & \textit{if } d \equiv 1 \bmod 3 \\ 269 & \textit{if } d \equiv 2 \bmod 3 \end{cases} .$$

## 3.2 Kreyvium

Our first aim is to offer a variant of Trivium with 128-bit key and IV, without increasing the multiplicative depth of the corresponding circuit. Besides a higher security level, another advantage of this variant is that the number of possible IVs, and then the maximal length of data which can be encrypted under the same key, increases from $2^{80} N_{\mathsf{trivium}}(d)$ to $2^{128} N_{\mathsf{kreyvium}}(d)$. Increasing the key-size in Trivium is a challenging task: in particular, Maximov and Biryukov [MB07] pointed out that increasing the key-size in Trivium without any additional modification cannot be secure due to some attack with complexity less than $2^{128}$.

**Description.** Our proposal, Kreyvium, accomodates a key and an IV of 128 bits each. The only difference with the original Trivium is that we have added to the 288-bit internal state a 256-bit part corresponding to the secret key and the IV. This part of the state aims at making both the filtering and transition functions key- and IV-dependent. More precisely, these two functions $f$ and $\Phi$ depend on the key bits and IV bits, through the successive outputs of two shift-registers $K^*$ and $IV^*$ initialized by the key and by the IV respectively. The internal state is then composed of five registers of sizes 93, 84, 111, 128 and 128 bits, having an internal state size of 544 bits in total, among which 416 become unknown to the attacker after initialization.

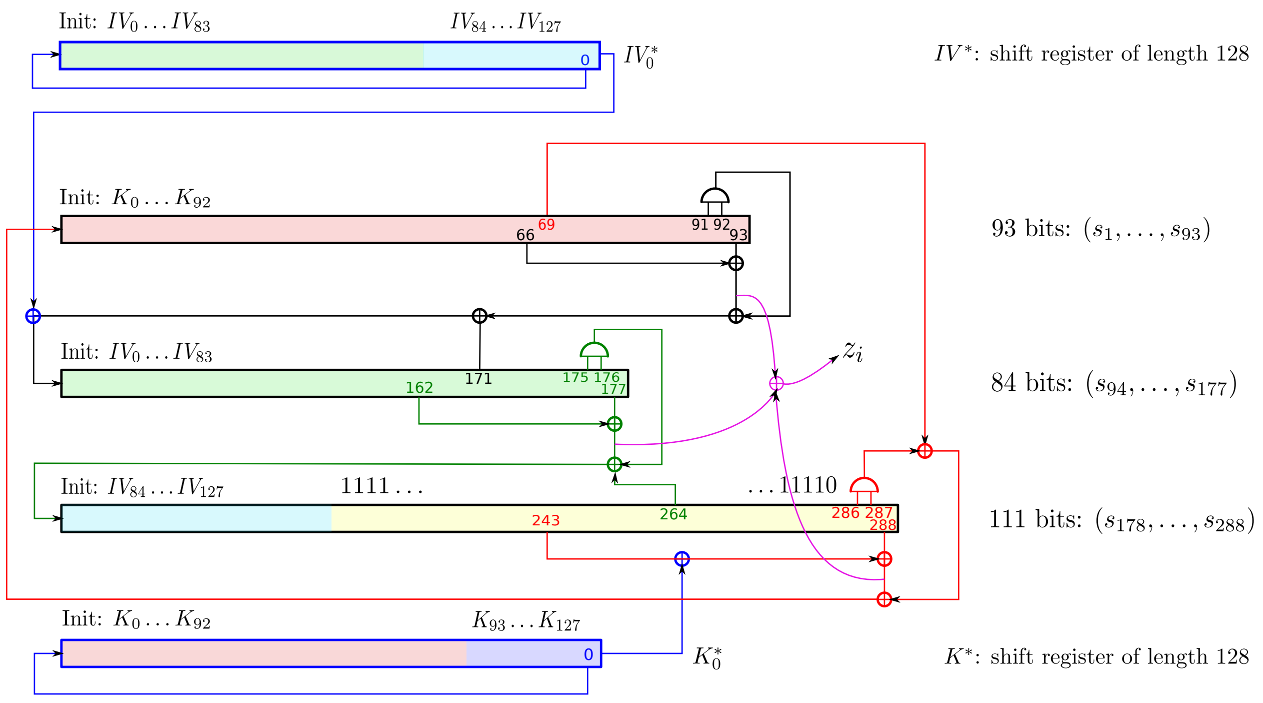
We will use the same notation as the description of Trivium, and for the additional registers we use the usual shift-register notation: the leftmost bit is denoted by $K^*_{127}$ (or $IV^*_{127}$), and the rightmost bit (*i.e.*, the output) is denoted by $K^*_0$ (or $IV^*_0$). Each one of these two registers are rotated independently from the rest of the cipher. The generator is described below, and depicted on Fig. 2.

```
$(s_1, s_2, \dots, s_{93}) \leftarrow (K_0, \dots, K_{92})$
$(s_{94}, s_{95}, \dots, s_{177}) \leftarrow (IV_0, \dots, IV_{83})$
$(s_{178}, s_{179}, \dots, s_{288}) \leftarrow (IV_{84}, \dots, IV_{127}, 1, \dots, 1, 0)$
$(K^*_{127}, K^*_{126}, \dots, K^*_0) \leftarrow (K_0, \dots, K_{127})$
$(IV^*_{127}, IV^*_{126}, \dots, IV^*_0) \leftarrow (IV_0, \dots, IV_{127})$
for $i = 1$ to $1152 + N$ do
    $t_1 \leftarrow s_{66} + s_{93}$
    $t_2 \leftarrow s_{162} + s_{177}$
    $t_3 \leftarrow s_{243} + s_{288} + \boldsymbol{K^*_0}$
        if $i > 1152$ do
            output $z_{i-1152} \leftarrow t_1 + t_2 + t_3$
        end if
    $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171} + \boldsymbol{IV^*_0}$
    $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
    $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
    $t_4 \leftarrow K^*_0$
    $t_5 \leftarrow IV^*_0$
    $(s_1, s_2, \dots, s_{93}) \leftarrow (t_3, s_1, \dots, s_{92})$
    $(s_{94}, s_{95}, \dots, s_{177}) \leftarrow (t_1, s_{94}, \dots, s_{176})$
    $(s_{178}, s_{179}, \dots, s_{288}) \leftarrow (t_2, s_{178}, \dots, s_{287})$
    $(K^*_{127}, K^*_{126}, \dots, K^*_0) \leftarrow (t_4, K^*_{127}, \dots, K^*_1)$
    $(IV^*_{127}, IV^*_{126}, \dots, IV^*_0) \leftarrow (t_5, IV^*_{127}, \dots, IV^*_1)$
end for
```

**Related ciphers.** KATAN [CDK09] is a lightweight block cipher with a lot in common with Trivium. It is composed of two registers, whose feedback functions are very sparse, and have a single nonlinear term. The key, instead of being used for initializing the state, is introduced by XORing two key information-bits per round to each feedback bit. The recently proposed stream cipher Sprout [AM15], inspired by Grain but with much smaller registers, also inserts the key in a similar way: instead of using the key for initializing the state, one key information-bit is XORed at each clock to the feedback function. We can see the parallelism between these two ciphers and our newly proposed variant. In particular, the previous security analysis on KATAN shows that this type of design does not introduce any clear weakness. Indeed, the best attacks on round-reduced versions of KATAN so far [FM14] are meet-in-the-middle attacks, that exploit the knowledge of the values of the first and the last internal states (due to the block-cipher setting). As this is not the case here, such attacks do not apply. The best attacks, when excluding MitM techniques, are conditional differential attacks [KMN10,KMN11].

**Design rationale.** In Kreyvium, we have decided to XOR the keybit $K^*_0$ to the feedback function of the register that interacts with the content of $(s_1, \dots, s_{63})$ the later, since $(s_1, \dots, s_{63})$ is initialized with some key bits. The same goes for the $IV^*$ register. Moreover, as the keybits that start entering the state are the ones that were not in the initial state, all the keybits affect the state the soonest possible.

We also decided to initialize the state with some keybits and with all the IV bits, and not with a constant value, as this way the mixing will be performed quicker. Then we can expect that the



Init: $IV_0 \dots IV_{83}$
$IV_{84} \dots IV_{127}$
$IV_0^*$
$IV^*$: shift register of length 128
Init: $K_0 \dots K_{92}$
93 bits: $(s_1, \dots, s_{93})$
$z_i$
Init: $IV_0 \dots IV_{83}$
84 bits: $(s_{94}, \dots, s_{177})$
Init: $IV_{84} \dots IV_{127}$
1111 ...
... 11110
111 bits: $(s_{178}, \dots, s_{288})$
Init: $K_0 \dots K_{92}$
$K_{93} \dots K_{127}$
$K_0^*$
$K^*$: shift register of length 128

**Fig. 2.** Kreyvium. The three registers in the middle correspond to the original Trivium. The modifications defining Kreyvium correspond to the two registers in blue.

internal-state bits after initialization are expressed as more complex and less sparse functions in the key and IV bits.

Our change of constant is motivated by the conditional differential attacks from [KMN11]: the conditions needed for a successful attack are that 106 bits from the IV or the key are equal to '0' and a single one needs to be '1'. This suggests that values set to zero "encourage" non-random behaviors, leading to our new constant. In other words, in Trivium, an all-zero internal state is always updated in an all-zero state, while an all-one state will change through time. The 0 at the end of the constant is added for preventing slide attacks.

**Multiplicative depth.** Exactly as for Trivium, we can compute the number of keystream bits which can be generated from the key at a given depth (see the proof in Appendix B). The only difference with Trivium is that the first register now contains 93 key bits instead of 80. For this reason, the optimization using hybrid cleartext/ciphertext calculations is a bit less interesting: for any fixed depth $d \geq 4$, we can generate 11 bits less than with Trivium.

**Proposition 2.** *In Kreyvium, the keystream length $N(d)$ which can be produced from the 128-bit key with a circuit of multiplicative depth $d$, $d \geq 4$, is given by*

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 70 & \textit{if } d \equiv 0 \bmod 3 \\ 149 & \textit{if } d \equiv 1 \bmod 3 \\ 258 & \textit{if } d \equiv 2 \bmod 3 \end{cases} .$$

**Security analysis.** We investigate in more detail how all the known attacks on Trivium, and some other techniques, can apply to Kreyvium.

- **TMDTO.** TMDTO attacks aiming at recovering the initial state of the cipher do not apply since the size of the secret part of the internal state (416 bits) is much larger than twice the key-size. As discussed at the beginning of Section 3, the size of the whole secret internal state has to be taken into account, even if the additional 128-part corresponding to $K^*$ is independent from the rest of the state. On the other hand, TMDTO aiming at recovering the key have complexity larger than exhaustive key search (even without any restriction on the precomputation time) since the key and the IV have the same size [HS05,CLP05].
- **Internal-state collision.** As discussed in Section 3, a distinguisher may be built if the attacker is able to find two colliding internal states, since the two keystream sequences produced from colliding states are identical. Finding such a collision requires around $2^{144}$ keystream bits generated from the same key/IV pair, which is much longer than the maximal keystream length allowed by the multiplicative depth of the circuit. But, for a given key, two internal states colliding on all bits except on $IV^*$ lead to two keystreams which have the same first 69 bits since $IV^*$ affects the keystream only 69 clocks later. Moreover, if the difference between the two values of $IV^*$ when the rest of the state collides lies in the leftmost bit, then this difference will affect the keystream bits $(69+128) = 197$ clocks later. This implies that, within around $2^{144}$ keystream bits generated from the same key, we can find two identical runs of 197 consecutive bits which are equal. However, this property does not provide a valid distinguisher because a random sequence of length $2^{144}$ blocks is expected to contain much more collisions on 197-bit runs. Therefore, the birthday-bound of $2^{144}$ bits provides a limit on the number of bits produced from the same key/IV pair, not on the bits produced from the same IV.
- **Cube attacks [DS09,FV13] and cube testers [ADMS09].** As previously pointed out, they provide the best attacks for round-reduced Trivium. In our case, as we keep the same main function, but we have two additional XORs per round, thus a better mixing of the variables, we can expect the relations to get more involved and hamper the application of previously defined round-reduced distinguishers. One might wonder if the fact that more variables are involved could ease the attacker's task, but we point out here that the limitation in the previous attacks was not the IV size, but the size of the cubes themselves. Therefore, having more variables available is of no help with respect to this point. We can conclude that the resistance of Kreyvium to these types of attacks is at least the resistance of Trivium, and even better.
- **Conditional differential cryptanalysis.** Because of its applicability to both Trivium and KATAN, the attack from [KMN11] is definitely of interest in our case. In particular, the highest number of blank rounds is reached if some conditions on two registers are satisfied at the same time (and not only conditions on the register controlled by the IV bits in the original Trivium). In our case, as we have IV bits in two registers, it is important to elucidate whether an attacker can take advantage of introducing differences in two registers simultaneously. First, let us recall that we have changed the constant to one containing mostly 1. We previously saw that the conditions that favor the attacks are values set to zero in the initial state. In Trivium, per design, we have $(108 + 4 + 13) = 125$ bits already fixed to zero in the initial state, 3 are fixed to one and the others can be controlled by the attacker in the weak-key setting (and the attacker will force them to be zero most of the time). Now, instead, we have 64 bits forced to be 1, 1 equal to zero, and $(128 + 93) = 221$ bits of the initial state controlled by the attacker in the weak-key setting, plus potentially 21 additional bits from the key still not used, that will be inserted during the first rounds. We can conclude that, while in Trivium is possible in the weak-key setting, to introduce zeros in the whole initial state but in 3 bits, in Kreyvium, we will never be able to set to zero

64 bits, implying that applying the techniques from [KMN11] becomes much harder. Additionally, as in the discussion on cube attacks, we can also hope here that we get more involved relations that will provide a better resistance against these attacks.

· **Algebraic attacks.** Several algebraic attacks have been proposed against Trivium, aiming at recovering the 288-bit internal state at the beginning of the keystream generation (i.e., at time $t = 1153$) from the knowledge of the keystream bits. The most efficient attack of this type is due to Maximov and Biryukov [MB07]. It exploits the fact that the 22 keystream bits at time $3t'$, $0 \leq t' < 22$, are determined by all bits of the initial state at indexes divisible by 3 (starting from the leftmost bit in each register). Moreover, once all bits at positions $3i$ are known, then guessing that the outputs of the three AND gates at time $3t'$ are zero provides 3 linear relations between the bits of the internal state and the keystream bits. The attack then consists of an exhaustive search for some bits at indexes divisible by 3. The other bits in such positions are then deduced by solving the linear system derived from the keystream bits at positions $3t'$. Once all these bits have been determined, the other 192 bits of the initial state are deduced from the other keystream equations. This process must be iterated until the guess for the outputs of the AND gates is correct. In the case of Trivium, the outputs of at least 125 AND gates must be guessed in order to get 192 linear relations involving the 192 bits at indexes $3i+1$ and $3i+2$. This implies that the attack has to be repeated $(4/3)^{125} = 2^{52}$ times. From these guesses, we get many linear relations involving the bits at positions $3i$ only, implying that only an exhaustive search with complexity $2^{32}$ for the other bits at positions $3i$ is needed. Therefore, the overall complexity of the attack is around $2^{32} \times 2^{52} = 2^{84}$. A similar algorithm can be applied to Kreyvium, but the main difference is that every linear equation corresponding to a keystream bit also involves one key bit. Moreover, the key bits involved in the generation of any 128 consecutive output bits are independent. It follows that each of the first 128 linear equations introduces a new unknown in the system to solve. For this reason, it is not possible to determine all bits at positions $3i$ by an exhaustive search on less than 96 bits like for Trivium. Moreover, the outputs of more than 135 AND gates must be guessed for obtaining enough equations on the remaining bits of the initial state. Therefore the overall complexity of the attack exceeds $2^{96} \times 2^{52} = 2^{148}$ and is much higher that the cost of the exhaustive key search. It is worth noticing that the attack would have been more efficient if only the feedback bits, and not the keystream bits, would have been dependent on the key. In this case, 22 linear relations independent from the key would have been available to the attacker.

## 4 Experimental Results

In this section, we discuss and compare the practicality of our generic construction (*cf.* Section 2) when instantiated with Trivium, Kreyvium and LowMC. The expansion function $G$ implements a mere counter, and these three algorithms are used to instantiate the function $F$ that produces $N$ bits of keystream per iteration. Table 1 recalls the characteristics of the different algorithms in use (for Trivium and Kreyvium, $N$ can be computed from Propositions 1 and 2).

**HE framework.** Our experiments were based on the Fan-Vercauteren homomorphic encryption scheme [FV12], a scale-invariant version of the BGV scheme [BGV14]. This scheme was used in several works [AMFF+13,FSF+13,LN14,CDS15], is conceptually simpler than the BGV scheme and is one of the most efficient HE schemes. In our experiments, we systematically used the Armadillo

**Table 1.** Algorithms in use and their characteristics: $N$ stands for the bit-size of the output produced by a circuit of multiplicative depth "$\times$ depth".

| Algorithm | security level $\kappa$ | $\times$ depth | $N$ |
|---|---|---|---|
| Trivium-12 | 80 | 12 | 57 |
| Trivium-13 | 80 | 13 | 136 |
| Kreyvium-12 | 128 | 12 | 46 |
| Kreyvium-13 | 128 | 13 | 125 |
| LowMC-80 | 80 | 11 | 256 |
| LowMC-128 | 128 | 12 | 256 |

compiler [CDS15]. This source-to-source compiler turns a C++ algorithm into a boolean circuit, optimizes it, and generates an OpenMP parallel code which can then be combined with our HE scheme.

**Parameter selection for subsequent homomorphic processing.** We selected the HE parameters to be able to handle circuits of multiplicative depth 20. Indeed, even though the algorithms of Table 1 would only require an homomorphic encryption scheme handling circuits of depth 11-13, we chose 20 to allow further homomorphic processing by Charlie (which is obviously what is expected in applications of homomorphic encryption). In practice, numerous applications use algorithms of multiplicative depth smaller than 7-9 (*e.g.* [GLN12,LLN14]).

It is worth mentioning that most previous works on the homomorphic evaluation of symmetric encryption schemes select parameters for the exact multiplicative depth required and not beyond [GHS12,CLT14,LN14,DSES14,ARS$^+$15]. This means that once the ciphertext is decompressed, no further homomorphic computation can actually be performed by Charlie – which makes the claimed timings less meaningful in a real-world context.

**Experimental results and interpretation.** Our results are given in Table 2.

**Table 2.** Latency and throughput for the algorithms in use. (4-core Intel Xeon E5630 at 2.53GHz)

| Algorithm | sec. level $\kappa$ | $\times$ depth | $N$ | latency | throughput |
|---|---|---|---|---|---|
| Trivium-12 | 80 | 12 | 57 | 13.3min | 4.27bits/min |
| Trivium-13 | 80 | 13 | 136 | 16.4min | 8.28bits/min |
| LowMC-80 | 80 | 11 | 256 | 17.7min | 14.46bits/min |
| Kreyvium-12 | 128 | 12 | 46 | 18.6min | 2.47bits/min |
| Kreyvium-13 | 128 | 13 | 125 | 19.0min | 6.57bits/min |
| LowMC-128 | 128 | 12 | 256 | 24.9min | 10.70bits/min |

We recall that latency refers to the time required to perform the entire homomorphic evaluation whereas the throughput is the number of blocks processed per time unit. As already emphasized in [LN14], real-world applications of homomorphic encryption (which are often cloud-based applications) should be implemented in a transparent and user-friendly way. In particular a seamless experience for the end-users demands a latency made as small as possible (possibly at the cost of a higher computational power).[7] Our results suggest that Trivium and Kreyvium have a latency

[7] Note that due to the highly parallel structure of the binary circuit of Trivium and Kreyvium, doubling the number of cores (up to $\approx$ 250 cores) will divide by 2 the latency of the homomorphic evaluation. For example, on a standard

25% smaller than LowMC. Now [ARS$^+$15] demonstrated the benefit of LowMC for homomorphic evaluation compared to other approaches in the literature.

On the other hand, Trivium and Kreyvium have a throughput 1.5 to 4 times smaller than LowMC. This means that for a bounded computational power, there is a latency/throughput trade-off between Trivium/Kreyvium and LowMC which depends on the application.

Note that the timings given in *e.g.* [ARS$^+$15] cannot be compared quantitatively with our timings because of the difference in the choice of scheme parameters. In particular, we emphasize again that we selected the parameters to handle circuits of multiplicative depth larger than just the multiplicative depth of Trivium/Kreyvium/LowMC to allow Charlie to perform subsequent homomorphic computations on the decompressed ciphertexts (as opposed to [ARS$^+$15] for LowMC–and other homomorphic evaluation of block ciphers [GHS12,CLT14,LN14,DSES14]).[8]

Finally, we recall that our construction was aiming at compressing the size of transmissions between Alice and Charlie. We support an encryption rate $|c'|/|m|$ that becomes asymptotically close to 1 for long messages. Table 3 provides concrete expansion rates for different sizes of $m$.

**Table 3.** Expansion Rate: $|c'|/|m|$

| Message length $\ell_m$ | Send $\mathsf{HE}_{\mathsf{pk}}(m)$ | Generic construction | |
|---|---|---|---|
| | | with Trivium/LowMC | with Kreyvium/LowMC |
| 1kB | 3232000 | 80801 | 166401 |
| 1MB | 3232000 | 81.8 | 167.4 |
| 1GB | 3232000 | 1.08 | 1.16 |

**Some deeper analysis.** We provide a more thorough analysis of the number of AND and XOR gates in the different circuits in Appendix C.

## 5 Approach 2: Using Discrete Logs on Binary Fields

We now introduce our second, discrete-log based embodiment of the generic compressed encryption scheme of Section 2.2. We recall that the homomorphic encryption scheme $\mathsf{HE}_{\mathsf{pk}}(\cdot)$ is assumed to encrypt separately each plaintext bit. For $h \in \mathbb{F}_{2^n}$, we identify $h$ with the vector of its coefficients and therefore by $\mathsf{HE}_{\mathsf{pk}}(h)$, we mean the vector composed of the encrypted coefficients of $h$.

### 5.1 Description

In this approach, the operating mode picks a fresh $IV \leftarrow \{0,1\}^{\ell_{IV}}$ for each compressed ciphertext. The expansion function $G$ is instantiated by some PRNG that we will view as a random oracle in the security proof. Also, we set

$$\ell_x = N = n \ ,$$

and therefore $F$ maps $n$-bit inputs to $n$-bit outputs under $\ell_k$-bit parameters. Given $k \in \{0,1\}^{\ell_k}$ and $x \in \{0,1\}^n$, $F_k(x)$ views $x$ as a field element in $\mathbb{F}_{2^n}$ and $k$ as an $\ell_k$-bit integer, computes $z = x^k$ over

high-end 48-core NUMA server, an homomorphic evaluation of Trivium (resp. Kreyvium) took 76 seconds (resp. 95 seconds).

[8] Also, note that the homomorphic evaluation of the 128-bit secure LowMC-128 block cipher in [ARS$^+$15] was perform with a 80-bit secure HE scheme, providing a 80-bit security overall (and obviously faster performances).

$\mathbb{F}_{2^n}$, views $z$ as an $n$-bit string and outputs $z$. This completes the description of the compressed encryption scheme.

## 5.2 A log-log-depth exponentiation circuit over $\mathbb{F}_{2^n}$

We describe a circuit $\mathcal{C}_{\exp}$ which, given a field element $h \in \mathbb{F}_{2^n}$ and an encrypted exponent $\mathsf{HE}_{\mathsf{pk}}(k)$ with $k \in \{0,1\}^{\ell_k}$, computes $\mathsf{HE}_{\mathsf{pk}}(h^k)$ and has multiplicative depth at most $\lceil \log \ell_k \rceil$.

Stricto sensu, $\mathcal{C}_{\exp}$ is not just a Boolean circuit evaluated homomorphically, as it combines computations in the clear, homomorphic $\mathbb{F}_2$-arithmetic on encrypted bits, and $\mathbb{F}_2$-arithmetic on mixed cleartext/encrypted bits.

$\mathcal{C}_{\exp}$ uses implicitly some irreducible polynomial $p$ to represent $\mathbb{F}_{2^n}$ and we denote by $\oplus$ and $\otimes_p$ the field operators. The basic idea here is that for any $a, b \in \mathbb{F}_{2^n}$, computing $\mathsf{HE}(a \otimes_p b)$ from $\mathsf{HE}(a), \mathsf{HE}(b)$ requires *only* 1 *multiplicative level*, simply because $\otimes_p$ is $\mathbb{F}_2$-bilinear. Therefore, knowing $p$ and the characteristics of $\mathsf{HE}$, we can efficiently implement a bilinear operator on encrypted binary vectors to compute

$$\mathsf{HE}(a \otimes_p b) = \mathsf{HE}(a) \ \otimes_p^{\mathsf{HE}} \ \mathsf{HE}(b) \ .$$

A second useful observation is that for any $a \in \mathbb{F}_{2^n}$ and $\beta \in \{0,1\}$, there is a multiplication-free way to deduce $\mathsf{HE}(a^\beta)$ from $a$ and $\mathsf{HE}(\beta)$. When $\beta = 1$, $a^\beta$ is just $a$ and $a^\beta = 1_{\mathbb{F}_{2^n}} = (1, 0, \ldots, 0)$ otherwise. Therefore to construct a vector $v = (v_0, \ldots, v_{n-1}) = \mathsf{HE}(a^\beta)$, it is enough to set

$$v_i := \begin{cases} \mathsf{HE}(0) & \text{if } a_i = 0 \\ \mathsf{HE}(\beta) & \text{if } a_i = 1 \end{cases}$$

for $i = 1, \ldots, n-1$ and

$$v_0 := \begin{cases} \mathsf{HE}(\beta \oplus 1) & \text{if } a_0 = 0 \\ \mathsf{HE}(1) & \text{if } a_i = 1 \end{cases}$$

where it does not matter that the same encryption of 0 be used multiple times. Let us denote this procedure as

$$\mathsf{HE}(a^\beta) = \mathsf{L}_a\left(\mathsf{HE}(\beta)\right) \ .$$

Now, given as input $h \in \mathbb{F}_{2^n}$, $\mathcal{C}_{\exp}$ first computes in the clear $h_i = h^{2^i}$ for $i = 0, \ldots, \ell_k - 1$. Since

$$h^k = h_0^{k_0} \otimes_p h_1^{k_1} \otimes_p \cdots \otimes_p h_{\ell_k - 1}^{k_{\ell_k - 1}} \ ,$$

one gets

$$\begin{aligned} \mathsf{HE}(h^k) &= \mathsf{HE}\left(h_0^{k_0}\right) \ \otimes_p^{\mathsf{HE}} \mathsf{HE}\left(h_1^{k_1}\right) \ \otimes_p^{\mathsf{HE}} \ \cdots \ \otimes_p^{\mathsf{HE}} \mathsf{HE}\left(h_{\ell_k-1}^{k_{\ell_k-1}}\right) \\ &= \mathsf{L}_{h_0}\left(\mathsf{HE}\left(k_0\right)\right) \ \otimes_p^{\mathsf{HE}} \mathsf{L}_{h_1}\left(\mathsf{HE}\left(k_1\right)\right) \ \otimes_p^{\mathsf{HE}} \ \cdots \ \otimes_p^{\mathsf{HE}} \mathsf{L}_{h_{\ell_k-1}}\left(\mathsf{HE}\left(k_{\ell_k-1}\right)\right) \ . \end{aligned}$$

Viewing the $\ell_k$ variables as the leaves of a binary tree, $\mathcal{C}_{\exp}$ therefore requires at most $\lceil \log \ell_k \rceil$ levels of homomorphic multiplications to compute and return $\mathsf{HE}_{\mathsf{pk}}(h^k)$.

## 5.3 Security Results

Given some homomorphic encryption scheme $\mathsf{HE}$ and security parameters $\kappa, n, \ell_k$, we define a family of decision problems $\{\mathsf{DP}_t\}_{t>0}$ as follows.

**Definition 1 (Decision Problem $\mathsf{DP}_t$).** *Let $\mathsf{pk} \leftarrow \mathsf{HE.KeyGen}(1^\kappa)$ be a random public key, $k \leftarrow \{0,1\}^{\ell_k}$ a random $\ell_k$-bit integer and $g_1, \ldots, g_t, g'_1, \ldots, g'_t \leftarrow \mathbb{F}_{2^n}$, $2t$ random field elements. Distinguish the distributions*

$$D_{t,1} = \left(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, g_1^k, \ldots, g_t^k\right) \quad \text{and}$$
$$D_{t,0} = \left(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, g'_1, \ldots, g'_t\right) \ .$$

**Theorem 1.** *Viewing $G$ as a random oracle, the compressed encryption scheme described above is semantically secure (IND-CPA), unless breaking $\mathsf{DP}_t$ is efficient, for messages of bit-size $\ell_m$ with $(t-1)n < \ell_m \le tn$.*

*Proof (Sketch).* A random-oracle version of the PRNG function $G$ is an oracle that takes as input a pair $(IV, \ell)$ where $IV \in \{0,1\}^{\ell_{IV}}$ and $\ell \in \mathbb{N}^*$, and returns an $\ell$-bit random string. It is also imposed to the oracle that $G(IV; \ell_1)$ be a prefix of $G(IV; \ell_2)$ for any $IV$ and $\ell_1 \le \ell_2$.

We rely on the real-or-random flavor of the IND-CPA security game and build a reduction algorithm $\mathcal{R}$ that uses an adversary $\mathcal{A}^G$ against the scheme to break $\mathsf{DP}_t$ as follows. $\mathcal{R}$ is given as input some $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, \tilde{g}_1, \ldots, \tilde{g}_t)$ sampled from $D_{t,b}$ and has to guess the bit $b$. $\mathcal{R}$ runs $\mathcal{A}^G(\mathsf{pk})$ and receives some challenge plaintext $m^\star \in \{0,1\}^{\ell_m}$ where $(t-1)n < \ell_m \le tn$. $\mathcal{R}$ makes use of its input to build a compressed ciphertext $c'$ as follows:

1. Set $\mathsf{keystream}$ to the $\ell_m$ leftmost bits of $\tilde{g}_1 \,||\, \ldots \,||\, \tilde{g}_t$,
2. Pick a random $IV^\star \leftarrow \{0,1\}^{\ell_{IV}}$,
3. Abort if $G(IV^\star; \ell')$ is already defined for some $\ell'$,
4. Set $G(IV^\star; tn)$ to $g_1 \,||\, \ldots \,||\, g_t$.
5. Set $c' = (\mathsf{HE}_{\mathsf{pk}}(k), IV^\star, m^\star \oplus \mathsf{keystream})$,

$\mathcal{R}$ then returns $c'$ to $\mathcal{A}$ and forwards $\mathcal{A}$'s guess $\hat{b}$ to its own challenger. At any moment, $\mathcal{R}$ responds to $\mathcal{A}$'s queries to $G$ using fresh random strings for each new query or to extend a past query to a larger size. Obviously, all the statistical distributions comply with their specifications. Consequently $c'$ is an encryption of $m^\star$ if the input instance comes from $D_{t,1}$ and is an encryption of some perfectly uniform plaintext if the instance follows $D_{t,0}$. The reduction is tight as long as the abortion probability $q2^{-\ell_{IV}}$ remains negligible, $q$ being the number of oracle queries made by $\mathcal{A}$. $\square$

Interestingly, we note the following fact about our family of decision problems.

**Theorem 2.** *For any $t \ge 2$, $\mathsf{DP}_t$ is equivalent to $\mathsf{DP}_2$.*

*Proof.* Obviously, a problem instance $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, \ldots, g_t, \tilde{g}_1, \ldots, \tilde{g}_t)$ sampled from $D_{t,b}$ can be converted into an instance of $D_{2,b}$ for the same $b$, by just removing $g_3, \ldots, g_t$ and $\tilde{g}_3, \ldots, \tilde{g}_t$. This operation preserves the distributions of all inner variables. Therefore $\mathsf{DP}_t$ can be reduced to $\mathsf{DP}_2$. Now, we describe a reduction $\mathcal{R}$ which, given an instance $(\mathsf{pk}, \mathsf{HE}_{\mathsf{pk}}(k), g_1, g_2, \tilde{g}_1, \tilde{g}_2)$ sampled from $D_{2,b}$, makes use of an adversary $\mathcal{A}$ against $\mathsf{DP}_t$ to successfully guess $b$. $\mathcal{R}$ converts its instance of

$D_{2,b}$ into an instance of $D_{t,b}$ as follows. For $i = 3, \ldots, t$, $\mathcal{R}$ randomly selects $\alpha_i \leftarrow \mathbb{Z}/(2^n - 1)\mathbb{Z}$ and sets

$$g_i = g_1^{\alpha_i} g_2^{1-\alpha_i} \ , \quad \tilde{g}_i = \tilde{g}_1^{\alpha_i} \tilde{g}_2^{1-\alpha_i} \ .$$

It is easily seen that, if $\tilde{g}_1 = g_1^k$ and $\tilde{g}_2 = g_2^k$ then $\tilde{g}_i = g_i^k$ for every $i$. If however $\tilde{g}_1, \tilde{g}_2$ are uniformly and independently distributed over $\mathbb{F}_{2^n}$ then so are $\tilde{g}_3, \ldots, \tilde{g}_t$. Our reduction runs $\mathcal{A}$ over that instance and outputs the guess $\hat{b}$ returned by $\mathcal{A}$. Obviously $\mathcal{R}$ is tight. □

Overall, the security of our compressed encryption scheme relies on breaking $\mathsf{DP}_1$ for messages of bit-size at most $n$ and on breaking $\mathsf{DP}_2$ for larger messages. Beyond the fact that $\mathsf{DP}_2$ reduces to $\mathsf{DP}_1$, we note that these two problems are unlikely to be equivalent since $\mathsf{DP}_2$ is easily broken using a DDH oracle over $\mathbb{F}_{2^n}$ while $\mathsf{DP}_1$ seems to remain unaffected by it.

## 5.4 Performance Issues

**Concrete security parameters.** Note that our decisional security assumptions $\mathsf{DP}_t^{\exp}$ for all $t \geq 1$ reduce to the discrete logarithm computation in the finite field $\mathbb{F}_{2^n}$. Solving discrete logarithm in finite fields of small characteristics is currently a very active research area, marked notably by the quasi-polynomial algorithm of Barbulescu, Gaudry, Joux and Thomé [BGJT14]. In particular, the expected security one can hope for has been recently completely redefined [GKZ14,AMOR14]. In our setting, we will select a prime $n$ so that computing discrete logarithms in $\mathbb{F}_{2^n}$ has complexity $2^\kappa$ for $\kappa$-bit security. The first step of Barbulescu *et al.* algorithm runs in polynomial time. This step has been extensively studied and its complexity has been brought down to $\mathcal{O}((2^{\log_2 n})^6)$ using a very complex and tight analysis by Joux and Pierrot [JP14]. As for the quasi-polynomial step of the algorithm, its complexity can be upper-bounded, but in practice numerous trade-off can be used and it is difficult to give to lower bound it [BGJT14,AMOR14]. To remains conservative in our choice of parameters, we will base our security on the first step. To ensure a 80-bit (resp. 128-bit) security level, one should therefore choose a prime $n$ of $\log_2 n \approx 14$ bits (resp. 23 bits), *i.e.* work in a finite field of about $16,000$ elements (resp. 4 million elements).

**How impractical is this approach?** We now briefly see why our discrete-log based construction on binary fields is impractical. We focus more specifically on the exponentiation circuit $\mathcal{C}_{\exp}$ whose most critical subroutine is a general-purpose field multiplication in the encrypted domain. Taking homomorphic bit multiplication as the complexity unit and neglecting everything else, how fast can we expect to multiply encrypted field elements in $\mathbb{F}_{2^n}$?

When working in the cleartext domain, several families of techniques exist with attractive asymptotic complexities for large $n$, such as algorithms derived from Toom-Cook [Bod07] or Schönhage-Strassen [Pin89]. It is unclear how these different strategies can be adapted to our case and with what complexities[9]. However, let us optimistically assume that they *could be adapted somehow* and that one of these adaptations would just take $n$ homomorphic bit multiplications.

A straightforward implementation of $\mathcal{C}_{\exp}$ consists in viewing all circuit inputs $\mathsf{L}_{h_i}(\mathsf{HE}(k_i))$ as generic encrypted field elements and in performing generic field multiplications along the binary tree, which would require $\ell_k \cdot n$ homomorphic bit multiplications. Taking $\ell_k = 160$, $n = 16000$

[9] One could expect these techniques to become the most efficient ones here since their prohibitive overhead would disappear in the context of homomorphic circuits.

and 0.5 seconds for each bit multiplication (as a rough estimate of the timings of Section 4), this accounts for more than 14 days of computation.

This can be improved because the circuit inputs are precisely not generic encrypted field elements; each one of the $n$ ciphertexts in $\mathsf{L}_{h_i}(\mathsf{HE}(k_i))$ is known to equal either $\mathsf{HE}(k_i)$, $\mathsf{HE}(k_i \oplus 1)$, $\mathsf{HE}(0)$ or $\mathsf{HE}(1)$. Similarly, a circuit variable of depth 1 *i.e.*

$$\mathsf{L}_{h_i}(\mathsf{HE}(k_i)) \otimes_p^{\mathsf{HE}} \mathsf{L}_{h_{i+1}}(\mathsf{HE}(k_{i+1})) \ ,$$

contains $n$ ciphertexts that are all an encryption of one of the 16 quadratic polynomials $ak_ik_{i+1} + bk_i + ck_{i+1} + d$ for $a, b, c, d \in \{0, 1\}$. This leads us to a strategy where one *simulates* the $\tau$ first levels of field multiplications at once, by computing the $2^{\lceil \log \ell_k \rceil - \tau}$ dictionaries of the form

$$\left\{\mathsf{HE}\left(k_i^{b_0} k_{i+1}^{b_1} \cdots k_{i+2^\tau-1}^{b_{2^\tau-1}}\right)\right\}_{b_0,\ldots,b_{2^\tau-1} \in \{0,1\}}$$

and computing the binary coefficients (in clear) to be used to reconstruct each bit of the $2^{\lceil \log \ell_k \rceil - \tau}$ intermediate variables of depth $\tau$ from the dictionaries through linear (homomorphic) combinations. By assumption, this accounts for nothing in the total computation time. The rest of the binary tree is then performed using generic encrypted field multiplications as before, until the circuit output is fully aggregated. This approach is always more efficient than the straightforward implementation and optimal when the total number

$$\left(2^{2^\tau} - 2^\tau - 1\right) \cdot 2^{\lceil \log \ell_k \rceil - \tau} + \left(2^{\lceil \log \ell_k \rceil - \tau - 1} - 1\right) \cdot n$$

of required homomorphic bit multiplications is minimal. With $\ell_k = 160$ and $n = 16000$ again, the best choice is for $\tau = 4$. Assuming 0.5 seconds for each bit multiplication, this still gives a prohibitive 6.71 days of computation for a single evaluation of $\mathcal{C}_{\text{exp}}$.

## 6 Conclusion

Our work shows that the promising performances obtained by the recently proposed HE-dedicated cipher LowMC can also be achieved with Trivium, a well-known primitive whose security has been thoroughly analyzed, *e.g.* [MB07,DS09], and [ADMS09,FV13,KMN11]. The 10-year analysis effort from the whole community, initiated by the eSTREAM competition, enables us to gain confidence in its security. Also our variant Kreyvium, with a 128-bit security, benefits from the same analysis since the core of the cipher is essentially the same.

From a more fundamental perspective, one may wonder how many multiplicative levels are *strictly necessary* to achieve a secure compressed encryption scheme, irrespective of any performance metric such as the number of homomorphic bit multiplications to perform in the decompression circuit. We already know that a multiplicative depth of $\lceil \log \kappa \rceil + 1$ is achievable for $\kappa$-bit security. Can one do better or prove that this is a lower bound?

Also, the provable security of a KEM-DEM construct where the KEM is homomorphic remains an open question. In particular, assuming the KEM part is just IND-CPA, what would be the minimum security requirements expected from the DEM part to yield an IND-CPA construction?

## References

[ADMS09] Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In *FSE 2009*, volume 5665 of *LNCS*, pages 1–22. Springer, 2009.

[AGKS05] Masayuki Abe, Rosario Gennaro, Kaoru Kurosawa, and Victor Shoup. Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of kurosawa-desmedt KEM. In *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 128–146. Springer, 2005.

[AM15] Frederik Armknecht and Vasily Mikhalev. On lightweight stream ciphers with shorter internal states. In *FSE 2015*, LNCS. Springer, 2015. To appear.

[AMFF$^+$13] Carlos Aguilar-Melchor, Simon Fau, Caroline Fontaine, Guy Gogniat, and Renaud Sirdey. Recent advances in homomorphic encryption: a possible future for signal processing in the encrypted domain. *IEEE Signal Processing Magazine*, 30(2):108–117, 2013.

[AMOR14] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodríguez-Henríquez. Computing discrete logarithms in $\mathbb{F}_{3^{6*137}}$ using Magma. *IACR Cryptology ePrint Archive*, 2014:57, 2014.

[ARS$^+$15] Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015*, LNCS. Springer, 2015. To appear.

[Bab95] Steve Babbage. A space/time trade-off in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, number 408. IEEE Conference Publication, 1995.

[BCG$^+$12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012.

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

[BG07] Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In *FSE 2007*, volume 4593 of *LNCS*, pages 254–273. Springer, 2007.

[BGJT14] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 1–16. Springer, 2014.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *TOCT 2014*, 6(3):13, 2014.

[Bod07] Marco Bodrato. Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In *WAIFI 2007*, volume 4547 of *LNCS*, pages 116–133. Springer, 2007.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, 2000.

[CCK$^+$13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 315–335. Springer, 2013.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.

[CDS15] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: a compilation chain for privacy preserving applications. In *ACM Symposium on Information, Computer and Communications Security (3rd International Workshop on Security in Cloud Computing)*, 2015.

[CLP05] Christophe De Cannière, Joseph Lano, and Bart Preneel. Comments on the rediscovery of time memory data tradeoffs. Technical report, eSTREAM - ECRYPT Stream Cipher Project, 2005. `https://www.cosic.esat.kuleuven.be/publications/article-595.pdf`.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In *PKC 2014*, volume 8383 of *LNCS*, pages 311–328. Springer, 2014.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 244–266. Springer, 2008.

[CT14] Massimo Chenal and Qiang Tang. On key recovery attacks against existing somewhat homomorphic encryption schemes. *IACR Cryptology ePrint Archive*, 2014:535, 2014. To appear at LATINCRYPT 2014.

[DHS14] Yarkin Doröz, Yin Hu, and Berk Sunar. Homomorphic AES evaluation using NTRU. *IACR Cryptology ePrint Archive*, 2014:39, 2014.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.

[DSES14] Yarkin Doröz, Aria Shahverdi, Thomas Eisenbarth, and Berk Sunar. Toward practical homomorphic evaluation of block ciphers using prince. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *WAHC 2014*, volume 8438 of *LNCS*, pages 208–220. Springer, 2014.

[ECR05] ECRYPT - European Network of Excellence in Cryptology. The eSTREAM Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`, 2005.

[FM14] Thomas Fuhr and Brice Minaud. Match Box Meet-in-the-Middle Attack against KATAN. In *FSE 2014*, LNCS. Springer, 2014. To appear.

[FSF$^+$13] Simon Fau, Renaud Sirdey, Caroline Fontaine, Carlos Aguilar, and Guy Gogniat. Towards practical program execution over fully homomorphic encryption schemes. In *Proceedings of the 8th IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 284–290, 2013.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[FV13] Pierre-Alain Fouque and Thomas Vannet. Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks. In *FSE 2013*, volume 8424 of *LNCS*, pages 502–517. Springer, 2013.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC 2009*, pages 169–178. ACM, 2009.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, 2012.

[GKZ14] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. Breaking '128-bit secure' supersingular binary curves - (or how to solve discrete logarithms in $\mathbb{F}_{2^{4\cdot 1223}}$ and $\mathbb{F}_{2^{12\cdot 367}}$). In *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 126–145. Springer, 2014.

[GLN12] Thore Graepel, Kristin E. Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *ICISC 2012*, volume 7839 of *LNCS*, pages 1–21. Springer, 2012.

[GLSV14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *FSE 2014*, LNCS. Springer, 2014. To appear.

[Gol97] Jovan Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 239–255. Springer-Verlag, 1997.

[HK07] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In *CRYPTO 2007*, volume 4622 of *LNCS*, pages 553–571. Springer, 2007.

[HS05] Jin Hong and Palash Sarkar. New applications of time memory data tradeoffs. In *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 353–372. Springer, 2005.

[Iwa06] Tetsu Iwata. New blockcipher modes of operation with beyond the birthday bound security. In *FSE 2006*, volume 4047 of *LNCS*, pages 310–327. Springer, 2006.

[JP14] Antoine Joux and Cécile Pierrot. Improving the Polynomial time Precomputation of Frobenius Representation Discrete Logarithm Algorithms - Simplified Setting for Small Characteristic Finite Fields. In *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 378–397. Springer, 2014.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* Chapman and Hall/CRC Press, 2014.

[KMN10] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of nlfsr-based cryptosystems. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 130–145. Springer, 2010.

[KMN11] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of Trivium and KATAN. In *SAC 2011*, volume 7118 of *LNCS*, pages 200–212. Springer, 2011.

[LLN14] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private computation on encrypted genomic data. In *LATINCRYPT 2014*, LNCS, 2014.

[LN14] Tancrède Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In *AFRICACRYPT 2014*, volume 8469 of *LNCS*, pages 318–335. Springer, 2014.

[LST12] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable Blockciphers with Beyond Birthday-Bound Security. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 14–30. Springer, 2012.

[MB07] Alexander Maximov and Alex Biryukov. Two trivial attacks on Trivium. In *SAC 2007*, volume 4876, pages 36–55. Springer, 2007.

[Nat01] National Institute of Standards and Technology. NIST Special Publication 800-38A — Recommendation for Block Cipher Modes of Operation. NIST Special Publication 800-38A, 2001.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In Christian Cachin and Thomas Ristenpart, editors, *ACM CCSW 2011*, pages 113–124. ACM, 2011.

[Pin89] Antonio Pincin. A new algorithm for multiplication in finite fields. *IEEE Transactions on Computers*, 38(7):1045–1049, 1989.

[Rog11] Phillip Rogaway. Evaluation of some blockcipher modes of operation. Cryptrec, 2011. `http://web.cs.ucdavis.edu/~rogaway/papers/modes.pdf`.

[Yas11] Kan Yasuda. A New Variant of PMAC: Beyond the Birthday Bound. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 596–609. Springer, 2011.

## A Which quantity must be encrypted under the HE?

In order to limit the multiplicative depth of the decryption circuit, we may prefer to transmit a longer secret $\tilde{k}$, from which more calculations can be done at a small multiplicative depth. Typically, for a block cipher, the sequence formed by all round-keys can be transmitted to the server. In this case, the key scheduling does not have to be taken into account in the homomorphic evaluation of the decryption function. Similarly, stream ciphers offer several such trade-offs between the encryption rate and the encryption throughput. The encryption rate, *i.e.*, the ratio between the size of $c' = (\mathsf{HE}_{\mathsf{pk}}(k), \mathsf{E}_k(m))$ and the plaintext size $\ell_m$, is defined as

$$\rho = \frac{|c'|}{\ell_m} = \frac{|E_k(m)|}{\ell_m} + \frac{|\tilde{k}| \times (\text{HE expansion rate})}{\ell_m} \, .$$

The extremal situation obviously corresponds to the case where the message encrypted under the homomorphic scheme is sent directly, *i.e.*, $c' = \mathsf{HE}_{\mathsf{pk}}(m)$. The multiplicative depth here is 0, as no decryption needs to be performed. In this case, $\rho$ corresponds to the HE expansion rate.

The following alternative scenarios can then be compared.

1. Only the secret key is encrypted under the homomorphic scheme, *i.e.*, $\tilde{k} = k$. Then, since we focus on symmetric encryption schemes with rate 1, we get

$$\rho = 1 + \frac{\ell_k \times (\text{HE expansion rate})}{\ell_m}$$

which is the smallest encryption rate we can achieve for an $\ell_k$-bit security. In a nonce-based stream cipher, $\ell_m$ is limited by the IV size $\ell_{IV}$ and by the maximal keystream length $N(d)$ which can be produced for a fixed multiplicative depth $d \geq \text{depth}(\mathsf{Sync}) + \text{depth}(f)$. Then, the minimal encryption rate is achieved for messages of any length $\ell_m \leq 2^{\ell_{IV}} N(d)$.

2. An intermediate case consists in transmitting the initial state of the generator, *i.e.*, the output of $\mathsf{Sync}$. Then, the number of bits to be encrypted by the HE increases to the size $n$ of the internal state, while the number of keystream bits which can be generated from a given initial state with a circuit of depth $d$ corresponds to $N(d + \text{depth}(\mathsf{Sync}))$. Then, we get

$$\rho = 1 + \frac{n \times (\text{HE expansion rate})}{N(d + \text{depth}(\mathsf{Sync}))} \, ,$$

for any message length. The size of the internal state is at least twice the size of the key. Therefore, this scenario is not interesting, unless the number of plaintext bits $\ell_m$ to be encrypted under the same key is smaller than twice $N(d + \text{depth}(\mathsf{Sync}))$.

## B Proofs of Propositions 1 and 2

We first observe that, within any register in Trivium, the degree of the leftmost bit is greater than or equal to the degrees of the other bits in the register. It is then sufficient to study the evolution of the leftmost bits in the three registers. Let $t_i(d)$ denotes the first time instant (starting from $t = 1$) where the leftmost bit in Register $i$ is computed by a circuit of depth $d$. The depth of the feedback bit in Register $i$ can increase from $d$ to $(d+1)$ if either a bit of depth $(d+1)$ reaches a XOR gate in the feedback function, or a bit of depth $d$ reaches one of the inputs of the AND gate. From the distance between the leftmost bit and the first bit involved in the feedback (resp. and the first entry of the AND gate) in each register, we derive that

$$\begin{aligned}
t_1(d+1) &= \min(t_3(d+1) + 66, t_3(d) + 109) \\
t_2(d+1) &= \min(t_1(d+1) + 66, t_1(d) + 91) \\
t_3(d+1) &= \min(t_2(d+1) + 69, t_2(d) + 82)
\end{aligned}$$

In Trivium, the first key bits $K_{78}$ and $K_{79}$ enter the AND gate in Register 1 at time $t = 13$ (starting from $t = 1$), implying $t_2(1) = 14$. Then, $t_3(1) = 83$ and $t_1(1) = 149$. This leads to

$$t_1(4) = 401, t_2(4) = 296 \text{ and } t_3(4) = 335 \; .$$

From $d = 3$, the differences $t_i[d+1] - t_i[d]$ are large enough so that the minimum in the three recurrence relation corresponds to the right-hand term. We then deduce that, for $d \geq 4$,

- if $d \equiv 1 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-1)}{3} + 119,\; t_2(d) = 282 \times \frac{(d-1)}{3} + 14,\; t_3(d) = 282 \times \frac{(d-1)}{3} + 53.$$

- if $d \equiv 2 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-2)}{3} + 162,\; t_2(d) = 282 \times \frac{(d-2)}{3} + 210,\; t_3(d) = 282 \times \frac{(d-2)}{3} + 96.$$

- if $d \equiv 0 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-3)}{3} + 205,\; t_2(d) = 282 \times \frac{(d-3)}{3} + 253,\; t_3(d) = 282 \times \frac{(d-3)}{3} + 292.$$

The degree of the keystream produced at time $t$ corresponds to the minimum between the degrees of the bit at position 66 in Register 1, the bit at position 69 in Register 2 and the bit at position 66 in Register 3. Then, for $d > 3$,

$$N(d) = \min(t_1(d+1) + 64, t_2(d+1) + 67, t_3(d+1) + 64) \; .$$

This leads to, for any $d \geq 4$,

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 81 & \text{if } d \equiv 0 \bmod 3 \\ 160 & \text{if } d \equiv 1 \bmod 3 \\ 269 & \text{if } d \equiv 2 \bmod 3 \end{cases} .$$

In Kreyvium, the recurrence relations defining the $t_i(d)$ are the same. The only difference is that the first key bits now enter the AND gate in Register 1 at time $t = 1$, implying $t_2(1) = 2$. Then, $t_3(1) = 71$, $t_1(1) = 137$ and $t_3[2] = 85$. The situation is then similar to Trivium, except that we start from

$$t_1(4) = 390, t_2(4) = 285 \text{ and } t_3(4) = 324 \ .$$

These three values are equal to the values obtained with Trivium minus 11. This fixed difference then propagated, leading to, for any $d \geq 4$,

– if $d \equiv 1 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-1)}{3} + 108,\ t_2(d) = 282 \times \frac{(d-1)}{3} + 3,\ t_3(d) = 282 \times \frac{(d-1)}{3} + 42.$$

– if $d \equiv 2 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-2)}{3} + 151,\ t_2(d) = 282 \times \frac{(d-2)}{3} + 199,\ t_3(d) = 282 \times \frac{(d-2)}{3} + 85.$$

– if $d \equiv 0 \bmod 3$,

$$t_1(d) = 282 \times \frac{(d-3)}{3} + 194,\ t_2(d) = 282 \times \frac{(d-3)}{3} + 242,\ t_3(d) = 282 \times \frac{(d-3)}{3} + 281.$$

We eventually derive that, for Kreyvium, for any $d \geq 4$,

$$N(d) = 282 \times \left\lfloor \frac{d}{3} \right\rfloor + \begin{cases} 70 & \text{if } d \equiv 0 \bmod 3 \\ 149 & \text{if } d \equiv 1 \bmod 3 \\ 258 & \text{if } d \equiv 2 \bmod 3 \end{cases} .$$

## C Deeper Analysis of Trivium, Kreyvium and LowMC circuits

**Table 4.** Number of AND and XOR gates to homomorphically evaluate in Trivium, Kreyvium and LowMC

| Algorithm | $\lambda$ | # ANDs | # XORs | × depth | # key stream |
|---|---|---|---|---|---|
| Trivium-12 | 80 | 3422 | 10459 | 12 | 57 bits |
| Trivium-13 | 80 | 3659 | 11328 | 13 | 136 bits |
| LowMC-80 | 80 | 2805 | 369056 | 11 | 256 bits |
| Kreyvium-12 | 128 | 3401 | 11989 | 12 | 46 bits |
| Kreyvium-13 | 128 | 3638 | 12957 | 13 | 125 bits |
| LowMC-128 | 128 | 3060 | 402641 | 12 | 256 bits |

**Trivium.** In order to produce 57 bits of keystream using a circuit of multiplicative depth 12, the boolean circuit of Trivium contains 3627 AND gates (with 3422 to homomorphically evaluate), and 10995 XOR gates (with 10459 to homomorphically evaluate).

**Kreyvium.** To generate the 46 bits of keystream using a circuit of multiplicative depth 12, the boolean circuit of Kreyvium contains 3594 AND gates (with 3401 to homomorphically evaluate), and 13270 XOR gates (with 11989 to homomorphically evaluate). The resulting circuit has 72 nicely balanced layers (with around 250 gates per layer).

**LowMC.** The LowMC block cipher has been designed with FHE-friendliness in mind [ARS$^+$15]. After an initial XORing of the plaintext block with an initial (secret) key, each round consists in applying a multiplicative depth-1 substitution box, multiplying (in $\mathbb{F}_2$) the current block by a (public) round matrix followed by XORing a (public) round constant to the result and then further XORing a (secret) round key to it. Since the matrix involved in the multiplication is public, the matrix multiplication can be performed by means of the simple hybrid cleartext/ciphertext calculation rules defined in Sec. 3.1 and does not require any homomorphic multiplication.

LowMC-80 can generate 256 bits of keystream (with a depth-11 circuit). Its boolean circuit has 723701 AND gates (with *only* 2805 to homomorphically evaluate) and 732394 XOR gates (with 369056 to homomorphically evaluate).[10] From a Boolean circuit point of view, this leads to 2873 layers, most of them having the width of the block size (excepted for the layers associated to the Sbox).

LowMC-128 can generate 256 bits of keystream (with a depth-12 circuit). The structure of this algorithm is the same as that of the previous variant. Its boolean circuit has 789492 AND gates (with *only* 3060 to homomorphically evaluate) and 798952 XOR gates (with 402641 to homomorphically evaluate).

[10] It is a characteristic of the LowMC family to have a very high number of XOR gates. Note that their homomorphic evaluation ends up being a significant contributor in the algorithm running time.