

Hash Function Constructions from Lightweight Block Ciphers for Fully Homomorphic Encryption

Olivier Bernard
Zama
Paris, France
olivier.bernard@zama.ai

Marc Joye
Zama
Paris, France
marc@zama.ai

Abstract

This paper investigates hash-function constructions derived from lightweight block ciphers, that are suitable for evaluation in fully homomorphic encryption (FHE) settings. We focus on PRINCEv2, a 64-bit lightweight block cipher with 128-bit keys and low algebraic complexity, which is particularly amenable to FHE evaluation. However, the small block size of such ciphers limits the applicability of standard hash-function transforms. Indeed, achieving 128-bit collision resistance in the $(n, 2n)$ setting, i.e., with 64-bit blocks, requires a quadruple-block-length (QBL) compression function, for which no generic construction is known.

In this work, we propose a concrete QBL compression construction tailored to PRINCEv2 and analyze its collision resistance. Candidate QBL designs inspired from existing double-block-length constructions are also outlined. As a further contribution, we describe a carefully optimized homomorphic circuit design for PRINCEv2. The resulting implementation outperforms previous works in both operation counts and computational depth. Experimental timings demonstrate the practical feasibility of evaluating the corresponding hash constructions under FHE with low latency, while providing cryptographically small failure probability.

CCS Concepts

• Security and privacy → Hash functions and message authentication codes.

Keywords

Lightweight block ciphers, PRINCEv2, Iterated hash functions, Quadruple-block-length (QBL) compression, Fully homomorphic encryption (FHE)

ACM Reference Format:

Olivier Bernard and Marc Joye. 2026. Hash Function Constructions from Lightweight Block Ciphers for Fully Homomorphic Encryption. In *14th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC '26)*, November 15–19, 2026, The Hague, Netherlands. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3847161.3858424>

1 Introduction

Cryptographic hash functions are fundamental primitives used in a wide range of applications, including digital signatures, message authentication, and integrity verification. Modern hash functions are

typically designed to process messages of arbitrary length by iterating a fixed-length compression function. Classical constructions such as the Merkle–Damgård paradigm [22, 33] achieve this by repeatedly applying a block-cipher-based or dedicated compression function, starting from a fixed initialization vector. The security and efficiency of the resulting hash function depend heavily on the properties of the underlying primitive.

In parallel, lightweight cryptography has received significant attention in recent years, driven by the need for secure primitives in constrained environments such as embedded devices, RFID tags, and the Internet of Things (IoT). To address stringent performance requirements, block ciphers like PRESENT [11], SIMON and SPECK [5], SKINNY/MANTIS [6], and PRINCE [12] have been proposed. PRINCE in particular is notable for its ultra-low latency in hardware, achieved through an involutive structure and a carefully crafted key schedule. Its successor, PRINCEv2 [13], improves security margins while preserving efficiency, making it a strong candidate for exploration as a building block in hash-function design.

Beyond constrained hardware, such structural simplicity is also highly desirable in fully homomorphic encryption (FHE) settings [24, 34], where the cost of evaluating cryptographic primitives is dominated by algebraic depth and circuit regularity. Recent work has surveyed the homomorphic evaluation of a range of block ciphers, taking AES as a baseline for comparison, and reported that PRINCE exhibits particularly favorable evaluation characteristics among the lightweight designs considered [38]. Meanwhile, while block-cipher evaluation under FHE has received increasing attention, comparatively little work has addressed the construction of hash functions that are both secure and efficiently evaluable in FHE. An overview of applications of hash functions in FHE-based protocols can be found in [8, Section 3].

These considerations naturally raise the question of how to construct secure hash functions from lightweight block ciphers. A central obstacle in this setting is the small block size of designs such as PRINCEv2. Generic transforms, such as Hirose’s double-block-length construction [26, 27], Nandi’s generalized constructions [16], or the Counter-*b*DM construction of Abed et al. [1], show how compression functions can be derived from block ciphers. In particular, achieving 128-bit collision resistance from a 64-bit block cipher generically requires a quadruple-block-length (QBL) compression function. However, to the best of our knowledge, no generic QBL construction is known for block ciphers with an n -bit block size and a $2n$ -bit key, and existing constructions are incompatible with lightweight designs such as PRINCEv2.



This work is licensed under a Creative Commons Attribution 4.0 International License. WAHC '26, The Hague, Netherlands

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-3022-1/2026/11

<https://doi.org/10.1145/3847161.3858424>

Our techniques and contributions. This work combines design-level modifications, security analysis, and implementation techniques to enable efficient homomorphic hashing based on lightweight $(n, 2n)$ block ciphers.

At the construction level, we propose several QBL compression functions tailored to $(n, 2n)$ block ciphers. Our main construction builds on the Counter-*b*DM framework by extending the effective key space of PRINCEv2 and by carefully selecting counter constants to mitigate internal collisions. Unlike Counter-4DM, this choice avoids structural cancellations that degrade collision resistance. To efficiently support the resulting key extension, we introduce a family of linear orthomorphisms designed to align with nibble-based state representations, making them particularly well suited for FHE evaluation.

From a security perspective, we analyze the collision resistance of the resulting compression function in the ideal-cipher model [10]. As in [1], using the super-query technique [2], we derive explicit bounds that separate the contributions of normal queries, super-queries, and internal collisions. Our analysis highlights the concrete impact of constant selection and shows that the modified construction leads to better concrete bounds than Counter-4DM.

On the implementation side, we present a highly optimized FHE implementation of PRINCEv2 using the TFHE scheme [20], establishing the practical feasibility of homomorphic hashing based on lightweight ciphers. The implementation leverages the nibble-oriented structure of PRINCEv2 and introduces a hybrid LUT strategy that exploits flexible output formats, allowing bit-level extraction and recombination within programmable bootstrapping, to optimally interface the successive cipher layers. This approach significantly reduces both the computational depth and the number of bootstrappings, which are the dominant cost factors in TFHE-based evaluation. As a result, efficient homomorphic evaluation of PRINCEv2 is achieved with sub-second latency, while maintaining a cryptographically small failure probability of 2^{-128} and 128-bit security. Our results indicate that the proposed hash constructions can be efficiently evaluated under FHE.

2 Multi-Block-Length Compression

An *iterated hash function* is a standard method for building a cryptographic hash function that can process inputs of arbitrary length by repeatedly applying a fixed-length compression.

For simplicity, we assume that the input message M is divided into ℓ -bit blocks $m_1, m_2, \dots, m_l$. In practice, message lengths are not always exact multiples of the block size; to handle arbitrary input lengths securely, the message is first padded so that it becomes a multiple of ℓ , and the original length of the message is appended as part of the padding. This approach, formalized in the Merkle-Damgård construction [22, 33], ensures that different messages cannot collide due to ambiguous block boundaries.

We start from a fixed initial value h_0 , called the initialization vector (IV). The chaining values are then computed successively as

$$h_i \leftarrow F(h_{i-1}, m_i), \quad 1 \leq i \leq l,$$

where F is a *compression function* that combines the previous chaining value with the current message block. After the last block has been processed, the final chaining value h_l is defined as the hash of the message; i.e., $H(M) = h_l$.

2.1 Hirose's Transform

In [26, 27], Hirose proposes a transform, called Hirose-DM, for constructing a *double-block-length compression function* from a block cipher.

Let $E_K(\cdot)$ denote an n -bit block cipher with key length $(n + \ell)$. Given a chaining value $(h_{i-1}^{(1)}, h_{i-1}^{(2)})$ with $h_{i-1}^{(1)}, h_{i-1}^{(2)} \in \{0, 1\}^n$ and a message block $m_i \in \{0, 1\}^\ell$, the compression function produces a new chaining value $(h_i^{(1)}, h_i^{(2)})$ by computing

$$\begin{cases} h_i^{(1)} \leftarrow E_{h_{i-1}^{(2)} \| m_i}(h_{i-1}^{(1)}) \oplus h_{i-1}^{(1)} \\ h_i^{(2)} \leftarrow E_{h_{i-1}^{(2)} \| m_i}(h_{i-1}^{(1)} \oplus C) \oplus h_{i-1}^{(1)} \oplus C \end{cases}$$

where $C \in \{0, 1\}^n$ is a fixed nonzero constant.

By making two parallel block-cipher calls with distinct inputs (ensured by the constant C), the transform prevents trivial collisions. Note that although two calls are made, only a single key schedule is required.

In the ideal cipher model, Hirose proves that the above construction provides about 2^n collision resistance (the birthday bound). Other constructions of this type include Tandem-DM and Abreast-DM [31, Section 3.5], as well as Weimar-DM [23].

2.2 MDC-2 Scheme

When the block size and key length coincide, one possible approach is to use MDC-2 [14, 36] (see also ISO/IEC 10118-2 standard [28, Section 7]) to obtain a double-block-length compression function.

Given a chaining value $(h_{i-1}^{(1)}, h_{i-1}^{(2)})$ with $h_{i-1}^{(1)}, h_{i-1}^{(2)} \in \{0, 1\}^n$ and a message block $m_i \in \{0, 1\}^n$, the compression function computes a new chaining value $(h_i^{(1)}, h_i^{(2)})$ as

$$\begin{cases} T_i^{(1)} \leftarrow E_{h_{i-1}^{(1)}}(m_i) \oplus m_i = LT_i^{(1)} \| RT_i^{(1)} \\ T_i^{(2)} \leftarrow E_{h_{i-1}^{(2)}}(m_i) \oplus m_i = LT_i^{(2)} \| RT_i^{(2)} \\ h_i^{(1)} \leftarrow LT_i^{(1)} \| RT_i^{(2)} \\ h_i^{(2)} \leftarrow LT_i^{(2)} \| RT_i^{(1)} \end{cases}$$

where $LT_i^{(\cdot)}, RT_i^{(\cdot)}$ are the left and right $n/2$ -bit halves of $T_i^{(\cdot)}$.

Remark 2.1. The description in [28, Section 7] is slightly more general: some functions $u, u': \{0, 1\}^n \rightarrow \{0, 1\}^n$ are applied, respectively, to $h_{i-1}^{(1)}$ and $h_{i-1}^{(2)}$ when computing $T_i^{(1)}$ and $T_i^{(2)}$, namely

$$T_i^{(1)} \leftarrow E_{u(h_{i-1}^{(1)})}(m_i) \oplus m_i \quad \text{and} \quad T_i^{(2)} \leftarrow E_{u'(h_{i-1}^{(2)})}(m_i) \oplus m_i.$$

Remark 2.2. Although Hirose-DM and MDC-2 each invoke the block cipher twice, when applicable, Hirose-DM should be preferred because it reuses the same key for both calls, reducing key-schedule cost. In addition, the best proven collision-security bound for MDC-2 *in the iteration* is lower, namely $2^{3n/5}$ [36], however the true bound is generally believed to be close to 2^n [30].

2.3 Multi-Block-Length Compression Functions

Hirose's transform assumes an n -bit block cipher with key length $n + \ell$, and it outputs a $2n$ -bit chaining value. While this construction is suitable when $n \geq 128$, it is insufficient for smaller n (e.g., with $n = 64$ it provides only about 2^{64} collision resistance). A *multi-block-length compression function* addresses this issue by outputting bn bits for any $b \geq 2$.

To the best of our knowledge, the only multi-block-length compression functions that support $b > 2$ are due to Chang et al. [16] and to Abed et al. [1]. The construction by Abed et al., known as Counter- b DM (see also [35, Appendix C]) builds on an n -bit block cipher $E_K(\cdot)$ with a bn -bit key K .

Given a bn -bit chaining value $(h_{i-1}^{(1)}, \dots, h_{i-1}^{(b)})$ with $h_{i-1}^{(j)} \in \{0, 1\}^n$ and a message block $m_i \in \{0, 1\}^n$, define

$$K_i := h_{i-1}^{(2)} \parallel \dots \parallel h_{i-1}^{(b)} \parallel m_i \in \{0, 1\}^{bn}.$$

The next chaining value $(h_i^{(1)}, \dots, h_i^{(b)})$ is computed in parallel as

$$h_i^{(j)} \leftarrow E_{K_i}(h_{i-1}^{(1)} \oplus (j-1)) \oplus h_{i-1}^{(j)}, \quad \text{for } 1 \leq j \leq b.$$

The counter term, $j-1$, ensures that the b input plaintexts to the block cipher are pairwise distinct.

Remark 2.3. When $b = 2$, Counter- b DM is closely related to Hirose-DM with constant $C = 1$. The main difference lies in an additional post-whitening by C , which is applied to the second output block in Hirose-DM.

3 Main Construction

Recall that Counter- b DM produces bn output bits for any b , but is incompatible with lightweight $(n, 2n)$ block ciphers such as PRINCEv2. Achieving 128-bit collision resistance requires setting $b = 4$, which yields $bn = 256$. In this case, however, Counter- b DM requires a 256-bit key, whereas PRINCEv2 supports only 128-bit keys. Separately, we modify the choice of constants in Counter- b DM to limit the impact of internal collisions.

To address this incompatibility, following a similar approach used for MBLhash in [35, Appendix S2], we augment PRINCEv2 with an additional tweak input in order to emulate a larger key space without increasing the master key length. Specifically, we introduce a concrete construction that extends the effective key size to $4n$ bits by leveraging the TWEAKEY framework [29]. The design draws on techniques from the block ciphers MANTIS [6] and QARMA [3]. Moreover, with a view toward more efficient FHE evaluation and as a contribution of independent interest, we develop a new family of orthomorphisms, as used, e.g., in PRINCE [12] and QARMA [3].

We analyze the collision resistance of the proposed construction in Section 4.

3.1 Tweaking PRINCEv2

In PRINCEv2, a $2n$ -bit tweak (T_0, T_1) is provided as an additional input to the cipher, and each round key is alternately XORed with a permutation of T_0 and T_1 . As in MANTIS, the Galois-field multiplication used in the TWEAKEY framework is replaced by this permutation [6, Section 6], which is chosen as an orthomorphism¹ following the design rationale of PRINCE and QARMA.

The next proposition introduces a family of orthomorphisms that are well suited for FHE-based implementations, as the parameter k can be chosen to correspond to an integer number of nibbles.

¹For an additively written group G , a map $\varphi: G \rightarrow G$ is called an orthomorphism if both φ and $\varphi - \text{id}$ are permutations of G , where id denotes the identity map on G .

PROPOSITION 3.1. *Let $n \geq 2$ and k be integers satisfying $2 \leq 2k < n$. Then the map $\Theta_k(x) := (x \ggg k) \oplus (x \gg (n-k))$ is an orthomorphism of $\mathbb{Z}_2^n$.*

PROOF. We show that both Θ_k and $\Theta_k \oplus \text{id}$ are permutations of $\mathbb{Z}_2^n$. Fix k . Define

$$R: x \mapsto x \ggg k, \quad S: x \mapsto x \gg (n-k),$$

so that $\Theta_k = R \oplus S = R \circ (\text{id} \oplus R^{-1}S)$. The map R rotates all bits k positions to the right, and R^{-1} rotates k positions to the left. The map S inserts $n-k$ zero bits at the top. Since $n-k > k$, one can verify that after $\alpha = \lceil \frac{n-k}{n-2k} \rceil$ iterations of $R^{-1}S$, we obtain the zero map; that is, $(R^{-1}S)^\alpha = 0$. Consequently, $\text{id} \oplus R^{-1}S$ is invertible because $(\text{id} \oplus R^{-1}S) \circ (\text{id} \oplus R^{-1}S \oplus (R^{-1}S)^2 \oplus \dots \oplus (R^{-1}S)^{\alpha-1}) = \text{id} \oplus (R^{-1}S)^\alpha = \text{id}$. As R is a permutation and $\text{id} \oplus R^{-1}S$ is invertible, it follows that Θ_k is bijective, and therefore a permutation of $\mathbb{Z}_2^n$.

Write $x = (x_{n-1}, \dots, x_0) \in \mathbb{Z}_2^n$. From the definition $\Theta_k = R \oplus S$, the condition $x = \Theta_k(x)$ holds if and only if

$$x_i = \begin{cases} x_{i+k} \oplus x_{i+(n-k)} & \text{for } 0 \leq i < k \\ x_{i+k} & \text{for } k \leq i < n-k \\ x_{i-(n-k)} & \text{for } n-k \leq i < n \end{cases}. \quad (*)$$

A change of variables shows that the last relation in $(*)$ implies $x_j = x_{j+(n-k)}$ for $0 \leq j < k$. Combining this with the first relation in $(*)$ yields $x_{i+k} = 0$ for $0 \leq i < k$. Finally, the two last relations of $(*)$ can be combined into

$$x_i = x_{(i+k) \bmod n} \quad \text{for } k \leq i < n,$$

which expresses the k -periodicity of the remaining coordinates. We so have $x_k = x_{k+1} = \dots = x_{n-1} = 0$. With the first relation of $(*)$, we then have $x_i = x_{i+k} \oplus 0 = 0$ for $0 \leq i < k$. Summing up, $x = \Theta_k(x)$ if and only if x is the zero vector; i.e., $x = 0$. Hence, $\ker(\Theta_k \oplus \text{id}) = \{0\}$. Being a linear map on a finite-dimensional vector space, $\Theta_k \oplus \text{id}$ is therefore bijective, and thus a permutation of $\mathbb{Z}_2^n$. $\square$

Four additional rounds are added in PRINCEv2 to preserve a conservative security margin; see e.g., [6, Table 1]. Overall, this yields an effective $(n, 4n)$ block cipher $E_K(\cdot) = \widehat{E}_K(T_0, T_1, \cdot)$ with a key $K \in \{0, 1\}^{4n}$, composed of a tweak $(T_0, T_1) \in \{0, 1\}^{2n}$ and a cipher key $\mathfrak{K} \in \{0, 1\}^{2n}$.

3.2 A Modified Compression Function

We consider a modified compression function based on our tweak-augmented variant of PRINCEv2, instantiating Counter- b DM with $b = 4$ and achieving improved collision-resistance bounds (see Section 4.4).

Concretely, the compression function is defined as follows. For $1 \leq j \leq 4$ and appropriately defined, pairwise distinct constants C_j ,

$$\begin{aligned} h_i^{(j)} &\leftarrow \widehat{E}_{h_{i-1}^{(4)} \parallel m_i} (h_{i-1}^{(2)}, h_{i-1}^{(3)}, h_{i-1}^{(1)} \oplus C_{j-1}) \oplus h_{i-1}^{(1)}, \\ &= E_{K_i} (h_{i-1}^{(1)} \oplus C_{j-1}) \oplus h_{i-1}^{(1)}, \end{aligned}$$

where $K_i \leftarrow h_{i-1}^{(2)} \parallel h_{i-1}^{(3)} \parallel h_{i-1}^{(4)} \parallel m_i$. This is depicted in Fig. 1.

A key distinction from the Counter-4DM compression function lies in the choice of these constants. In Counter-4DM, they are fixed as $C_{j-1} = j-1$, which satisfies $\bigoplus_{j=1}^4 (j-1) = 0$ and gives rise

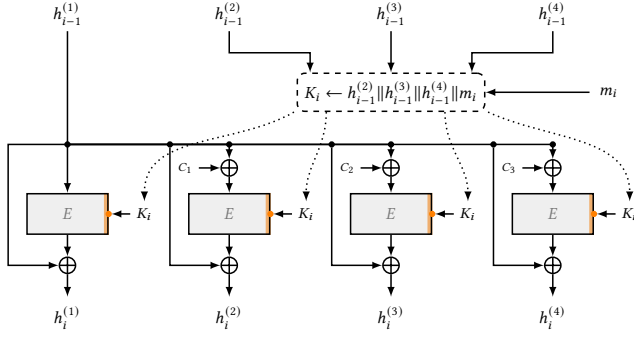


Figure 1: Modified compression function.

to structural cancellations. In contrast, our construction employs constants C_{j-1} whose XOR does not vanish. For efficiency, we set $C_0 = 0$ and use pairwise distinct non-zero constants $C_1, C_2, C_3 \in \{0, 1\}^n$ satisfying

$$C_1 \oplus C_2 \oplus C_3 \neq 0.$$

For example, one may set $C_i = 2^i$ for $1 \leq i \leq 3$.

4 Analysis

4.1 Security Model

The analysis is carried out in the ideal-cipher model, in which the block cipher $E_K(\cdot)$ is instantiated as a random permutation. Specifically, for every key K , an independent random permutation $E_K(\cdot)$ is chosen [10]. The adversary is given oracle access to E and may issue both forward and inverse queries: on input (K, X) , the oracle returns $Y = E_K(X)$, and on input (K, Y) , it returns $X = E_K^{-1}(Y)$. The oracle maintains a list $\mathcal{Q}$ of triples (X, K, Y) . If a query matches a triple already in the list, the corresponding value is returned. As is customary, we therefore assume, without loss of generality, that the adversary never issues a query for which it already knows the output.

Furthermore, as in [1], to accommodate a number of queries close to or exceeding 2^n , where n denotes the block size, we employ the *super-query* technique of [2]. For example, when applied to our main construction, if the adversary issues an encryption query $E_K(X)$, it also receives *for free* the values $E_K(X \oplus C_b)$ for several given constants C_b , and analogously for decryption queries. In all cases, a single (normal) query generates several entries in $\mathcal{Q}$. Moreover, if the query list $\mathcal{Q}$ exceeds 2^{n-1} entries for a fixed key K , the adversary is additionally given *for free* all remaining input-output pairs (X, Y) for that key (i.e., $Y = E_K(X)$ for all previously unseen X).

Since an adversary with a budget of q queries receives additional input-output pairs *for free* in the super-query model, its advantage can only increase. We may therefore analyze the adversary's advantage in the super-query model to obtain an upper bound on the advantage in the standard query model.

4.2 Collision Resistance

Let $\mathbf{B}(n, k)$ denote the set of all block ciphers with block size n and key length k . For a block-cipher-based hash function H , the collision-finding advantage of an adversary $\mathcal{A}$ against H is defined

as

$$\text{Adv}_H^{\text{coll}}(\mathcal{A}) = \Pr \left[E \xleftarrow{\$} \mathbf{B}(n, k), (M, M') \leftarrow \mathcal{A}^{E, E^{-1}} : M \neq M' \wedge H(M) = H(M') \right].$$

The complexity of an adversary is measured by the total number of queries it issues to the encryption oracle E or the decryption oracle E^{-1} . For $q \geq 1$, we define

$$\text{Adv}_H^{\text{coll}}(q) := \max_{\mathcal{A}} \text{Adv}_H^{\text{coll}}(\mathcal{A}),$$

where the maximum is taken over all adversaries that make at most q oracle queries in total.

4.3 Security Proof

Let $\mathcal{A}$ be an adversary as defined in Section 4.1. The encryption/decryption oracle maintains a query list $\mathcal{Q}$ of triples (X_i, K_i, Y_i) , where $Y_i = E_{K_i}(X_i)$, initially set to $\emptyset$. The adversary is allowed to make at most q encryption or decryption queries and maintains a list $\mathcal{L}$ of (sets of) compression pairs (U, V) , where $V = F(U)$ for the compression function

$$F: \{0, 1\}^{4n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{4n}, U = (K, X) \mapsto V = (V_1, V_2, V_3, V_4),$$

defined by

$$\begin{cases} V_1 = E_K(X) \oplus X \\ V_2 = E_K(X \oplus C_1) \oplus X \\ V_3 = E_K(X \oplus C_2) \oplus X \\ V_4 = E_K(X \oplus C_3) \oplus X \end{cases},$$

where $C_1, C_2, C_3 \in \{0, 1\}^n$ satisfy $C_1 \oplus C_2 \oplus C_3 \neq 0$. For convenience, define $C_0 = 0$, $C_4 = C_1 \oplus C_2$, $C_5 = C_1 \oplus C_3$, $C_6 = C_2 \oplus C_3$, and $C_7 = C_1 \oplus C_2 \oplus C_3$. Define further the set

$$\begin{aligned} \mathfrak{S} &= \{C_0 = 0, C_1, C_2, C_3, C_4, C_5, C_6, C_7\} \\ &= \{\delta_1 C_1 \oplus \delta_2 C_2 \oplus \delta_3 C_3 \mid \delta_1, \delta_2, \delta_3 \in \{0, 1\}\}. \end{aligned}$$

The goal of $\mathcal{A}$ is to find collisions in the compression function F , that is, to identify two distinct pairs (K, X) and $(K', X') \in \{0, 1\}^{4n} \times \{0, 1\}^n$ such that

$$F(K, X) = F(K', X'),$$

while making at most q queries to E or E^{-1} . The list $\mathcal{L}$ is initially empty.

Whenever $\mathcal{A}$ issues a query (K_i, X_i) to the encryption oracle E (resp., (K_i, Y_i) to the decryption oracle E^{-1}), it receives $Y_i = E_{K_i}(X_i)$ (resp., $X_i = E_{K_i}^{-1}(Y_i)$), and the corresponding triple (X_i, K_i, Y_i) is recorded in the query list $\mathcal{Q}$. In accordance with the super-query model, $\mathcal{A}$ additionally receives *for free* the 7 values $E_{K_i}(X_i \oplus C_{j-1})$ for $2 \leq j \leq 8$, and the associated triples

$$(X_i \oplus C_{j-1}, K_i, E_{K_i}(X_i \oplus C_{j-1}))$$

are also added to $\mathcal{Q}$. From these eight responses, $\mathcal{A}$ can derive a set $\mathbf{L}_i$ composed of exactly eight valid compression pairs, referred to as a *compression bundle*,

$$\mathbf{L}_i = \{(K_i, X_i \oplus C_{j-1}), F(K_i, X_i \oplus C_{j-1})\}_{1 \leq j \leq 8},$$

where

$$\begin{aligned}
F(K_i, X_i \oplus C_0) &= V_{i,1} \parallel V_{i,2} \parallel V_{i,3} \parallel V_{i,4} \\
F(K_i, X_i \oplus C_1) &= (V_{i,2} \oplus C_1) \parallel (V_{i,1} \oplus C_1) \parallel (V_{i,5} \oplus C_1) \parallel (V_{i,6} \oplus C_1) \\
F(K_i, X_i \oplus C_2) &= (V_{i,3} \oplus C_2) \parallel (V_{i,5} \oplus C_2) \parallel (V_{i,1} \oplus C_2) \parallel (V_{i,7} \oplus C_2) \\
F(K_i, X_i \oplus C_3) &= (V_{i,4} \oplus C_3) \parallel (V_{i,6} \oplus C_3) \parallel (V_{i,7} \oplus C_3) \parallel (V_{i,1} \oplus C_3) \\
F(K_i, X_i \oplus C_4) &= (V_{i,5} \oplus C_4) \parallel (V_{i,3} \oplus C_4) \parallel (V_{i,2} \oplus C_4) \parallel (V_{i,8} \oplus C_4) \\
F(K_i, X_i \oplus C_5) &= (V_{i,6} \oplus C_5) \parallel (V_{i,4} \oplus C_5) \parallel (V_{i,8} \oplus C_5) \parallel (V_{i,2} \oplus C_5) \\
F(K_i, X_i \oplus C_6) &= (V_{i,7} \oplus C_6) \parallel (V_{i,8} \oplus C_6) \parallel (V_{i,4} \oplus C_6) \parallel (V_{i,3} \oplus C_6) \\
F(K_i, X_i \oplus C_7) &= (V_{i,8} \oplus C_7) \parallel (V_{i,7} \oplus C_7) \parallel (V_{i,6} \oplus C_7) \parallel (V_{i,5} \oplus C_7)
\end{aligned}$$

with $V_{i,j} = E_{K_i}(X_i \oplus C_{j-1}) \oplus X_i$. These compression pairs are subsequently added to the list $\mathcal{L}$; i.e., $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_i$. Note that, by construction, $\#Q$ is always a multiple of 8.

If, for a given key K_i , the number of entries of the form $(\cdot, K_i, \cdot)$ in Q exceeds $N/2$, where $N = 2^n$, then $\mathcal{A}$ receives *for free* all remaining triples involving the key K_i . All such triples are added to Q . $\mathcal{A}$ derives the set $\mathcal{L}_i$ consisting of the corresponding $(N/2)/8 = N/16$ compression bundles (totalling $N/2$ compression pairs), which are all added to $\mathcal{L}$. In this case, we say that a *super-query* occurs.

We are now ready to upper-bound the adversary's advantage in successfully finding a collision in the compression function F . Collisions can be external or internal.

4.3.1 External collisions. At query $\#i$, $\mathcal{A}$ obtains $\mathcal{L}_i \in \mathcal{L}$, which consists of

- a single compression bundle containing eight additional compression pairs if the query is normal, and
- a set of $N/16$ compression bundles, corresponding to a total of $N/2$ additional compression pairs, if the query is a super-query.

These correspond to the NormalQueryWin and SuperQueryWin events in the terminology of [1].

NormalQueryWin In this case, $\mathcal{A}$ finds a collision between at least one compression pair derived from the current normal query and compression pairs obtained from previous queries.

Fix $i \in \llbracket 2, q \rrbracket$. We consider the event C_i that at least one of the eight compression pairs obtained at normal query $\#i$ collides with a compression pair obtained at an earlier query $\#j$; more precisely, that there exist a pair $((K, X), F(K, X)) \in \mathcal{L}_i$ and a pair $((K', X'), F(K', X')) \in \mathcal{L}_j$ for some $j < i$ such that $F(K, X) = F(K', X')$. A collision $F(K, X) = F(K', X')$ occurs if and only if

$$\begin{cases} E_K(X) \oplus X = E_{K'}(X') \oplus X' \\ E_K(X \oplus C_1) \oplus X = E_{K'}(X' \oplus C_1) \oplus X' \\ E_K(X \oplus C_2) \oplus X = E_{K'}(X' \oplus C_2) \oplus X' \\ E_K(X \oplus C_3) \oplus X = E_{K'}(X' \oplus C_3) \oplus X' \end{cases} \quad (1)$$

We distinguish two cases:

- If $K = K'$, then query $\#j$ cannot be a super-query. Furthermore, since $\mathcal{L}_i \not\subseteq \mathcal{L}_j$ because all compression pairs are assumed to be distinct, one must have $X \oplus X' \notin \mathcal{S} \cup \{0\}$. As a consequence, all elements $X \oplus C_b$ and $X' \oplus C_{b'}$ (with $X \neq X'$) are distinct for all $b, b' \in \llbracket 0, 3 \rrbracket$. Hence, when $K = K'$, the four relations in Eq. (1) are independent.

- If $K \neq K'$, then the block cipher is instantiated with independently chosen permutations for the keys K and K' . In this case as well, the four relations in Eq. (1) are independent.

Since query $\#i$ is normal, the eight distinct encryption/decryption query responses used to obtain $\mathcal{L}_i$ are chosen from a set of size at least $N/2$. As $\#\mathcal{L}_i = 8$, it follows from the independence of the four relations that the probability that $\mathcal{A}$ finds a collision within a *specific* compression bundle in $\mathcal{L}_j$ (i.e., a set of eight compression pairs) is upper-bounded by

$$8 \cdot \frac{8}{(N/2)^4} = \frac{8^2 \cdot 2^4}{N^4}.$$

A regular query produces one compression bundle, whereas a super-query produces $(N/8)/2$ compression bundles. At query $\#i$, the maximum number of compression bundles present in *all* earlier $\mathcal{L}_j$ is therefore always at most $2(i-1)$ (observe that a super-query doubles the number of compression bundles for a given key). We therefore obtain

$$\Pr[C_i] \leq 2(i-1) \cdot \frac{8^2 \cdot 2^4}{N^4}.$$

Hence, summing over all i finally yields

$$\Pr[\text{NormalQueryWin}(\mathcal{L})] \leq \sum_{i=2}^q \Pr[C_i] \leq 8^2 \cdot 2^4 \frac{q^2}{N^4}.$$

SuperQueryWin In this case, $\mathcal{A}$ wins via a super-query $\#i$ for some $i \in \llbracket 1, q \rrbracket$.

For any query $\#i$ with $i \in \llbracket 1, q \rrbracket$, the maximum number of compression bundles present in $\mathcal{L}$ is upper-bounded by $2q$. Fix super-query $\#i$, and let D_i denote the event that at least one of the compression bundles it produces collides with a distinct compression bundle among fewer than $2q$ existing ones. Each such collision occurs with probability at most

$$8 \cdot \frac{8}{(N/2)^4} = \frac{8^2 \cdot 2^4}{N^4}.$$

Moreover, since a super-query yields $(N/2)/8$ compression bundles, we so have

$$\Pr[D_i] \leq \frac{N/2}{8} 2q \frac{8^2 \cdot 2^4}{N^4} = 8 \cdot 2^4 \frac{q}{N^3}.$$

The fact that a super-query implies that there have previously been $(N/8)/2$ normal queries under the same key, each giving rise to a bundle of 8 compression pairs, further implies that the total number of super-queries is at most

$$\left\lfloor \frac{q}{(N/8)/2} \right\rfloor \leq 8 \cdot 2 \frac{q}{N}.$$

Combining the above bounds, we finally obtain

$$\Pr[\text{SuperQueryWin}(\mathcal{L})] \leq 8^2 \cdot 2^5 \frac{q^2}{N^4}.$$

4.3.2 Internal collisions. Internal collisions are those that occur among the eight compression pairs within the same compression bundle. This event is referred to as SameQueryWin in [1].

SameQueryWin An internal collision occurs when there exist two distinct pairs $((K, X), F(K, X)), ((K', X'), F(K', X')) \in L_i$ within the same compression bundle such that $F(K, X) = F(K', X')$. Since all compression pairs within a compression bundle use the same key, we necessarily have $K = K'$ (and hence $X \neq X'$). We must also have $X' = X \oplus C$ for some $C \in \mathfrak{S} \setminus \{0\}$. Therefore, an internal collision occurs if and only if

$$\begin{cases} E_K(X) \oplus X = E_K(X') \oplus X' \\ E_K(X \oplus C_1) \oplus X = E_K(X' \oplus C_1) \oplus X' \\ E_K(X \oplus C_2) \oplus X = E_K(X' \oplus C_2) \oplus X' \\ E_K(X \oplus C_3) \oplus X = E_K(X' \oplus C_3) \oplus X' \end{cases}, \quad (2)$$

where $X' = X \oplus C$ for some $C \in \mathfrak{S} \setminus \{0\}$. If $C \neq C_7$, write $C = C_b \oplus C_{b'}$ with distinct $b, b' \in \llbracket 0, 3 \rrbracket$ (recall that $C_0 = 0$). We thus have $X' = X \oplus C_b \oplus C_{b'}$. Hence, two relations in Eq. (2) collapse into a single relation, since $X \oplus C_b = X' \oplus C_{b'}$ and $X' \oplus C_b = X \oplus C_{b'}$:

$$\begin{aligned} E_K(X \oplus C_b) \oplus X \oplus C_b &= E_K(X' \oplus C_b) \oplus X' \oplus C_b \\ E_K(X \oplus C_{b'}) \oplus X \oplus C_{b'} &= E_K(X' \oplus C_{b'}) \oplus X' \oplus C_{b'} \end{aligned}$$

Consequently, only three distinct and independent relations remain in Eq. (2) for an internal collision when $C \neq C_7$. Since a compression bundle consists of eight compression pairs, an internal collision then occurs with a probability upper-bounded by

$$\frac{\sum_{k=2}^8 (k-1)}{(N/2)^3} = \frac{28 \cdot 2^3}{N^3}.$$

Finally, as the total number of compression bundles cannot exceed $2q$, we obtain

$$\Pr[\text{SameQueryWin}(\mathcal{L})] \leq 28 \cdot 2^4 \frac{q}{N^3}. \quad (3)$$

4.3.3 Putting it all together. Combining the bounds derived for the three winning events, we obtain

$$\begin{aligned} &\Pr[\text{NormalQueryWin}(\mathcal{L})] + \Pr[\text{SuperQueryWin}(\mathcal{L})] + \\ &\Pr[\text{SameQueryWin}(\mathcal{L})] \leq 3 \cdot 2^{10} \frac{q^2}{N^4} + 7 \cdot 2^6 \frac{q}{N^3}. \end{aligned}$$

In turn, for the derived iterated hash function H , we immediately obtain the collision-resistance bound

$$\text{Adv}_H^{\text{coll}}(q) \leq 3 \cdot 2^{10} \frac{q^2}{N^4} + 7 \cdot 2^6 \frac{q}{N^3},$$

where $N = 2^n$ and n denotes the block size.

Remark 4.1. It is important to note that, for practically relevant values of the query parameter q , say 2^{60} , the contribution due to internal collisions captured by the SameQueryWin event dominates the bound. Indeed, the linear term in q remains larger than the quadratic term as long as $q \leq 2^n$.

4.4 Comparison with Counter-4DM

Counter-4DM uses a similar construction but with constants $C_{j-1} = j-1$ for $1 \leq j \leq 4$, so that $C_1 \oplus C_2 \oplus C_3 = 0$. While this choice does not affect the success probabilities of NormalQueryWin or SuperQueryWin in any significant way, it has a noticeable impact on the SameQueryWin event.

With Counter-4DM, four encryption or decryption queries using the same key allow one to form a bundle of four valid compression pairs. An internal collision occurs whenever there exist two integers $\ell, m \in \llbracket 0, 3 \rrbracket$, with $\ell \neq m$, such that²

$$\begin{cases} E_K(X \oplus m \oplus 0) \oplus X \oplus m = E_K(X \oplus \ell \oplus 0) \oplus X \oplus \ell \\ E_K(X \oplus m \oplus 1) \oplus X \oplus m = E_K(X \oplus \ell \oplus 1) \oplus X \oplus \ell \\ E_K(X \oplus m \oplus 2) \oplus X \oplus m = E_K(X \oplus \ell \oplus 2) \oplus X \oplus \ell \\ E_K(X \oplus m \oplus 3) \oplus X \oplus m = E_K(X \oplus \ell \oplus 3) \oplus X \oplus \ell \end{cases}.$$

Let $\kappa = m \oplus \ell \in \llbracket 1, 3 \rrbracket$. In particular, $\kappa \neq 0$ by hypothesis. We show that a collision with a counter value $0 \leq c \leq 3$ implies a collision with a counter value $c \oplus \kappa \neq c$, i.e., the relations in the above system collapse by pairs. Indeed, assume

$$E_K(X \oplus m \oplus c) \oplus X \oplus m = E_K(X \oplus \ell \oplus c) \oplus X \oplus \ell.$$

Then, starting from $E_K(X \oplus m \oplus (c \oplus \kappa)) \oplus X \oplus m$, which by rewriting terms is equal to $E_K(X \oplus \ell \oplus c) \oplus X \oplus m$, yields, using the collision hypothesis for c ,

$$\begin{aligned} E_K(X \oplus m \oplus (c \oplus \kappa)) \oplus X \oplus m &= E_K(X \oplus m \oplus c) \oplus X \oplus m \oplus \kappa \\ &= E_K(X \oplus \ell \oplus (c \oplus \kappa)) \oplus X \oplus \ell. \end{aligned}$$

Consequently, only two distinct and independent relations remain in the case of Counter-4DM. By an argument analogous to that used in the proof of Section 4.3, we therefore obtain

$$\Pr[\text{SameQueryWin}(\mathcal{L})] \leq 6 \cdot 2^4 \frac{q}{N^2}$$

for Counter-4DM, which has to be compared to Eq. (3). A similar analysis applies to other values of b in the general case of Counter- b DM, as well as to alternative choices of constants whose XOR is equal to 0.

5 Further Constructions

Designing QBL compression functions that *natively* support $(n, 2n)$ block ciphers remains an open and challenging problem; see, e.g., [1, Section 7].

This section presents candidate designs that enable an $(n, 2n)$ block cipher $E_K(\cdot)$, such as PRINCEv2, to be used within compression functions. While we believe the proposed constructions to be secure, we do not provide formal security proofs. We hope that these designs will motivate further research on QBL compression functions based on $(n, 2n)$ block ciphers.

QBL-MDC. This construction draws on the classical MDC-2 compression function [14, 36] and combines MDC-style output mixing with Hirose's double-block-length (DBL) transform [26]. Following Hirose's approach, it evaluates pairs of related block-cipher calls under the same key but with distinct input differences. An explicit output-mixing step, in the spirit of MDC constructions, is then applied to increase diffusion across four parallel sub-states.

A natural and lightweight choice for this output-mixing permutation is the AES ShiftRows (nibble-level) permutation [21], which coincides with the permutation layer P-Layer used in PRINCE. This operation acts on 16 words and provides an efficiently implementable diffusion layer.

²One $\oplus m$ term and one $\oplus \ell$ term have been omitted in the equations displayed in [1, Section 5] (cf. Subcase 1.3), which leads to the (incorrect) conclusion that $\Pr[\text{SameQueryWin}(\mathcal{L})] = 0$.

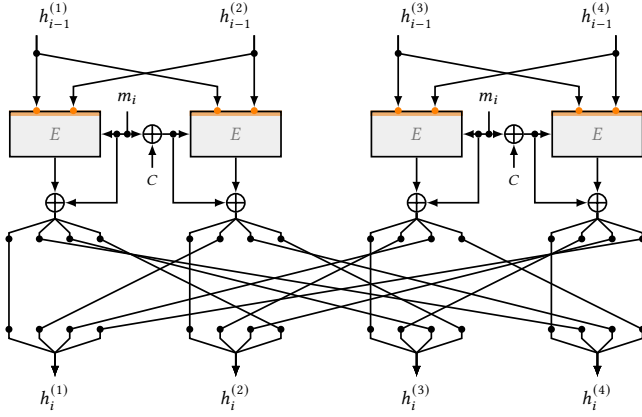


Figure 2: QBL-MDC construction.

More precisely, as depicted in Fig. 2, for each message block m_i , the construction computes four block-cipher evaluations as follows:

$$\begin{cases} T_i^{(1)} \leftarrow E_{h_{i-1}^{(1)} \| h_{i-1}^{(2)}}(m_i) \oplus m_i & = A_i^{(1)} \| B_i^{(1)} \| C_i^{(1)} \| D_i^{(1)} \\ T_i^{(2)} \leftarrow E_{h_{i-1}^{(1)} \| h_{i-1}^{(2)}}(m_i \oplus C) \oplus m_i \oplus C & = A_i^{(2)} \| B_i^{(2)} \| C_i^{(2)} \| D_i^{(2)} \\ T_i^{(3)} \leftarrow E_{h_{i-1}^{(3)} \| h_{i-1}^{(4)}}(m_i) \oplus m_i & = A_i^{(3)} \| B_i^{(3)} \| C_i^{(3)} \| D_i^{(3)} \\ T_i^{(4)} \leftarrow E_{h_{i-1}^{(3)} \| h_{i-1}^{(4)}}(m_i \oplus C) \oplus m_i \oplus C & = A_i^{(4)} \| B_i^{(4)} \| C_i^{(4)} \| D_i^{(4)} \end{cases}$$

for a fixed nonzero constant $C \in \{0, 1\}^n$. Then, each intermediate value $T_i^{(j)}$ is parsed into four $n/4$ -bit words $A_i^{(j)} \| B_i^{(j)} \| C_i^{(j)} \| D_i^{(j)}$ that are subsequently mixed across the four branches according to this permutation, and the chaining value is updated as:

$$\begin{aligned} & (h_i^{(1)}, h_i^{(2)}, h_i^{(3)}, h_i^{(4)}) \\ & \leftarrow (A_i^{(1)} \| B_i^{(2)} \| C_i^{(3)} \| D_i^{(4)}, A_i^{(2)} \| B_i^{(3)} \| C_i^{(4)} \| D_i^{(1)}, \\ & \quad A_i^{(3)} \| B_i^{(4)} \| C_i^{(1)} \| D_i^{(2)}, A_i^{(4)} \| B_i^{(1)} \| C_i^{(2)} \| D_i^{(3)}) . \end{aligned}$$

This crosswise recombination generalizes the output-mixing step of the classical MDC-2 construction and can be viewed as a fixed mixing on $4n$ bits that ensures sufficient inter-branch diffusion.

QBL-TDM. This construction is inspired by the Tandem-DM compression function [31]. It first derives two ephemeral subkeys from the message block and the previous chaining value by means of block-cipher evaluations:

$$k_i^{(1)} \leftarrow E_{h_{i-1}^{(4)} \| m_i}(h_{i-1}^{(3)}) \quad \text{and} \quad k_i^{(2)} \leftarrow E_{m_i \| h_{i-1}^{(1)}}(h_{i-1}^{(2)}) .$$

These subkeys are then used to update the chaining value in a Davies–Meyer-like fashion:

$$\begin{cases} h_i^{(1)} \leftarrow E_{k_i^{(1)}}(h_{i-1}^{(1)}) \oplus h_{i-1}^{(1)} \\ h_i^{(2)} \leftarrow k_i^{(2)} \oplus h_{i-1}^{(2)} \\ h_i^{(3)} \leftarrow k_i^{(1)} \oplus h_{i-1}^{(3)} \\ h_i^{(4)} \leftarrow E_{m_i \| k_i^{(2)}}(h_{i-1}^{(4)}) \oplus h_{i-1}^{(4)} . \end{cases}$$

This is pictured in Fig. 3.

This update ensures that all components of the previous chaining value $(h_{i-1}^{(1)}, \dots, h_{i-1}^{(4)})$ influence the next state. Indeed, $h_{i-1}^{(2)}$ and $h_{i-1}^{(3)}$

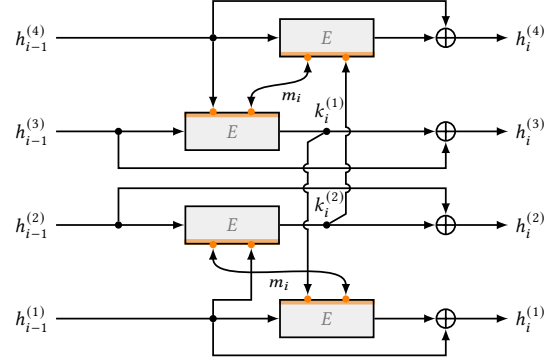


Figure 3: QBL-TDM construction.

are first processed through block-cipher evaluations to derive the subkeys $k_i^{(1)}$ and $k_i^{(2)}$, while $h_{i-1}^{(1)}$ and $h_{i-1}^{(4)}$ are updated via Davies–Meyer-like constructions keyed by these values. Observe that the updates of $h_i^{(2)}$ and $h_i^{(3)}$ mirror, in structure, those of $h_i^{(1)}$ and $h_i^{(4)}$, in the sense that all four state words are combined with subkeys derived from non-linear block-cipher calls.

As in Tandem-DM, this design is intended to promote diffusion and to preclude simple internal collisions. While this construction follows well-understood design principles, we do not claim a formal security proof in the QBL setting, i.e., for compression functions built from $(n, 2n)$ block ciphers.

6 FHE Implementation with PRINCEv2

We outline practical considerations for implementing PRINCEv2 [13] in FHE as part of the proposed constructions. Although we primarily focus on computational depth and latency, our implementation also happens to feature the smallest number of operations in an equivalent setting to [38], i.e., using the Multi-Value Bootstrapping (MVB) technique from [15].

6.1 FHE Settings

We use the TFHE scheme [20], leveraging its fast programmable bootstrapping (PBS) capability [19], which makes TFHE particularly well suited for low-latency applications and implementation of general-purpose algorithms.

Parameters and failure probability. As already noted in [39, Table 3], [37, Table 1], and [4, Table 2], using a 4-bit plaintext space in TFHE seems optimal in terms of performance and functional efficiency, as parameters tend to blow up as the plaintext space increases. This is also one of the reasons PRINCEv2 is particularly well suited for TFHE as its S-box layers directly operate on 4-bit nibbles.

Within this setting, we target parameters achieving a failure probability p_{fail} of (at most) 2^{-128} . As will be seen later, our implementation of PRINCEv2 also requires a noise budget to allow for five ciphertext additions between two PBSes. Hence, we use the `PARAM_MESSAGE_2_CARRY_2_KS_PBS_GAUSSIAN_2M128 (v1.4)` parameter set from the TFHE-rs library [40], which corresponds exactly to this setting.

Achieving such a small failure probability is in fact crucial for IND-CPA^D security, even for exact FHE schemes [9, 17, 18, 32]. This contrasts with e.g., parameters used in the implementation of [38], which, according to [38, Table 2] only achieve $p_{\text{fail}} \approx 2^{-23}$, or e.g., the advertised timings for the AES implementation in [7] that were obtained for $p_{\text{fail}} \approx 2^{-40}$. We stress however that [7] also provides additional timings for their implementation with $p_{\text{fail}} \approx 2^{-128}$ [7, Table 4], which are more than 10 times slower and clearly show the significance of this parameter when comparing homomorphic computations.

A new hybrid LUT approach: On the importance of FHE plumbing. Two homomorphic circuit strategies were considered in [38]. The first is the so-called *Full LUT* approach, which combines the Tree-Based Method from [25] and the Multi-Value Bootstrapping from [15]. The second is the *Hippogryph* framework [7], which leverages the concept of (o, p) -encodings with the ability to switch between encodings via a decomposer/recomposer homomorphic operation. Concretely, the Full LUT approach always considers LUTs from 4 bits to 4 bits, while the Hippogryph framework uses decomposition/recomposition on top of that to obtain/recombine individual bits for binary operations.

However, we adopt a more efficient and versatile approach. Namely, in our Hybrid LUT approach, every layer is computed by means of LUTs whose inputs remain encrypted 4-bit nibbles, as in the Full LUT approach, except that we may directly decompose the output encrypted value in (shifted) bits, or (shifted) pairs of bits, i.e., whatever format is more convenient for the next homomorphic layer in the circuit. This is achieved very naturally by duplicating the original LUT as needed. For example, to output individual bits on the least significant position of the plaintext space, a LUT $L[x]$ for $x \in \llbracket 0, 15 \rrbracket$ is replaced by four parallel LUTs L_i that return $L_i[x] := (L[x] \gg i) \& 1$. Note that this is intrinsically an ideal setting for the MVB technique of [15]. Then, the output ciphertexts may be recombined through (almost free) ciphertext additions to form adequate input 4-bit nibbles for the next homomorphic layer.

Although this seemingly increases the number of PBSEs,³ we will show that this design overall saves many PBSEs, especially in the linear and XOR layers, while also featuring a better computational depth.

Admittedly, the design of the homomorphic circuit in our Hybrid LUT approach becomes noticeably more involved, since the output of each layer now depends on the design of the subsequent homomorphic layer. Nevertheless, in the case of PRINCEv2, this results in a much better operation count and computational depth compared to previous frameworks, showing that proper FHE plumbing is indeed crucial.

Key scheduling. In previous works [7, 38] in the context of transciphering, a common assumption is that FHE encryptions of all round keys are provided directly, instead of just the FHE encryption of the (master) secret key. Consequently, key expansion is never done in the homomorphic domain in those works.

This is in stark contrast with our context of iterated hash functions, where the secret keys are not known in advance and depend on the encrypted message blocks. Fortunately, in the case of PRINCEv2, we are able to perform the key expansion directly within the S-box layers, as explained in the next section, making the key schedule virtually free.

6.2 PRINCEv2 Layers in FHE

A detailed specification of the PRINCEv2 blockcipher is provided in [13] and sketched in Appendix A. Recall that the block size is 64 bits and the 128-bit key K is split into two 64-bit chunks $K = k_0 \parallel k_1$.

We describe here the FHE implementation of its linear, XOR, and S-box layers. Since the permutation layers (and inverses) act on 4-bit nibbles, those can always be obtained by simply reindexing the ciphertexts.

For convenience, all encrypted global inputs/outputs of the homomorphic block cipher (namely, m , k_0 , k_1 , and $E_K(m)$) are given as lists of 32 ciphertexts, where each ciphertext encrypts 2 bits in the *least significant bits* of their 4-bit plaintext space. This representation facilitates chaining between successive calls in the iterated hash function. *A contrario*, the encrypted state between two layers may be seen as:

- (a) 16 ciphertexts, each encrypting one 4-bit nibble, or
- (b) 32 ciphertexts, each encrypting 2 bits in the *most significant bits* of their 4-bit plaintext space.

We will also encounter intermediate values represented as 64 ciphertexts, each encrypting a single bit that we may place in any position within the 4-bit plaintext space.

6.2.1 On the linear layers. We begin, somewhat counter-intuitively, by describing our FHE implementation of the linear layer. Indeed, this is the most challenging component to implement in FHE (as noted e.g., in [38, Subsection 4.4.2]), and our chosen input format for this layer constrains the output of other layers.

The linear layer is defined by 16×16 matrices $\hat{M}^{0/1}$. An important observation is that their columns only involve individual bits whose indices are identical modulo 4. For instance, the columns $\hat{M}^0$ all have 3 ones and 13 zeros: the first column is non-zero at positions 48c, the second at positions 19d, then 26e, 37a, 048, 59d, (...). Therefore, we assume the encrypted input of the linear layer is given as encrypted 4-bit nibbles containing bits 048c, 159d, 26ae, 37bf (...). As we will see, this is obtained by “modifying” the previous layer, whether it is an S-box layer or a XOR layer, so as to output encryptions of individual shifted bits, followed by 16 summations of four ciphertexts (e.g., the first input encrypts the sum of $(b_0 \ll 3)$, $(b_4 \ll 2)$, $(b_8 \ll 1)$ and b_c).

Consequently, the linear layer now essentially boils down to computing, on each such 4-bit nibble, the XOR of all bits except one for each possible exception, i.e., 64 parallel LUT evaluations. This contrasts with the approach of [38], which computes the multiplications $\tilde{c}_{i,j}$'s of 4-bit vector with matrix blocks, followed by 3 homomorphic XOR on 4-bit values. Even using the MVB+TBM approach, this is a depth-5 computation involving an equivalent of 112 LUT evaluations.

Output formats. In forward and middle rounds, the linear layer is followed by a XOR layer. In this case, encrypted output bits of even

³For instance, the forward S-box layer will be transformed from 16 parallel PBSEs in the Full LUT approach to 64 parallel PBSEs (or 16 times a 4-value PBS) in our Hybrid LUT approach, outputting carefully shifted individual bits.

(resp. odd) indices are natively left-shifted by 3 (resp. 2) positions, so that summing pairs of consecutive ciphertexts give 32 ciphertexts containing $(b_0b_1 \ll 2), (b_2b_3 \ll 2), \dots$

In backward rounds, the next layer is an S-box layer. In this case, encrypted output bits of index $b \in \llbracket 0, 63 \rrbracket$ are pre-shifted by $(3 - b) \bmod 4$, so that summing 16 quadruples of consecutive ciphertexts gives the next input nibbles encrypting bits 0123, 4567, (...).

Remark 6.1. All these pre-shiftings are directly included when building LUTs, so this has no impact on the noise, apart from the subsequent summation of 2/4 freshly bootstrapped ciphertexts, which stays within our noise budget.

6.2.2 On the S-box layers: Merging with round constants. As mentioned earlier, we cannot assume, as in previous works, that round keys are given as pre-encrypted inputs. Luckily, in PRINCEv2 the key schedule is rather simple, being a XOR of k_0 (alternatively k_1) with predefined round constants RC_t , for $1 \leq t \leq 11$. However, performing this naively would still incur, for every round, a layer of homomorphic operations in our critical path.

We observe that the round constants can be moved through both linear and permutation layers right into the S-box layers, specializing the S-boxes according to their nibble index and specific round number. Formally, the S-box applied on the w -th encrypted 4-bit nibble x ($w \in \llbracket 0, 15 \rrbracket$) in the t -th round is redefined as the following LUT

$$S_w^{(t)}[x] := \begin{cases} S[x] \oplus (\pi^{-1}(RC_t) \cdot M^{-1})[w] & \text{if } t \in \llbracket 1, 5 \rrbracket, \\ S^{-1}[x] \oplus RC_t[w] & \text{if } t \in \llbracket 6, 11 \rrbracket, \end{cases} \quad (4)$$

where $M = M^{-1}$ is the linear layer matrix and π corresponds to the permutation layer. Similarly, $S_w^{(m)}[x] := S[x] \oplus (RC_{11} \cdot M^{-1})[w]$ redefines the first S-box layer in the middle round, so that the complete sequence of applied S-boxes is

$$S^{(1)}, \dots, S^{(5)}, S^{(m)}, S^{(6)}, \dots, S^{(11)}.$$

This results in 16×12 different LUTs, one per S-box layer and per 4-bit nibble. It is possible to reduce this number to 16×8 different LUTs by combining the round constants differently, namely by integrating the previous one in the input and the next one in the output, i.e.,

$$\hat{S}_w^{(tu)}[x] := S[x \oplus RC_t[w]] \oplus (\pi^{-1}(RC_u) \cdot M^{-1})[w] \quad (5)$$

where $t \in \{1, 3\}$, $u = t + 1$. The middle round S-box layer becomes

$$\hat{S}_w^{(5m)}[x] := S[x \oplus RC_5[w]] \oplus (RC_{11} \cdot M^{-1})[w] \quad (6)$$

and the backward S-box layers are given, for $t \in \{6, 8, 10\}$ and $u = t + 1$, by

$$\hat{S}_w^{(tu)}[x] := S^{-1}[x \oplus (\pi^{-1}(RC_t) \cdot M^{-1})[w]] \oplus RC_u[w]. \quad (7)$$

In this setting, the complete sequence of applied S-boxes thus becomes

$$S, \hat{S}^{(12)}, S, \hat{S}^{(34)}, S, \hat{S}^{(5m)}, S^{-1}, \hat{S}^{(67)}, S^{-1}, \hat{S}^{(89)}, S^{-1}, \hat{S}^{(10-11)}.$$

Output formats. Each (inverse) S-box layer is either followed by a linear layer or by a XOR layer. When followed by a linear layer, as discussed in the previous paragraph, we need to output (and sum) individual bits whose alignment depend on their nibble index modulo 4. Concretely, e.g., for applying an S-box layer $S^{(t)}$ as defined above on the encrypted state $(n_0, \dots, n_{15})$, we first output, for each $w \in \llbracket 0, 15 \rrbracket$ and $b \in \llbracket 0, 3 \rrbracket$,

$$s[4w + b] := \left((S_w^{(t)}[n_w] \gg (3 - b)) \& 1 \right) \ll (3 - w \bmod 4),$$

which is a homomorphic evaluation of 64 parallel LUTs, or using the MVB technique, 16 parallel 4-value bootstrappings. Then, the linear layer receives an encrypted state $(c_0, \dots, c_{15})$, where for $w \in \llbracket 0, 15 \rrbracket$, c_w encrypts

$$\sum_{0 \leq b \leq 3} s[16\lfloor w/4 \rfloor + 4b + (w \bmod 4)], \quad (8)$$

which is evaluated homomorphically by simply summing ciphertexts.

The situation is much simpler when the next layer is a XOR layer. In that case, we only need to extract separately pairs of least/most significant output bits and place them always in the most significant bits of the 4-bit plaintext space. This means 32 parallel LUT evaluations, or 16 parallel 2-value bootstrappings.

6.2.3 On the XOR layers. We finally tackle the XOR layers, which XOR the encrypted state with the encrypted keys. Recall these are given as lists of 32 ciphertexts encrypting 2 bits at a time, that are placed in the *least significant* bits of the plaintext space. The reader may also have noticed that previous layers, when followed by a XOR layer, output 32 ciphertexts encrypting 2 bits at a time, but placed in the *most significant* bits of the plaintext space.

Then, we use a very natural trick, called a *Bivariate PBS* in e.g., [4, Remark 28], to obtain the result. Both operands are homomorphically summed, and a 4-bit LUT applied to this sum simply outputs $L[x] := (x \& 3) \oplus (x \gg 2)$.

Output formats. If followed by a linear layer, we may extract bits and sum them following the same pattern as in Eq. (8). Likewise, if the next layer is an S-box layer, the resulting bits may be left-shifted by 2 positions on even indices so that summing pairs of consecutive ciphertexts produces encrypted 4-bit nibbles that are ready to enter the S-boxes.

Remark 6.2. The first XOR layer receives an encrypted message that is not left-aligned in the plaintext space. However, if this message is a freshly bootstrapped ciphertext, the noise budget is sufficient to homomorphically multiply it by 4.

6.2.4 On middle round options. The authors of [13] propose a trick to remove a XOR layer in the critical path of the middle round by moving the XOR with k_0 after the linear layer, so that $k_1 \oplus (k_0 \cdot M)$ is XORed to the state in a single pass.

This trick also works in the homomorphic domain. During forward rounds, spare computation units could be used for homomorphically: *a.* extracting and shifting bits of k_0 and recombining them for *b.* computing $(k_0 \cdot M)$ and finally *c.* computing $k_1 \oplus (k_0 \cdot M)$. This would reduce the computational depth of our implementation from 37 layers of parallel PBSes to 36 layers.

Table 1: Computational depth depending on the number of parallel computing units for different FHE implementations of PRINCEv2 in the MVB (+TBM for [38]) setting.

Parallelism	1	2	4	8	16	32	64
Full LUT [38]	1840	920	460	230	115	93	(93)
Hippogryph [7, 38]	1152	576	288	144	72	48	36
Ours (using MVB)	816	408	204	102	51	37	(37)
Ours (no MVB, see Remark 6.3)	1888	944	472	236	118	59	37

However, we argue this unbalanced computing setting would be rather difficult to manage (and probably counter-productive) compared to the much simpler, very synchronized, setting where 64 cores apply parallel PBSes for 37 times.

6.3 Counting PBSes

Our homomorphic evaluation of PRINCEv2, including the key schedule, consists of 37 layers of 32/64 parallel 4-bit PBSes, using the techniques described above. More precisely, where $(k \times n)$ denotes the possibility of performing k parallel n -value bootstrappings:

- (1) There are 5 S-box layers followed by a linear layer, i.e., 64 (16×4) parallel PBSes each, and 7 (inverse) S-box layers followed by a XOR layer, i.e., 32 (16×2) parallel PBSes each;
- (2) Linear layers are always 64 (16×4) parallel PBSes, and there are 11 of these;
- (3) Among the XOR layers, the seven followed by an S-box layer, as well as the last one, consist of 32 parallel PBSes each, and the other six are followed by a linear layer, meaning they require 64 (32×2) parallel PBSes.

This yields a grand total of 1,888 *sequential* PBSes, *without* considering Multi-Value Bootstrapping. We stress that this number is already competitive with the 1,808 figure obtained by [38, Table 1] with no key scheduling in their MVB+TBM setting on PRINCE (v1), i.e., with two fewer XOR layers.

When considering Multi-Value Bootstrapping, each $(k \times n)$ figure above is basically replaced by k PBSes and $O(kn)$ negligible polynomial multiplications.⁴ Hence, the total PBS count is reduced to only 816. Despite the fact that our key scheduling is performed in the homomorphic domain, this represents less than half of the 1,808 PBSes reported in [38] and even outperforms their Hippogryph figure (1,152 PBSes, neglecting the complexity of the linear layer) by 29%.

Finally, regarding computational depth, we achieve depth 37 using 64 parallel units without MVB, slightly more than Hippogryph (depth 36). Using MVB, this depth is already achieved in our case for 32 cores (resp. depth 51 for 16 cores), whereas Hippogryph stands at depth 48 in this case (resp. depth 72 on 16 cores) due to its heavy recomposer. By contrast, the FullLUT strategy of [38] would require at least 93 layers on 32/64 cores (resp. 115 on 16 cores). Computational depths are summarized in Table 1.

Remark 6.3. A very important caveat of the Multi-Value Bootstrapping approach is its significant impact on noise growth and,

thus, on parameter selection. This issue is somehow addressed in [38] by substantially increasing the failure probability [38, Appendix C]. If we instead target a constant failure probability of 2^{-128} , this would introduce an algorithmic overhead of about 53% per PBS, increasing as much the global latency on a fully parallelized architecture. However, it would still be interesting on smaller architectures as long as the computational depth is divided by at least 1.5, i.e., with fewer than 16 cores.

6.4 Experimental Results and Timings

We report performance results obtained using the TFHE-rs library (v1.4.2) [40] and the Rayon crate (v1.11.0) for parallelization. The resulting code will be included in a subsequent release of TFHE-rs.

We ran our homomorphic implementation of PRINCEv2 on an Amazon AWS hpc7a.96xlarge with 2×96 AMD EPYC 9R14 physical cores. Recall that our implementation is explicitly designed to achieve a failure probability of 2^{-128} using the parameter set PARAM_MESSAGE_2_CARRY_2_KS_PBS_GAUSSIAN_2M128 (v1.4), in contrast to [38] where the failure probability exceeds 2^{-40} . Also, aiming at the lowest possible latencies on a massively parallel architecture, we did not include the MVB approach in our implementation (see Remark 6.3).

Table 2: Homomorphic evaluation timings of PRINCEv2 as a function of the degree of parallelism, targeting a failure probability of 2^{-128} (without MVB).

# cores	1	2	4	8	16	32	64
Timing (s)	33.123	19.527	10.236	5.132	2.701	1.492	0.776

The results are summarized in Table 2. As expected, they exhibit excellent scalability up to 64 cores, achieving an impressive 776 ms latency per call, while also remaining competitive in single-core performance. For comparison, the corresponding latencies reported in [38, Table 1] are 105 s for FullLUT and 35 s for Hippogryph, both obtained on a single core and for a *much higher* failure probability. As shown in [7, Table 4], reducing the failure probability from 2^{-40} to 2^{-128} incurs a performance penalty exceeding 10×, despite using the same implementation and techniques.

References

- [1] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. 2014. Counter-BDM: A provably secure family of multi-block-length compression functions. In *Progress in Cryptology – AFRICACRYPT 2014 (Lecture Notes in Computer Science, Vol. 8469)*, D. Pointcheval and D. Vergnaud (Eds.). Springer, 440–458. doi:10.1007/978-3-319-06734-6_26

⁴Each of these multiplications is about $1,000 \times$ cheaper than a PBS.

- [2] Frederik Armknecht, Ewan Fleischmann, Matthias Krause, Jooyoung Lee, Martijn Stam, and John Steinberger. 2011. The preimage security of double-block-length compression functions. In *Advances in Cryptology – ASIACRYPT 2011 (Lecture Notes in Computer Science, Vol. 7073)*, D. H. Lee and X. Wang (Eds.). Springer, 233–251. doi:10.1007/978-3-642-25385-0_13
- [3] Roberto Avanzi. 2017. The QARMA block cipher family – Almost MDS matrices over rings with zero divisors, nearly symmetric Even-Mansour constructions with non-involutory central rounds, and search heuristics for low-latency S-boxes. *IACR Transactions on Symmetric Cryptology* 2017, 1 (2017), 4–44. doi:10.13154/tosc.v2017.i1.4-44
- [4] Mathieu Ballardras, Loris Bergerat, Charlotte Bonte, Carl Bootland, Benjamin R. Curtis, Jad Khatib Jakub Klemsa, Thomas Montaigu, Jean-Baptiste Orfila, and Samuel Tap. 2025. *TFHE-rs: A (Practical) Handbook* (1st ed.). Zama. <https://github.com/zama-ai/tfhe-rs>
- [5] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. 2015. The SIMON and SPECK lightweight block ciphers. In *52nd Annual Design Automation Conference (DAC '15)*. ACM Press, 175.1–175.6. doi:10.1145/2744769.274794
- [6] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. 2016. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology – CRYPTO 2016, Part II (Lecture Notes in Computer Science, Vol. 9815)*, M. Robshaw and J. Katz (Eds.). Springer, 123–153. doi:10.1007/978-3-662-53008-5_5
- [7] Sonia Belaid, Nicolas Bon, Aymen Boudguiga, Renaud Sirdey, Daphné Trama, and Nicolas Ye. 2025. Further improvements in AES execution over TFHE. *IACR Communications in Cryptology* 2, 1 (2025), 24 pages. doi:10.62056/ahmp-4tw9
- [8] Adda-Akram Bendoukha, Oana Stan, Renaud Sirdey, Nicolas Quero, and Luciano Freitas. 2023. Practical homomorphic evaluation of block-cipher-based hash functions with applications. In *Foundations and Practice of Security (FPS 2022) (Lecture Notes in Computer Science, Vol. 13877)*, G.-V. Jourdan et al. (Eds.). Springer, 88–103. doi:10.1007/978-3-031-30122-3_6
- [9] Olivier Bernard, Marc Joye, Nigel Smart, and Michael Walter. 2025. Drifting towards better error probabilities in fully homomorphic encryption schemes. In *Advances in Cryptology – EUROCRYPT 2025, Part VIII (Lecture Notes in Computer Science, Vol. 15608)*, S. Fehr and P.-A. Fouque (Eds.). Springer, Cham, 181–211. doi:10.1007/978-3-031-91101-9_7
- [10] John Black, Phillip Rogaway, and Thomas Shrimpton. 2002. Blackbox analysis of the block-cipher-based hash-function constructions from PGV. In *Advances in Cryptology – CRYPTO 2002 (Lecture Notes in Computer Science, Vol. 2442)*, M. Yung (Ed.). Springer, 320–335. doi:10.1007/3-540-45708-9_21
- [11] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. 2007. PRESENT: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems – CHES 2007 (Lecture Notes in Computer Science, Vol. 4727)*, P. Paillier and I. Verbauwhede (Eds.). Springer, 450–466. doi:10.1007/978-3-540-74735-2_31
- [12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventsislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. 2012. PRINCE – A low-latency block cipher for pervasive computing applications. In *Advances in Cryptology – ASIACRYPT 2012 (Lecture Notes in Computer Science, Vol. 7658)*, X. Wang and K. Sako (Eds.). Springer, 208–225. doi:10.1007/978-3-642-34961-4_14
- [13] Dusan Bozilov, Maria Eichlseder, Miroslav Knežević, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventsislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. 2020. PRINCEv2: More security for (almost) no overhead. In *Selected Areas in Cryptography (SAC 2020) (Lecture Notes in Computer Science, Vol. 12804)*, O. Dunkelman et al. (Eds.). Springer, 483–511. doi:10.1007/978-3-030-81652-0_19
- [14] Bruno O. Brachtel, Don Coppersmith, Myrna M. Hyden, Stephen M. Matyas, Jr., Carl H. W. Meyer, Jonathan Oseas, Shai Pilpel, and Michael Schilling. 1990. Data authentication using modification detection codes based on a public one way encryption function. U.S. Patent #4,908,861. <https://patentimages.storage.googleapis.com/32/ab/b1/6c32edbc15b5b3/US4908861.pdf>
- [15] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New techniques for multi-value input homomorphic evaluation and applications. In *Topics in Cryptology – CT-RSA 2019 (Lecture Notes in Computer Science, Vol. 11405)*, M. Matsui (Ed.). Springer, 106–126. doi:10.1007/978-3-030-12612-4_6
- [16] Donghoon Chang, Mridul Nandi, Jesang Lee, Jaechul Sung, Seokhie Hong, Jongin Lim, Haeryong Park, and Kilsoo Chun. 2008. Compression function design principles supporting variable output lengths from a single small function. *IE-ICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E91.A, 9 (2008), 2607–2614. doi:10.1093/ietf/ec91-a.9.2607
- [17] Marina Checri, Renaud Sirdey, Aymen Boudguiga, and Jean-Paul Bultel. 2024. On the practical CPA^D security of “exact” and threshold FHE schemes and libraries. In *Advances in Cryptology – CRYPTO 2024, Part III (Lecture Notes in Computer Science, Vol. 14922)*, L. Reyzin and D. Stebila (Eds.). Springer, Cham, 3–33. doi:10.1007/978-3-031-68382-4_1
- [18] Jung Hee Cheon, Hyeonmin Choe, Alain Passetlègue, Damien Stehlé, and Elias Suvanto. 2024. Attacks against the IND-CPA^D security of exact FHE schemes. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS'24)*. ACM, 2505–2519. doi:10.1145/3658644.3690341
- [19] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In *Cyber Security Cryptography and Machine Learning (CSCML 2021) (Lecture Notes in Computer Science, Vol. 12716)*, S. Dolev et al. (Eds.). Springer, 1–19. doi:10.1007/978-3-030-78086-9_1
- [20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology* 33, 1 (2020), 34–91. doi:10.1007/s00145-019-09319-x
- [21] Joan Daemen and Vincent Rijmen. 2002. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer. doi:10.1007/978-3-662-04722-4
- [22] Ivan Damgård. 1990. A design principle for hash functions. In *Advances in Cryptology – CRYPTO '89 (Lecture Notes in Computer Science, Vol. 435)*, G. Brassard (Ed.). Springer, 416–427. doi:10.1007/0-387-34805-0_39
- [23] Ewan Fleischmann, Christian Forler, Stefan Lucks, and Jakob Wenzel. 2012. Weimar-DM: A highly secure double-length compression function. In *Information Security and Privacy (ACISP 2012) (Lecture Notes in Computer Science, Vol. 7372)*, W. Susilo et al. (Eds.). Springer, 152–165. doi:10.1007/978-3-642-31448-3_12
- [24] Craig Gentry. 2010. Computing arbitrary functions of encrypted data. *Commun. ACM* 53, 3 (2010), 97–105. doi:10.1145/1666420.1666444
- [25] Antonio Guimarães, Edson Borin, and Diego F. Aranha. 2021. Revisiting the functional bootstrap in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 2 (2021), 229–253. doi:10.46586/tches.v2021.i2.229-253
- [26] Shioichi Hirose. 2006. Some plausible constructions of double-block-length hash functions. In *Fast Software Encryption (FSE 2006) (Lecture Notes in Computer Science, Vol. 4047)*, M. J. B. Robshaw (Ed.). Springer, 210–225. doi:10.1007/11799313_14
- [27] Shioichi Hirose. 2006. How to construct double-block-length hash functions. Second NIST Cryptographic Hash Workshop, Santa Barbara, CA, USA. https://csrc.nist.gov/groups/ST/hash/documents/HIROSE_article.pdf Revised version of [26].
- [28] ISO/IEC. 2010. Information technology – Security techniques – Hash-functions, Part 2: Hash-functions using an n -bit block cipher. ISO/IEC 10118-2 Standard. <https://www.iso.org/standard/44737.html>
- [29] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. 2014. Tweaks and keys for block ciphers: The TWEAKE framework. In *Advances in Cryptology – ASIACRYPT 2014 (Lecture Notes in Computer Science, Vol. 8874)*, P. Sarkar and T. Iwata (Eds.). Springer, 274–288. doi:10.1007/978-3-662-45608-8_15
- [30] Lars R. Knudsen, Florian Mendel, Christian Rechberger, and Søren S. Thomsen. 2009. Cryptanalysis of MDC-2. In *Advances in Cryptology – EUROCRYPT 2009 (Lecture Notes in Computer Science, Vol. 5479)*, A. Joux (Ed.). Springer, 106–120. doi:10.1007/978-3-642-01001-9_6
- [31] Xuejia Lai and James L. Massey. 1993. Hash function based on block ciphers. In *Advances in Cryptology – EUROCRYPT '92 (Lecture Notes in Computer Science, Vol. 658)*, R. A. Rueppel (Ed.). Springer, 55–70. doi:10.1007/3-540-47555-9_5
- [32] Baiyu Li and Daniele Micciancio. 2021. On the security of homomorphic encryption on approximate numbers. In *Advances in Cryptology – EUROCRYPT 2021, Part I (Lecture Notes in Computer Science, Vol. 12696)*, A. Canteaut and F.-X. Standaert (Eds.). Springer, 648–677. doi:10.1007/978-3-030-77870-5_23
- [33] Ralph C. Merkle. 1990. One way hash functions and DES. In *Advances in Cryptology – CRYPTO '89 (Lecture Notes in Computer Science, Vol. 435)*, G. Brassard (Ed.). Springer, 428–446. doi:10.1007/0-387-34805-0_40
- [34] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. 1978. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, R. A. DeMillo et al. (Eds.). Academic Press, 169–179. <https://people.csail.mit.edu/rivest/pubs/RAD78.pdf>
- [35] Yaobin Shen, Thomas Peters, and François-Xavier Standaert. 2024. Multiplex: TBC-based authenticated encryption with Sponge-like rate. *IACR Transactions on Symmetric Cryptology* 2024, 2 (2024), 1–34. doi:10.46586/tosc.v2024.i2.1-34
- [36] John P. Steinberger. 2007. The collision intractability of MDC-2 in the ideal-cipher model. In *Advances in Cryptology – EUROCRYPT 2007 (Lecture Notes in Computer Science, Vol. 4515)*, M. Naor (Ed.). Springer, 34–51. doi:10.1007/978-3-540-72540-4_3
- [37] Daphné Trama, Aymen Boudguiga, Pierre-Emmanuel Clet, Renaud Sirdey, and Nicolas Ye. 2025. Designing a general-purpose 8-bit (T)FHE processor abstraction. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025, 2 (2025), 535–578. doi:10.46586/tches.v2025.i2.535-578
- [38] Daphné Trama, Aymen Boudguiga, and Renaud Sirdey. 2026. Running standard block ciphers beyond AES with TFHE: Experiments and lessons learnt. *IACR Communications in Cryptology* 2, 4 (2026), 23 pages. doi:10.62056/avom-4tw9
- [39] Daphné Trama, Pierre-Emmanuel Clet, Aymen Boudguiga, and Renaud Sirdey. 2023. A homomorphic AES evaluation in less than 30 seconds by means of TFHE. In *11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC 2023)*, M. Brenner et al. (Eds.). ACM, 79–90. doi:10.1145/3605759.3625260

[40] Zama. 2022. TFHE-rs: A pure Rust implementation of the TFHE scheme for boolean and integer arithmetics over encrypted data. <https://github.com/zama-ai/tfhe-rs>

A PRINCEv2 Block Cipher

This appendix summarizes PRINCEv2 [13], the 64-bit, 128-bit-key block cipher underlying our constructions; its encryption workflow is shown in Fig. 4.

A.1 Key Schedule

The master key is split as $K = k_0 || k_1$, and round keys are

$$rk_r = k_{r \bmod 2} \oplus RC_r, \quad 0 \leq r \leq 11.$$

The round constants are listed in Table 3; for $6 \leq r \leq 11$, $RC_r = RC_{11-r} \oplus \alpha$ for even r and $RC_r = RC_{11-r} \oplus \beta$ for odd r , where $\alpha = c0ac29b7c97c50dd$ and $\beta = 3f84d5b5b5470917$.

Table 3: Round constants RC_r (PRINCEv2).

$RC_0 = 0000000000000000$	$RC_6 = 7ef84f78fd955cb1$
$RC_1 = 13198a2e03707344$	$RC_7 = 7aacf4538d971a60$
$RC_2 = a4093822299f31d0$	$RC_8 = c882d32f25323c54$
$RC_3 = 082efa98ec4e6c89$	$RC_9 = 9b8ded979cd838c7$
$RC_4 = 452821e638d01377$	$RC_{10} = d3b5a399ca0c2399$
$RC_5 = be5466cf34e90c6c$	$RC_{11} = 3f84d5b5b5470917$

A.2 Encryption and Decryption

Forward rounds, the middle round, and backward rounds are

$$R = \text{P-Layer} \circ \text{M-Layer} \circ \text{S-Layer},$$

$$R'(X) = \text{S-Layer}^{-1}(rk_{11} \oplus \text{M-Layer}(rk_0 \oplus \text{S-Layer}(X))),$$

$$R^{-1} = \text{S-Layer}^{-1} \circ \text{M-Layer}^{-1} \circ \text{P-Layer}^{-1}.$$

PRINCEv2 has no pre- or post-whitening. Its encryption circuit can be reused for decryption: with $\mathcal{E}$ denoting the top branch of Fig. 4 followed by S-Layer and XOR with rk_0 ,

$$E_{k_0 || k_1}^{-1} = \mathcal{E}_{k_0 || k_1}^{-1} \circ \text{M-Layer} \circ \mathcal{E}_{(k_1 \oplus \beta) || (k_0 \oplus \alpha)}.$$

A.3 Round Layers

Substitution. Viewing the state as 16 nibbles, S-Layer applies S-box independently to each nibble, and S-Layer^{-1} applies S-box^{-1} ; the tables are given in Table 4.

Table 4: PRINCEv2 substitution tables.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\text{S-box}(x)$	b	f	3	2	a	c	9	1	6	7	8	0	e	5	d	4

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\text{S-box}^{-1}(x)$	b	7	3	2	f	d	8	9	a	6	4	0	5	e	c	1

Linear layer. Let M_i be the 4×4 identity matrix with row i set to zero, and define

$$\hat{M}^{(0)} = \begin{bmatrix} M_0 & M_1 & M_2 & M_3 \\ M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \end{bmatrix}, \quad \hat{M}^{(1)} = \begin{bmatrix} M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \\ M_0 & M_1 & M_2 & M_3 \end{bmatrix}.$$

Then M-Layer maps C to CM , where $M = \text{diag}(\hat{M}^{(0)}, \hat{M}^{(1)}, \hat{M}^{(1)}, \hat{M}^{(0)})$. Since $M^2 = I$, $\text{M-Layer}^{-1} = \text{M-Layer}$.

Permutation layer. On nibble indices, P-Layer applies

$$(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15) \mapsto (0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11),$$

while P-Layer^{-1} uses the inverse permutation $(0, 13, 10, 7, 4, 1, 14, 11, 8, 5, 2, 15, 12, 9, 6, 3)$.

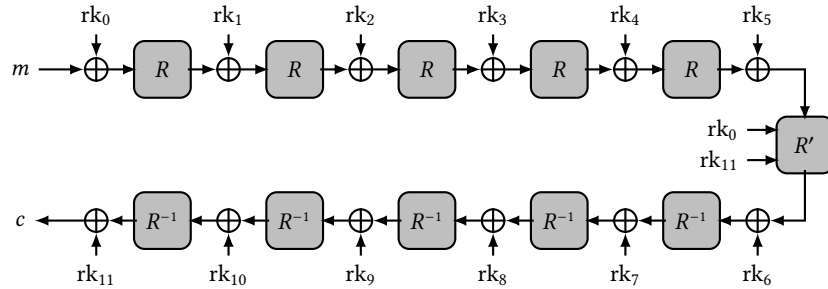


Figure 4: PRINCEv2 encryption.