Practical homomorphic evaluation of block-cipher-based hash functions with applications

Adda Akram Bendoukha[1], Oana Stan[1], Renaud Sirdey[1], Nicolas Quero[1,2⋆], and Luciano Freitas[3]

[1]Université Paris-Saclay, CEA-List, F-91120 Palaiseau, France
firstname.lastname@cea.fr
[2]Expleo, Saint-Quentin-en-Yvelines, France
nicolas.quero@expleogroup.com
[3]LTCI, Télécom Paris, Institut Polytechnique de Paris, France
lfreitas@telecom-paris.fr

Abstract. Fully homomorphic encryption (FHE) is a powerful cryptographic technique allowing to perform computation directly over encrypted data. Motivated by the overhead induced by the homomorphic ciphertexts during encryption and transmission, the transciphering technique, consisting in switching from a symmetric encryption to FHE encrypted data was investigated in several papers. Different stream and block ciphers were evaluated in terms of their "FHE-friendliness", meaning practical implementations costs while maintaining sufficient security levels. In this work, we present a first evaluation of hash functions in the homomorphic domain, based on well-chosen block ciphers. More precisely, we investigate the cost of transforming PRINCE, SIMON, SPECK, and LowMC, a set of lightweight block-ciphers into secure hash primitives using well-established hash functions constructions based on block-ciphers, and provide evaluation under bootstrappable FHE schemes. We also motivate the necessity of practical homomorphic evaluation of hash functions by providing several use cases in which the integrity of private data is also required. In particular, our hash constructions can be of significant use in a threshold-homomorphic based protocol for the single secret leader election problem occurring in blockchains with Proof-of-stake consensus. Our experiments showed that using a TFHE implementation of a hash function, we are able to achieve practical runtime, and appropriate security levels (e.g., for PRINCE it takes 1.28 minutes to obtain a 128 bits of hash).

Keywords: FHE · Hash functions.

1 Introduction

Fully homomorphic encryption (FHE) allows in theory to compute any function over an encrypted input. A plethora of works [5, 16, 20, 26] investigated

[⋆] This author contribution to this work was done while at CEA LIST.

the evaluation of symmetric cryptographic primitives over FHE encrypted keys. The interest in this topic is mainly due to the advent of proxy-re-encryption or transciphering [12], which is a technique that partially solves transmission of massive FHE ciphertexts through limited bandwidth networks, by having the receiver computing an homomorphic decryption of a symmetric cryptosystem. Therefore, many stream and block-ciphers were designed to be efficiently evaluated using an FHE encryption of their key. All the above methods were designed mainly to protect data confidentiality, either through symmetric encryption (for the encryption step and the transmission), or through homomorphic encryption for their processing by an honest-but-curious entity. We argue that there are applications of FHE in which it is useful not only to have confidentiality guarantees but also an integrity check over homomorphically encrypted data. More precisely, in this work we discuss the evaluation of hash functions over an FHE encrypted message and provide several scenarios in which this application can be a solution to achieve integrity along with data privacy. Let us now present the major contributions of our paper.

1.1 Contribution and motivation

In this paper, we present a set of FHE-friendly hash functions built on lightweight block-ciphers using provably-secure constructions, and with reasonable homomorphic execution times. Our choice for a block-cipher-based construction is well motivated and it is the result of investigating several other options, including the homomorphic execution of lightweight hash functions as well as the building of hash functions from FHE-friendly stream-ciphers. As discussed more in details in Section 1.2, the preliminary analysis of several lightweight hash functions candidate to the NIST competition on lightweight cryptography showed that they are not well suited for homomorphic execution. As for the second option, to the best of our knowledge, there is no known practical method to design a secure hash function directly from a stream-cipher (although universal constructions do exist based on Luby-Rackoff theory [31]). As such, we present here some hash function constructions from "FHE-friendly" block-ciphers such as PRINCE [11], LowMC [1] and SIMON [3]. These block-ciphers are interesting candidates to build hash functions, from the homomorphic evaluation point of view, since they have an appropriate design, and have already been implemented with second-generation homomorphic schemes in the context of transciphering.

First, we derive several constructions of hash functions from PRINCE by means of the double block length hash construction, which enables a 128-bits hash size taking into account that the original block size in the PRINCE design is only of 64 bits. We then look into more details and evaluate the performances of a TFHE [15] gate-bootstrapping implementation of these hash functions. Additionally, we leverage on SIMON and LowMC (in their 128-bits block size flavors) to obtain hash sizes of 256 bits via the same construction.

Finally, we describe several use-cases which may require to run our hash functions in the encrypted domain, including integrity checking of homomorphically encrypted data, oblivious authentication, homomorphic database querying, and a FHE-based protocol for single secret leader election.

1.2 Why block cipher-based constructions?

Beside security considerations, when constructing our hash functions, another criteria we looked at was to have a relatively fast evaluation in the homomorphic domain (e.g. less than one minute for a 256-size digest). A first idea for the construction of secure hashes suitable for homomorphic evaluation was to investigate three of the NIST lightweigt competition finalists [17]: SPARKLE [4], XOODYAK [18] and Photon-Beetle [38]. We analysed them in function of the type of homomorphic bitwise operations one should execute: "free" operations such as permutations and concatenations, relatively easy operations such as the XOR and the AND (recall we use mainly TFHE in this work), and more difficult operations such as the modulo. We found out that their underlying primitives (e.g. S-boxes, modulo) and the number of rounds they require makes their homomorphic evaluation too expensive even with a boostrapping-based homomorphic scheme, like TFHE. We also analyzed SPONGENT [8], another lightweight hash function, imposing to execute ≈ 30000 S-box in homomorphic domain (which corresponds to 68 S-boxes per round, 140 required rounds and 32 absorbing and squeezing steps) for 256-bits of output. Taking into account that the execution of the S-box used takes $\approx 0.6sec$ under TFHE, it follows that an homomorphic implementation of the SPONGENT hash function would be too slow to be of practical interest. Further details are available in [36].

Another appealing path was to explore hash-based constructions inspired from "FHE-friendly" stream ciphers. This option was tempting since nowadays there are several practical solutions implementing stream-ciphers into homomorphic domain (e.g. Kreyvium [12], Grain128 [5], PASTA [20]). However, even if it seems possible to obtain hash functions with very interesting homomorphic performances, their security seems difficult to assess and this, thus, remains an interesting open question. In essence, although theoretical constructions do exist, the symmetric cryptography community has, to the best of our knowledge, only marginally followed this path for building hash functions. Still, the possibility of achieving better FHE evalulation performances may be a new motivation for further investigations along this line.

As a consequence, we decided to consider block-cipher algorithms which have been already considered for homomorphic evaluations and turn them into secure hash functions using generic methods such the ones described in Section 2.3.

2 Background

2.1 Transciphering

Transciphering is a technique that allows offloading massive data from client to server with the aim to perform server-side homomorphic computations. Indeed, when a message m is encrypted under an FHE cryptosystem, the resulting size of the ciphertext $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(m)$ is much larger than the size of the original message m, by an expansion factor which depends polynomially on the security parameter λ. In all modern FHE schemes for a λ large enough (in the 110-130 bits of security ranges) ciphertext sizes reach several kbytes or even megabytes (depending on the chosen cryptosystem and its security level). So, instead of encrypting m directly using an FHE scheme and sending $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(m)$, a client will rather encrypt m using a symmetric cryptosystem and sends the encryption $\mathsf{SYM.Enc}_{\mathsf{SYM.sk}}(m)$ to the server along with $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(\mathsf{SYM.sk})$, the FHE encryption of the symmetric key $\mathsf{SYM.sk}$. The server then homomorphically runs $\mathsf{SYM.Dec}_{\mathsf{FHE.Enc(SYM.sk)}}(\mathsf{SYM.Enc}(m))$ and recovers the message encrypted under the homomorphic public key $\mathsf{FHE.Enc}_{\mathsf{FHE.pk}}(m)$.

$\mathsf{SYM.Enc}_{\mathsf{SYM.sk}}(m)$ is roughly of the same size as m while $\mathsf{SYM.sk}$, which is the only FHE encrypted and transmitted element, is of fixed size and often small enough to be homomorphically encrypted and sent (once and offline) through the network, whilst m can be arbitrarily large. Switching from a symmetric scheme to an FHE one allows a form of secure *compression* of the homomorphic ciphertexts. It requires however, the evaluation of $\mathsf{SYM.Enc}$ homomorphically, which introduces a non-negligible additional computational overhead on the server-side. In [5,12], it is argued that the use of a stream-cipher is more suitable for transciphering in the case of both 2nd generations FHE schemes (e.g., BGV, BFV) as well as TFHE. In [20] authors discuss the semantic security of transciphering seen as Key encapsulation / Data encapsulation mechanism (KEM-DEM) depending on the semantic security of both the symmetric and homomorphic schemes involved, and provide also an FHE-freindly stream-cipher named Pasta, suited for levelled FHE schemes.

2.2 Hash functions and security properties

A general definition of a hash function is a mapping of messages of arbitrary length to a fixed size digest. Additionally, a *cryptographic* hash function requires the following security properties.

Pre-image resistance. Given $h \in \{0,1\}^n$ the output of the hash function $H : \{0,1\}^* \to \{0,1\}^n$, it must be computationally hard to find $m \in \{0,1\}^*$ such that $H(m) = h$.

Collision resistance. It must be computationally hard to find two distinct messages m_1 and m_2 such that $H(m_1) = H(m_2)$.

Second pre-image resistance. Given m and h such that $H(m) = h$, it must be computationally hard to find m' such that $m' \neq m$ and $H(m) = H(m')$.

Since we only consider cryptographic hash functions, for simplicity sake, in the remaining of the paper we will refer to a ”cryptographic hash function” as ”a hash function”.

Black-Box model. To prove the security of a block-cipher-based hash function independently of the underlying cipher's structure, it is used the black-box model, in which a block-cipher is modeled as an invertible random permutation defined by the key. An adversary is given access to encryption and decryption oracles, such that given m (resp. c) the encryption $E_k(m)$ (resp. the decryption $E_k^{-1}(c)$) is returned. The complexity of an attack is measured by the number of encryption and decryption queries that an optimal adversary performs. Since most attacks on block-cipher-based hash functions do not take advantage of the block-cipher's potential structural weaknesses or flaws, it is relevant to use a black-box model for security analysis.

2.3 Block-cipher-based hash functions

Among the most widely used constructions of hash functions are the iterated hash functions, in which a round function, also referred to as a compression function $F : \{0,1\}^n \cdot \{0,1\}^l \rightarrow \{0,1\}^n$ is iterated over every message block, taking as input the current message block of size n and the previous hash value[1]. The output of the final compression function call is the hash of the input message as shown in Alg. 1. Due to its simplicity, this construction has been intensively studied in the state of the art [6, 30], giving birth to many hashing standards such as SHA-0, SHA-1, and SHA-2. A large part of the security of these hash functions can be attributed to the underlying compression function [2]. In [19] authors demonstrate that the collision resistance of F implies collision resistance of the hash function built from F using the Merkle-Damgård construction.

These results raised interests in building secure compression functions from which it will be easy to build secure hash functions. A block-cipher is a primitive that already provides security properties by construction. Although the security requirements of an encryption algorithm are different by nature from those of a hash function, the question of how to build a secure compression function from a block-cipher quickly appeared and was intensively investigated, laying foundation for instance for the MDC family of hash functions [34] based on the block-cipher DES. The main motivation of this approach is to minimize design efforts, and use existing primitives. The task is to transform the security properties of a block-cipher into those of a cryptographic hash function, by carefully executing it over well-chosen linear combinations of the current message block, the chaining variable, or other conventional constants, taken as encryption keys or message blocks. This gave birth to a plethora of constructions, some of them were proven secure in the black-box model, others exhibited weaknesses regardless of the underlying block-cipher's potential weaknesses.

One important security element is the size of the digest. Due to the birthday paradox, collision security level of a hash function is upper-bounded by $O(2^{n/2})$, where n is the size of the hash. Thus, having a size for the hash equal to the size of the block for the cipher used to construct the compression function raised some

[1] A chaining value to provide dependency between successive hash values.

[2] The security under all aspects : Pre-image, second pre-image, and collision resistance.

issues. The size of some block-cipher's blocks can be too small to be considered as a secure hash size, and using a block-cipher with a large block length often results in higher execution times. Providing a secure construction that produces a hash twice larger than the block-cipher's block length was subject to several research efforts.

Algorithm 1 Merkle Damgård iterated hash function

```
  input : $m = (m_0, m_1, \cdots, m_l)$
  $h_0$ is set to an initialization vector
  for i = 0 to $l$ do
      $h_i = F(h_{i-1}, m_i)$
  end for
return h$_l$
```

Single Block Length (SBL) hash functions. One of the very first constructions of single-block-length hash functions is the Davies-Meyer construction where $H_{i+1} = E_{M_i}(H_i) \oplus M_i$ and the Muguiyachi-Prennel's scheme with $H_{i+1} = E_{M_i}(H_i) \oplus M_i \oplus H_i$, where H_i is the previous hash value and each block of the message (M_i) is the key to a block cipher E.

Later, in [35], Prennel, Govaerts and Vandewalle (PGV) provided an exhaustive analysis of iterated hash functions defined over $\{0,1\}^* \to \{0,1\}^n$ and based on a block-cipher. The compression function is in the form $F(a,b) = E_a(b) \oplus c$ where a, b and c are in $\{m_i, h_{i-1}, IV, m_i \oplus h_{i-1}\}$, and E is $\{0,1\}^n \cdot \{0,1\}^n \to \{0,1\}^n$ block-cipher. There are $4^3 = 64$ such compression functions, among which 12 are presented as secure. Afterwards, Black, Rogaway and Shrimpton [7] provided formal security proofs in the black-box model of the 12 constructions analysed in [35]. They also demonstrated that among the remaining 52 constructions, 8 of them were actually secure with respect to collision and pre-image resistance. In this work we chose to evaluate Davies-Meyer's hash function under several block-ciphers, as it provides optimal security in the black-box model, and is equivalent in terms of computation complexity to other secure constructions from [7].

The security analysis and explicit constructions are provided in [7].

Double Block Length (DBL) hash functions. As mentioned before, constructions by PGV provide a hash of n-bits size when using a $\{0,1\}^n \cdot \{0,1\}^n \to \{0,1\}^n$ underlying block-cipher in the compression function. Due to the birthday paradox, these hash functions require block-ciphers with a large enough block length in order to provide security against collision attacks.

A measure of the efficiency of a hash function is its rate, that is, the inverse of the number of calls to the compression function per iteration.

In [33] Merkle presents three optimally collision resistant double block length hash functions, based on the block-cipher DES. However, their rates are low compared to the next generation of DBL constructions.

Lai and Massey proposed TANDEM-DM [28] for a rate 1/2 hash construction, using a $(n, 2n)$ block-cipher. It was proven optimally collision and pre-image secure in [21]. It makes however two non-independent calls[3] per iteration making it non-parallel. Abreast-DM [29] is another construction with a rate of 1/2 making two parallel calls to the block-cipher, and was proven to have optimal collision resistance in [22].

Lucks in [5] provides a first DBL construction of rate 1. Making a single block-cipher call per iteration comes at the cost of computing a heavy linear combination of the message block and the previous hash resulting in a significant overhead. Hirose in [25] provides a rate 1/2 construction with two distinct $(n, 2n)$ block-ciphers, then uses a tweak in order to use a single block-cipher. This construction provides optimal bounds for both collision and pre-image resistance in the black-box model, and is parallel. Indeed, the two calls to the compression function (and thus, to the block-cipher) are independent, making its performance comparable to rate 1 constructions.

Other works from [27, 32] studied the possibility to build DBL hash functions from an (n, n)-block-cipher. MDC-2 fails to provide optimal security, while MDC-4 [32] is near optimal, but has a rate smaller than 1/2.

In this work, we homomorphically evaluate the constructions of Hirose and Tandem-DM. The goal is to provide an idea of the runtime of two optimally secure hash functions of rate 1/2 from both the parallel and non-parallel types on top of an FHE encryption layer.

3 Applications of homomorphic hash functions

3.1 Homomorphic data integrity check

As described in Section 2.1, transciphering allows to transfer symmetrically encrypted data instead of homomorphically encrypted and thus reduces the required bandwidth. However, transciphering while preserving data privacy does not ensure data integrity during transmission. In [5] authors describe how to include data integrity check within transciphering, but their approach required an AEAD encryption scheme (Authenticated Encryption with Associated Data).

Indeed, all stream-ciphers suffer from malleability, i.e., the possibility for an adversary to create an encryption of $m + k$ where k is some constant, from an encryption of m[4]. A malleable encryption scheme can be subject to man-in-the-middle attacks. Some modern stream-ciphers (e.g. [24]) come with the possibility to compute a MAC (Message Authentication Code) along with the encryption in an attempt to circumvent this issue. Another simple way to perform integrity

[3] The output of the first block-cipher call is used to build the key of the second block-cipher call.

[4] $m \oplus keystream \oplus k = \mathsf{SYM.Enc}(m \oplus k)$

check within transciphering when the chosen stream-cipher does not embed a MAC computation is to include a hash function. A client encrypts m concatenated to $H(m)$ using a symmetric encryption scheme. She then transmits these elements to the server along with $\mathsf{FHE.Enc}(\mathsf{SYM.sk})$ (once and for all). Once the server has finished transciphering both the message and the hash, it recovers $\mathsf{FHE.Enc}(m')$ and $\mathsf{FHE.Enc}(h')$, he computes $[h] = H(\mathsf{FHE.Enc}(m'))$. If $m = m'$ then $h = h'$ (with overwhelming probability, of course). The server computes the homomorphically encrypted bit $[r]_{\mathsf{FHE.pk}} = \prod_{i=0}^{n}(1 \oplus h_i \oplus h'_i)$ where n is the size of the hash. $[r]_{\mathsf{FHE.pk}}$ is the output of the integrity check, such that :

$$[r] = \left[\begin{cases} \text{An encryption of } 1 & \text{if } m = m' \\ \text{An encryption of } 0 & \text{otherwise} \end{cases} \right], \tag{1}$$

This FHE encrypted bit could then be used in many ways. The server can simply transmit it to the client, in order to give him the ability to verify if his data was altered or corrupted during the transmission. Or the server could choose to reply with $[f(m)]$ or a NIL value outside of the range of f, according to the value of the bit r. In TFHE [15] for example, this can be realized using a homomorphic CMUX gate at roughly the cost of an extra homomorphic multiplication[5].

3.2 Single Secret Leader Election (SSLE)

The problem of securely electing a single leader in a distributed system was formally defined by Boneh et al. in [9]. For a committee of peers which collaboratively elect a node to complete a task, the problem consists in electing a node in a way that only this elected peer is able to know that he was elected and the others learn only that they were not elected. Also, the elected peer must be able to provide a proof of his election when he decides to reveal himself once his task is done. In [23] a solution to the SSLE problem is proposed based on Threshold Fully Homomorphic encryption [10] for partially-synchronous systems. A very high level description is the following. Every peer P_i wishing to register to the election at a given height and cycle (low and high level steps in the leader election protocol), provides an FHE encryption of $p_i = H(h||t_i||c)$ called the proof, where h is the height of the blockchain, c the current cycle of elections, and t_i a locally generated number belonging to process P_i. Every participating peer performs a sampling circuit following a weighted distribution over the FHE encrypted list of proofs and ids of all registered peers, using collaboratively generated randomness from [37]. Then, each peer homomorphically selects[6] a proof and the associated id from the set of all proofs. He then homomorphically hashes $(p_i||i)$ where i is the id of the elected peer, and p_i the corresponding proof. The next step is to broadcast a partial decryption of the voucher $v_{h,c,r} = H(p_i||i)$. Every honest peer samples the same p_i and i, and broadcasts his partial decryptions

[5] $\text{CMUX}([r], [f(m)], NIL]) = [r] \cdot [f(m)] + (1 - [r]) \cdot NIL$

[6] I.e., homomorphically computes a one-hot encoding of an index in the proofs list and performs a dot-product to extract one such proof, thus without knowing which one.

of $v_{h,c,r}$ using his secret key share. Assuming we have at least t honest peers in the system, where t is the decryption threshold, every peer must eventually receive enough partial decryptions and be able to perform a full decryption of $v_{h,c,r}$. The elected peer recognizes his *voucher*, whereas other peers gain no information from plain $v_{h,c,r}$, nor can fake the election, since H is secure against pre-image and second pre-image attacks. Afterward, the leader is able to prove his election by submitting his plaintext proof $p_i = H(h||t_i||c)$. The verification is simply performed by running the test $H(p_i||i) == v_{h,c,r}$.

The homomorphically evaluated hash function plays a significant role in this protocol. It hides the sensitive elements from Byzantine peers providing the secrecy of the election and a simple proof mechanism, making the election easily verifiable, yet computationally hard to forge fake proofs.[7]

3.3 Homomorphic Database querying

Suppose a server maintaining a database of elements DB such that query m has the answer A_m stored at index $H(m)$, where H is a hash function (with a small digest size w. r. t. to cryptographic standards). In this case H is not necessarily cryptographic. For instance pre-image resistance is not necessary since the query is already private under an FHE encryption layer. Nevertheless, we require from H to have balanced collisions[8] and, for this sake, one can use Luby-Rackoff's universal hash functions from [31]. In this setting, the server is able to homomorphically answer FHE encrypted queries.

A client homomorphically encrypts a query x and sends $[x]_{\mathsf{FHE.pk}}$ to the server. The server computes $[i]_{\mathsf{FHE.pk}} = H([x]_{\mathsf{FHE.pk}})$, which is an FHE encryption of the index of A_x inside his database. The server then computes a vector V which contains FHE encryptions of 0 everywhere except at index i in which an encryption of 1 is stored. V is computed as follows : $V[k] = ([i]_{\mathsf{FHE.pk}} == k)$ with $k \in [\![0, n-1]\!]$. Lastly, to extract an FHE encryption of A_x, the server performs a homomorphic dot product between the vector V and his database of elements $\sum_{i=0}^{n} DB[i] \cdot V[i]$, and sends back to the client the result of this final dot product, which will be $[A_x]_{\mathsf{FHE.pk}}$.

One remaining problem of this use-case is to homomorphically resolve collisions of H. A first approach is to have the server creating lists of answers to different queries which hash to the same index at the position $H(x)$ in DB, and provide a second hash function H', whose output is smaller than the one of H, and which will compute the index of A_x inside the corresponding list. Thus, when an FHE encrypted query $[x]_{\mathsf{FHE.pk}}$ is received, the position $(H([x]_{\mathsf{FHE.pk}}), H'([x]_{\mathsf{FHE.pk}}))$ provides an answer.

[7] Secrecy is granted by the pre-image resistance of the hash function. Having a single verifiable leader is due to the second pre-image resistance of the hash function.

[8] $H : \{0,1\}^* \to \{0,1\}^n$ has balanced collisions if all elements in $\{0,1\}^n$ have the same number of pre-images under H.

3.4 Oblivious authenticated (homomorphic) calculations

It is well known that (keyed) hash functions are used in many authentication protocols where an entity (the user) can prove its knowledge of a secret (the key of the hash function) to another entity (the server). To do so, the server sends a random challenge to the user which replies with the hash of the challenge. Since the server can also perform the same calculation, it can check the correctness of the client replies which proves the latter knowledge of the secret. With the ability of running hash functions in the homomorphic domain, we can now provide the server with an FHE encryption of the secret key and have the server performing the authentication in the encrypted domain i.e., the server generates a challenge in the clear domain, sends it to the user and get its (non encrypted) reply. The server can then run the (keyed) hash function homomorphically on its challenge, and homomorphically compare the obtained (encrypted) result with the reply received from the user. As the end of this process, the server possesses an encrypted boolean, say β, indicating whether or not the client has successfully authenticated (but has by construction no knowledge of whether or not that authentication was successful).

One way of using this consists in providing a valid calculation only to successfully authenticated users. In essence, rather than computing $f(x)$ in the homomorphic domain, the server can now compute $\beta f(x) + (1 - \beta)\perp$ (where $\perp$ denotes a constant value meaning, by convention, “not an answer”). As a consequence, (encrypted) valid calculation results are duly returned only to authenticated users, while other users receive only useless encryptions of $\perp$. This is then (nicely) done obliviously to the server, which cannot distinguish between ciphertexts of valid results and ciphertexts of $\perp$, and without revealing the secret hash function key (since it is only provided with an FHE-encryption of that secret key).

4 Adaptations of block ciphers for FHE-friendly hashes

4.1 Block-ciphers considered

The Low-MC block-cipher [1] is part of a family of symmetric schemes designed for practical instantiations in homomorphic domain with the objectives of minimizing both the multiplicative complexity and the multiplicative depth, making it efficient for levelled homomorphic schemes. This design principle, had to be compensated with a large number of xor gates in order to ensure algebraic properties that provide an appropriate level of security. This latter fact makes it rather inefficient when ran under TFHE, since the cost of all Boolean homomorphic gates is the same within this FHE scheme (A bootstrapping operation is performed after every Boolean gate). It remains however a interesting candidate for hash constructions targeting efficient homomorphic evaluation in a levelled FHE setting. Even if the first variants of LowMC were successfully attacked, the subsequent proposed design is more secure and highly parametrizable. In particular, there is a closed-form formula to determine the minimal number of rounds to reach a given security target depending on the block size (128 or 256 bits), the key size, the number of S-boxes and the allowed data complexity.

PRINCE [11], SPECK and SIMON [3] are lightweight block-ciphers, with a relatively small block length. They were initially designed for constrained embedded execution environments. Their design approaches result in small gate counts[9] which results in high performances when ran under TFHE. Due to its small block length, PRINCE is better suited with double block length constructions, resulting in a hash function which provides $O(2^{64})$ collision resistance, and $O(2^{128})$ for pre-image resistance. SPECK and SIMON can be instantiated in both the DBL and SBL settings since they both provide a double-key-size variants.

4.2 FHE schemes considered

We chose to run our experiments under the TFHE cryptosystem since it provides the possibility to evaluate (multiplicatively) unbounded homomorphic circuits thanks to its fast bootstrapping operation. This scheme is more suited for protocols where scalability is a requirement. For example, the secret single leader election protocol [9] described in section 3.2 requires flexibility regarding the number of peers being able to disconnect or join the committee at different times. These variations in the number of peers linearly increase the multiplicative depth of the sampling circuit, which would be difficult to manage if a levelled homomorphic scheme were to be used [10].

4.3 Tool: Cingulata Homomorphic Compiler

Cingulata, formerly known as Armadillo [14], is a toolchain and run-time environment (RTE) for implementing applications running over homomorphic encryption. Cingulata provides high-level abstractions and tools to facilitate the implementation and the execution of privacy-preserving applications.

Cingulata relies on instrumented C++ types to denote private variables, e.g., `CiInt` for integers and `CiBit` for Booleans. Integer variables are dynamically sized and are internally represented as arrays of `CiBit` objects. The Cingulata environment monitors/tracks each bit independently. Integer operations are performed using Boolean circuits, which are automatically generated by the toolchain. For example a full-adder circuit is employed to perform an integer addition. The Boolean circuit generation is configurable and two generators are available: focused on minimal circuit size or on small multiplicative-depth. More generally, it is possible to implement additional circuit generators or to combine them.

[9] A round of encryption of a block-cipher often includes a multiplication of the internal state with an F^2 matrix, this makes the number of operations quadratic with respect to its block size.

[10] In this category of homomorphic schemes, the multiplicative depth of the homomorphic circuit to be evaluated has to be known in advance in order to generate a parameter set which allows homomorphic computations up to this depth.

A `CiBit` object can be in either plain or encrypted state. Plain-plain and plain-encrypted bit operations are optimized out, in this way constant folding and propagation is automatically performed at the bit-level. Bit operations between encrypted values are performed by a “bit execution” object implementing the `IBitExec` interface. This object can either be a HE library wrapper, simply a bit-tracker object or even a plaint bit execution used for algorithm debugging purposes. When a HE library wrapper is used the Cingulata environment directly executes the application using the underlying HE library.

Another option is to use the bit-tracker in order to build a circuit representation of the application. This allows to use circuit optimization modules in order to further optimize the Boolean circuit representation. The hardware synthesis toolchain ABC[11] is used to minimize circuit size. It is an open-source environment providing implementations of state-of-the-art circuit optimization algorithms. These algorithms are mainly designed for minimizing circuit area or latency but, currently, none of them is designed for multiplicative depth minimization. In order to fill this gap, several heuristics for minimizing the multiplicative depth are available in Cingulata, refer to [2, 13] for more details.

The optimized Boolean circuit is then executed using Cingulata’s parallel run-time environment. The RTE is generic, meaning that it uses a HE library wrapper, i.e. a “bit execution” object as defined earlier, in order to execute the gates of the circuit. The scheduler of the run-time allows to fully take advantage of many-core processors. Besides, a set of utility applications are provided for parameter generation (given a target security level), key generation, encryption and decryption. These applications are also generic, in the same vein as the parallel RTE.

4.4 Experimental results and performances

We ran *multi core* performance tests on an Intel(R) Xeon(R) CPU E3-1240 v5 @ 3.50GHz and 8GB RAM using Cingulata in TFHE mode. We provide parallelism when possible using the OpenMP library.

For single block length construction, we implement Davies-Meyer’s compression function which requires a (n, n)-block-cipher. Therefore, we instantiate this construction with SPECK, SIMON [12], and the (128, 128) variant of LowMC. In the double block length setting, since these constructions require an $(n, 2n)$-block-cipher, we instantiate Hirose’s and Tandem-DM constructions with PRINCE, and the (128, 256) variants of LowMC, and SIMON. The results are shown in Table 4.4 with the execution times in minutes when the hash functions are instantiated, and an ”-” symbol when the construction is not compatible with the sizes of the key and the block of the cipher.

The obtained performances are as expected : lightweight ciphers provide better runtimes compared to LowMC. PRINCE is the most efficient cipher for DBL

[11] http://people.eecs.berkeley.edu/alanmi/abc/

[12] For SIMON, these are estimations based on the gate count from [3] and the gate-bootstrapping time of TFHE

constructions as it has the lowest gate-count, and is also the most parallelizable cipher. The number of rounds performed in every construction to produce the hash of a 128-bits message is $\lceil \frac{128}{blocklength} \rceil$. Thus, in the first row, DBL-PRINCE performs two iterations and produces a 128-bits hash. All the remaining constructions perform a single iteration.

Table 1. Evaluation of hash functions over a 128-bits TFHE encrypted message in minutes

Instantiation	Davies-Meyer (SBL)	Hirose (DBL)	Tandem-DM (DBL)
(64, 128)-PRINCE	-	1.28	2.98
(128, 128)-SPECK	3.78	-	-
(128, 256)-SPECK	-	4.91	8.16
(128, 128)-SIMON	2.14	-	-
(128, 256)-SIMON	-	3.64	7.05
(128, 128)-LowMC	6.12	-	-
(128, 256)-LowMC	-	8.58	17.32

5 Conclusion and perspectives

In this work, we have investigated scenarios in which the ability to (efficiently) evaluate hash functions in the homomorphic domain is an interesting building block. To the best of our knowledge, this work is one of the first to address this issue, at least for the TFHE cryptosystem. We also explored various provably-secure constructions of "(T)FHE friendly" hash functions based on respected block-ciphers in order to achieve several digest sizes. Fully homomorphic encryption on its own opens perspectives towards a new set of applications. Then, combining it with the execution of hash functions in the homomorphic domain provides it with additional versatility which can serve in various scenarios and protocols.

References

1. Albrecht, M., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for mpc and fhe. Cryptology ePrint Archive, Paper 2016/687 (2016), https://eprint.iacr.org/2016/687, https://eprint.iacr.org/2016/687
2. Aubry, P., Carpov, S., Sirdey, R.: Faster homomorphic encryption is not enough: improved heuristic for multiplicative depth minimization of boolean circuits. In: CT-RSA. pp. 345–363 (2020)
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Paper 2013/404 (2013), https://eprint.iacr.org/2013/404, https://eprint.iacr.org/2013/404

4. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: Schwaemm and esch: Lightweight authenticated encryption and hashing using the sparkle permutation family
5. Bendoukha, A.A., Boudguiga, A., Sirdey, R.: Revisiting stream-cipher-based homomorphic transciphering in the TFHE era. In: Aïmeur, E., Laurent, M., Yaich, R., Dupont, B., Garcia-Alfaro, J. (eds.) Foundations and Practice of Security. pp. 19–33. Springer International Publishing, Cham (2022)
6. Biham, E., Dunkelman, O.: A framework for iterative hash functions - haifa. Cryptology ePrint Archive, Paper 2007/278 (2007), https://eprint.iacr.org/2007/278, https://eprint.iacr.org/2007/278
7. Black, J., Rogaway, P., Shrimpton, T.: Black-box analysis of the block-cipher-based hash-function constructions from pgv. p. 320–335. CRYPTO '02, Springer-Verlag, Berlin, Heidelberg (2002)
8. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: Spongent: A lightweight hash function
9. Boneh, D., Eskandarian, S., Hanzlik, L., Greco, N.: Single secret leader election. Cryptology ePrint Archive, Paper 2020/025 (2020), https://eprint.iacr.org/2020/025, https://eprint.iacr.org/2020/025
10. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. Cryptology ePrint Archive, Paper 2017/956 (2017), https://eprint.iacr.org/2017/956, https://eprint.iacr.org/2017/956
11. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knežević, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: Prince - a low-latency block cipher for pervasive computing applications (full version). Cryptology ePrint Archive, Paper 2012/529 (2012), https://eprint.iacr.org/2012/529, https://eprint.iacr.org/2012/529
12. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression
13. Carpov, S., Aubry, P., Sirdey, R.: A multi-start heuristic for multiplicative depth minimization of boolean circuits. In: IWOCA. pp. 275–286 (2017)
14. Carpov, S., Dubrulle, P., Sirdey, R.: Armadillo: A compilation chain for privacy preserving applications. In: Bao, F., Miller, S., Chow, S.S.M., Yao, D. (eds.) Proceedings of the 3rd International Workshop on Security in Cloud Computing, SCC@ASIACCS '15, Singapore, Republic of Singapore, April 14, 2015. pp. 13–19. ACM (2015). https://doi.org/10.1145/2732516.2732520, https://doi.org/10.1145/2732516.2732520
15. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Paper 2018/421 (2018), https://eprint.iacr.org/2018/421, https://eprint.iacr.org/2018/421
16. Cho, J., Ha, J., Kim, S., Lee, B., Lee, J., Lee, J., Moon, D., Yoon, H.: Transciphering framework for approximate homomorphic encryption (full version). Cryptology ePrint Archive, Paper 2020/1335 (2020), https://eprint.iacr.org/2020/1335, https://eprint.iacr.org/2020/1335
17. https://csrc.nist.gov/Projects/Lightweight Cryptography: Nist lightweight cryptography
18. Daemen, J., Hoffert, S., Peeters, M., Assche, G.V., Keer, R.V.: Xoodyak and a lightweight cryptographic scheme

19. Damgard, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 416–427. Springer New York, New York, NY (1990)
20. Dobraunig, C., Grassi, L., Helminger, L., Rechberger, C., Schofnegger, M., Walch, R.: Pasta: A case for hybrid homomorphic encryption. Cryptology ePrint Archive, Paper 2021/731 (2021), https://eprint.iacr.org/2021/731, https://eprint.iacr.org/2021/731
21. Fleischmann, E., Gorski, M., Lucks, S.: On the security of tandem-dm. In: Dunkelman, O. (ed.) Fast Software Encryption. pp. 84–103. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
22. Fleischmann, E., Gorski, M., Lucks, S.: Security of cyclic double block length hash functions. In: Parker, M.G. (ed.) Cryptography and Coding. pp. 153–175. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
23. Freitas, L., Tonkikh, A., Tucci-Piergiovanni, S., Sirdey, R., Stan, O., Quero, N., Bendoukha, A.A., Kuznetsov, P.: Homomorphic sortition – secret leader election for blockchain (2022). https://doi.org/10.48550/ARXIV.2206.11519, https://arxiv.org/abs/2206.11519
24. Hell, M., Johansson, T., Meier, W., Sönnerup, J., Yoshida, H.: An aead variant of the grain stream cipher. In: Carlet, C., Guilley, S., Nitaj, A., Souidi, E.M. (eds.) Codes, Cryptology and Information Security. pp. 55–71. Springer International Publishing, Cham (2019)
25. Hirose, S.: Provably secure double-block-length hash functions in a black-box model. In: Park, C., Chee, S. (eds.) Information Security and Cryptology - ICISC 2004, 7th International Conference, Seoul, Korea, December 2-3, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3506, pp. 330–342. Springer (2004). https://doi.org/10.1007/11496618_24, https://doi.org/10.1007/11496618_24
26. Hoffmann, C., Méaux, P., Ricosset, T.: Transciphering, using filip and tfhe for an efficient delegation of computation. In: Bhargavan, K., Oswald, E., Prabhakaran, M. (eds.) Progress in Cryptology – INDOCRYPT 2020. pp. 39–61. Springer International Publishing, Cham (2020)
27. Jetchev, D., Özen, O., Stam, M.: Collisions are not incidental: A compression function exploiting discrete geometry. In: Cramer, R. (ed.) Theory of Cryptography. pp. 303–320. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
28. Lai, X., Massey, J.L.: Hash functions based on block ciphers. In: Rueppel, R.A. (ed.) Advances in Cryptology — EUROCRYPT' 92. pp. 55–70. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
29. Lee, J., Kwon, D.: The security of abreast-dm in the ideal cipher model. Cryptology ePrint Archive, Paper 2009/225 (2009), https://eprint.iacr.org/2009/225, https://eprint.iacr.org/2009/225
30. Lei, D., Lin, D., Chao, L., Feng, K., Qu, L.: The design principle of hash function with merkle-damgård construction. Cryptology ePrint Archive, Paper 2006/135 (2006), https://eprint.iacr.org/2006/135, https://eprint.iacr.org/2006/135
31. Luby, M.: Pseudorandomness and Cryptographic Applications (01 1996). https://doi.org/10.2307/j.ctvs32rpn
32. Mennink, B.: On the collision and preimage security of mdc-4 in the ideal cipher model. Cryptology ePrint Archive, Paper 2012/113 (2012), https://eprint.iacr.org/2012/113, https://eprint.iacr.org/2012/113
33. Merkle, R.C.: One way hash functions and des. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. pp. 428–446. Springer New York, New York, NY (1990)

34. Preneel, B.: MDC-2 and MDC-4, pp. 771–772. Springer US, Boston, MA (2011). https://doi.org/10.1007/978-1-4419-5906-5$_5$96, $https://doi.org/10.1007/978-1-4419-5906-5_596$
35. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: CRYPTO (1993)
36. Quero, N.: Etude des fonctions de hachage homomorphes pour un protocole d'élection secrète pour la blockchain (2022), internship report. REF: LIST/DSCIN/21-0189/NQ.
37. de Souza, L.F., Tucci Piergiovanni, S., Sirdey, R., Stan, O., Quero, N., Kuznetsov, P.: Randsolomon: optimally resilient multi-party random number generation protocol. CoRR **abs/2109.04911** (2021), https://arxiv.org/abs/2109.04911
38. Zhenzhen BGao Nanyang Technological University, S., Avik Chakraborti NTT Secure Platform Laboratories, J., Nilanjan Datta Indian Statistical Institute, Kolkata, I., Jian Guo Nanyang Technological University, S., Mridul Nandi Indian Statistical Institute, Kolkata, I., Thomas Peyrin Nanyang Technological University, S., Kan Yasuda NTT Secure Platform Laboratories, J.: The photon family of lightweight hash functions