ZesT: an all-purpose hash function based on Zémor-Tillich

Christophe Petit[1] ⋆, Giacomo de Meulenaer[1], Jean-Jacques Quisquater[1], Nicolas Veyrat-Charvillon[1], Jean-Pierre Tillich[2] and Gilles Zémor[3]

1
UCL Crypto Group⋆⋆
Université catholique de Louvain
Place du levant 3
1348 Louvain-la-Neuve, Belgium

2
Equipe SECRET
INRIA Rocquencourt
78153 Le Chesnay, France

3
Institut de Mathématiques de Bordeaux
Université Bordeaux 1
351, cours de la Libération
33405 Talence, France

e-mails:
`christophe.petit@uclouvain.be,giacomo.demeulenaer@uclouvain.be,jjq@uclouvain.be,`
`nicolas.veyrat@uclouvain.be,jean-pierre.tillich@inria.fr,Gilles.Zemor@math.u-bordeaux1.fr`

Abstract. Hash functions are a very important cryptographic primitive. The collision resistance of *provable* hash functions relies on hard mathematical problems. This makes them very appealing for the cryptographic community since collision resistance is by far the most important property that a hash function should satisfy. However, provable hash functions tend to be slower than specially-designed hash functions like SHA, and their algebraic structure often implies homomorphic properties and weak behaviors on particular inputs.

We introduce the ZesT hash function, a provable hash function that is based on the Zémor-Tillich hash function. ZesT is provably collision and preimage resistant if the balance problem corresponding to Zémor-Tillich is hard, a problem that has remained unbroken since CRYPTO'94. The function admits an ultra-lightweight implementation in ASIC and it is currently between 2 to 3 times less efficient than SHA on FPGA, and between 4 to 10 times slower than SHA in software. The function has structural parallelism, and its simplicity will certainly allow a much wider range of implementations and many code optimization techniques. A careful examination and pseudorandom tests performed with the Dieharder revealed no apparent malleability weakness, which suggests that the function can be used as a general-purpose hash function. Finally, ZesT can be slightly modified to reach all the requirements of the NIST competition.

We stress that the hardness of the balance problem corresponding to Zémor-Tillich should be further studied and better established by the cryptography community. In that case, our function ZesT will definitely become a very appealing all-purpose hash function.

⋆ Research Fellow of the Belgian Fund for Scientific Research (F.R.S.-FNRS) at Université catholique de Louvain (UCL).

⋆⋆ A member of BCRYPT network.

1 Introduction

"Provable" hash functions, whose collision resistance relies on hard mathematical problems, are very appealing. Collision resistance is by far the most important property that a hash function should satisfy. A mathematical problem can be studied independently outside the cryptographic community and the confidence in the collision resistance of the hash function may increase with the understanding of the problem. In particular, functions based on the factorization problem [25,26,17,40] or on the discrete logarithm problem [17,13] provide a very strong level of confidence.

On the other hand, provable hash functions have important issues that have impended their broad use in practice. Provable hash functions tend to be very slow compared to specially designed hash functions like SHA. This effect tend to disappear in recent provable hash function proposals like VSH [16] and SWIFFT [32], but the corresponding hard problems are only variants of standard factorization and lattice problems. Moreover, existing provable hash functions have a rich mathematical structure implying homomorphic properties and weak behaviors on particular inputs. In general, provable hash functions should only be used in applications requiring no more than collision resistance.

Hash functions are the Swiss army knives of Cryptography: they are supposed to possess a lot of functionalities for a lot of different applications. In practice, it would be dangerous to publicize a collision resistant hash function with certain weak behaviors because the function might be (wrongly) used by engineers without any background in Cryptology. The efficiency issue is also important since there are applications requiring high-speed or low-area implementations. At the light of most existing proposals, provable hash functions seem therefore not practical at all. However, the ZesT hash function presented here is a provably collision resistant hash function that suits all applications.

ZesT is Zémor-Tillich with Enhanced Security inside. It is essentially the vectorial version of the Zémor-Tillich hash function iterated twice. The Zémor-Tillich hash function is a provable hash function based on a non-standard assumption that has resisted 15 years of cryptanalysis since its publication at CRYPTO'94 [42]. Its vectorial version was introduced in [36] and shown to be as secure as the original function. The ZesT function preserves the *provable* collision resistance of the Zémor-Tillich hash function but it also *heuristically* satisfies good pseudorandom properties.

We argue that this approach is meaningful as it combines the main advantages of the fully heuristical approach and the fully theoretical approach. ZesT has *heuristic* pseudorandom properties comparable to the heuristic pseudorandom properties of custom designed hash functions and it is moreover *provably* secure with respect to the crucial notion of collision resistance. It is also by far more efficient than any provably pseudorandom and collision-resistant hash function could be. Besides, we point out that the same approach was chosen by Micciancio et al. in their NIST submission based on SWIFFT.

ZesT has exciting flavors of provable security. Like the Zémor-Tillich hash function, ZesT is collision resistant if and only if the balance problem is hard and in particular if the representation problem is hard for the group $SL(2, \mathbb{F}_{2^n})$ and the generators $A_0 = \left(\begin{smallmatrix} X & 1 \\ 1 & 0 \end{smallmatrix}\right)$ and $A_1 = \left(\begin{smallmatrix} X & X+1 \\ 1 & 1 \end{smallmatrix}\right)$. According to existing attacks, ZesT is provably preimage resistant, second preimage resistant and collision resistant up to $n/2$ bits, for an output of $2n$ bits.

For collision and second preimage resistance, the security of ZesT is indeed of $n/2$ bits in the sense that attacks with this complexity exist and that attacks with lower complexity would improve the resolution of the balance problem. For preimage resistance, the actual security of ZesT seems to be as large as $2n$ bits because the preimage attacks against the Zémor-Tillich hash function do not generalize to ZesT. Besides, an informal reasoning on the known weaknesses of Zémor-Tillich tends to assert the security of ZesT as a MAC, and tests performed with Dieharder [2] assert its pseudorandom behavior.

ZesT provides great recipes for many diets: it is really practical in a wide range of applications. ZesT is provably secure, reasonably efficient in software, about as efficient as SHA in FPGA and very compact in low area ASIC implementations. ZesT algorithm only consists of XORs, SHIFTs and TEST operations on bit vectors. This simplicity allows efficient implementations on a wide range of platforms, as well as software-assisted code optimization.

At equivalent collision resistance security levels, ZesT is 10 times as slow as SHA-1, 10 times as slow as SHA-256 and 4 times as slow as SHA-512 on our 32-bit architecture evaluation platform. With the notable exception of SWIFFT [31], this is comparable or better than other provably secure hash functions, and fast enough for most software applications that are limited in speed by the poor efficiency of their asymmetric cryptographic components anyway. Moreover, the function would take full benefit of graphic accelerators with large data buses and instructions.

ZesT is particularly efficient in hardware. For high speed designs in FPGA, first evaluations show that ZesT implementations may reach throughput per slice ratios comparable to very optimized FPGA implementations of SHA-1 and SHA-2. On the other hand, the simplicity of ZesT allows trading throughput for area with a lot of flexibility. Lightweight implementations of ZesT are much smaller than lightweight implementations of currently used hash functions. In particular, ZesT outperforms the lightweight implementation of SHA-1 presented in [22], the SQUASH implementation of [27] and the implementation of the block cipher-based hash function DM-PRESENT-80 presented in [21].

ZesT can be cut into small pieces. The function has inherited the parallelism of the Zémor-Tillich hash function. Unlike many hash functions proposed for the NIST competition, the computation of ZesT in serial and parallel modes gives the same result thanks to the inherent group structure of the function and in particular to the associativity property. Besides, additional parallelism can be exploited in software by using SIMD instructions for computing the XORs of large bit vectors.

After appropriate preparation, ZesT sweetly fits into NIST's cooking pot. Despite its very interesting properties, ZesT hash function does not completely fulfil standard requirements for hash functions like described in NIST's call [1]. However, slight modifications of the algorithm allow it to reach these requirements.

ZesT as such is a keyed hash function that possesses some weak keys, while a secure unkeyed version is necessary in many applications. This issue is solved by an appropriate choice of default keys. ZesT's collision resistance is only the square root of the birthday bound, and its second preimage resistance is not better than its collision resistance. The first issue is solved by using the projective version of Zémor-Tillich instead of the vectorial version in the second round of ZesT; and the second issue is solved by doubling the parameters' sizes in the first round. Finally, ZesT's parameter n must be prime to be protected against subgroup attacks while NIST required output lengths of 224, 256, 384 and 512 bits. This issue can be solved either by truncating or by extending the outputs by a few bits.

All these changes in ZesT's recipe may influence its simplicity, its efficiency and its security. Like tastes and colors this cannot be discussed. We believe that the choice between one or another version of ZesT's recipe will depend on everybody's personal taste for simplicity, efficiency, security and conformity with NIST's requirements. We present our favorite recipe but we invite everybody to select their personal first-rate ingredients.

This paper is organized as follows. Section 2 recalls the main results on the Zémor-Tillich hash function. Section 3 introduces the ZesT function. Section 4 and 5 give provable and heuristic security results on ZesT. Section 6 gives efficiency results for a software optimized C code, a high-speed FPGA implementation and a low-area ASIC implementation and it furthermore discusses additional implementations. Section 7 modifies the function to approach NIST's requirements, Section 8 discusses some alternative choices in the design's parameters and Section 9 concludes the paper.

2 The Zémor-Tillich hash function

In this paper, we will often abusively identify a polynomial of degree n over $\mathbb{F}_2$ $P_n(X) = X^n + p_{n-1}X^{n-1} + ... + p_1X + p_0$ to its corresponding bit sequence $p_{n-1}...p_1p_0$. When $P_n(X)$ is irreducible, it defines a finite field $\mathbb{F}_{2^n}$ whose elements will be identified to bitstrings through the isomorphism $\mathbb{F}_{2^n} \approx \mathbb{F}_2[X]/(P_n(X))$. Finally, a vector $\left(\begin{smallmatrix} a & b \end{smallmatrix} \right) \in (\mathbb{F}_{2^n})^2$ will be abusively identified to the bit sequence $a||b$.

Let $m = m_0m_1...m_\mu$ be the bitstring representation of a message m. Let $P_n(X)$ be an irreducible polynomial of degree n over $\mathbb{F}_2$ (Tillich and Zémor suggested using $130 \leq n \leq 170$) defining the field $\mathbb{F}_{2^n} \approx \mathbb{F}_2[X]/(P_n(X))$. Let A_0, A_1 be the matrices of $G := SL(2, \mathbb{F}_{2^n})$ (the group of 2×2 matrices over $\mathbb{F}_{2^n}$ with unitary determinant) defined by

$$A_0 = \begin{pmatrix} X & 1 \\ 1 & 0 \end{pmatrix} \qquad A_1 = \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix}$$

The matrices A_0 and A_1 will be called *graph generators*. The Zémor-Tillich hash value of m is defined as the matrix product [42]

$$H_{ZT}(m) = H_{ZT}\left(P_n(X), m\right) := A_{m_0} A_{m_1} ... A_{m_\mu}.$$

As the group $SL(2, \mathbb{F}_{2^n})$ has size $2^n(2^{2n}-1)$, the output size is roughly $3n$ bits if the matrices of $SL(2, \mathbb{F}_{2^n})$ are mapped to bitstrings. This very simple design is also very elegant and appealing, as many properties of the hash function can be captured through graph-theoretical and group-theoretical problems [45,46,42]. Finding collisions is hard if the corresponding *representation problem* is hard to solve, and if and only if the corresponding *balance problem* is hard to solve.

Problem 1 (Representation problem) *Given a group G and a subset S thereof, find a reduced product of subset elements of length at most L that is equal to the unit element of the group, that is*

$$\prod_{0 \leq i < \mu} s_i = 1$$

with $s_i \in S$, $s_i s_{i+1} \neq 1$ and $\mu \leq L$.

Problem 2 (Balance problem) *Given a group G and a subset S thereof, find two reduced products of subset elements of lengths at most L that are equal, that is*

$$\prod_{0 \leq i < \mu} s_i = \prod_{0 \leq i < \mu'} s'_i$$

with $s_i, s'_i \in S$, $s_i s_{i+1}, s'_i s'_{i+1} \neq 1$ and $\mu, \mu' \leq L$.

The parameter L in these problems corresponds to the maximal bit size of message that is considered as "reasonable". We may require L to be polynomial in n for theory or to be smaller than 2^{50} for practice.

The representation and balance problem are not classical problems in Cryptography but in the case of the Zémor-Tillich hash function they have resisted 15 years of cryptanalysis attempts. "Provable security" with respect to non-standard assumptions may not give *today* the same confidence as a security reduction to the factorization problem, but the confidence grows over the years if the function does not get broken. At least, the security reduction for the Zémor-Tillich hash function provides a concise well-defined mathematical problem to solve in order to break the function, and this problem has not been solved for 15 years.

The hardness of representation and balance problems seems to depend a lot on the group G and the subset S. Representation problems have been long studied in Spectral Graph Theory of Cayley graphs [28]; balance problems have been introduced in [9] for Abelian groups and factorization problems are well-studied in other settings. Finding *the shortest* representation and factorization are hard problems for generic groups [29,20]. The representation problem is as hard as the discrete logarithm problem in Abelian groups [9], but for Abelian groups the balance problem is clearly easy. The representation problem was solved for the LPS and Morgenstern hash functions [43,35] but these functions use very particular generator sets that give rise to Ramanujan graphs [11,30,33]. In general, the hardness of representation and balance problems is a very interesting open problem for Cryptography.

Despite partial cryptanalysis results, the Zémor-Tillich hash function remains essentially unbroken today, 15 years after its publication. The message $0^{\mathrm{ord}(A_0)}$ and any similarly constructed message provide solutions to the representation problems, but for parameter L far too large for practice, unless the polynomial is chosen explicitly to make these attacks practical [12,7]. Similarly, the attack described by Geiselmann in [24] produces collisions that are far too large for practice. Steinwandt et al. [41] exploited the subgroup structure of $SL(2, \mathbb{F}_{2^n})$ to build trapdoor attacks (that find collisions by "cheating" in the choice of the polynomial $P_n(X)$) as well as collision attacks but only for composite parameters n.

Recently, Petit et al. [36] presented both collision and preimage attacks running in time $2^{n/2}$ instead of the optimal $2^{3n/2}$ and 2^{3n} birthday and exhaustive search bounds. Those attacks are generic in the sense that they work for any parameters. They perform "meet-in-the-middle" attacks in a space of size 2^n to find messages m_i hashing to matrices with a left eigenvector equal to $(1, 0)$, and then combine these matrices efficiently to solve the representation problem. The attack gives collisions to the void message that are concatenations of many individual messages m_i.

Besides existing partial attacks on the collision and preimage resistance properties, the Zémor-Tillich hash function has issues that impede its use as a general-purpose hash function. First, the function can be inverted on messages of size up to $n + 80$ as no polynomial reduction appears in the hash computation for message sizes smaller than n [41]. This may become a problem for example if ECDLP keys are hashed. Second, the hash function is clearly *malleable*, in the sense that for any $m_1, m_2 \in \{0,1\}^*$, $H_{ZT}(m_1||m_2) = H_{ZT}(m_1)H_{ZT}(m_2)$.

As a counterpart of their attacks, Petit et al. [36] proposed two variants of the Zémor-Tillich hash function with reduced output sizes but essentially the same collision and preimage resistance security. Both variants are parameterized by a polynomial $P_n(X)$ plus an *initial vector* $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) \neq \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix}\right)$. The *vectorial Zémor-Tillich* is defined by $H^{vec}_{ZT}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m\right) := \left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right) H_{ZT}\left(P_n(X), m\right)$. The *projective Zémor-Tillich* returns the projective point[1] corresponding to the output of the vectorial variant, that is $H^{proj}_{ZT}(P_n(X)||(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}), m) = [a : b]$ if $H^{vec}_{ZT}(P_n(X)||(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}), m) = (\begin{smallmatrix} a & b \end{smallmatrix})$. When the initial vector is randomly chosen, the collision resistance of the Zémor-Tillich hash function is equivalent to the collision resistance of its vectorial variant, and equivalent to its projective variant for sufficiently small parameters [36].

In the remainder of this paper, we work with prime parameter n hence the subgroup attacks of Steinwandt et al. do not apply. Moreover, we *assume* that the best attack against the Zémor-Tillich hash function and its variants is the attack described in [36] that require time $2^{n/2}$ and produce structured messages (except for the projective variant).

The vectorial version was used in [37] to construct a hash function based on Zémor-Tillich but without its main weaknesses that are malleability and invertibility for short messages. The basic idea was to iterate the vectorial Zémor-Tillich in order to remove the malleability properties but to keep the provable security and the parallelism of Zémor-Tillich. The function was evaluated for software applications by Petit et al. [37] and for FPGA and area-constrained applications by de Meulenaer et al. [19].

The ZesT function proposed in this paper builds upon these works. We provide the full theoretical justifications that are missing in [37] and we slightly modify the function to achieve these proofs, decrease its area requirements and simplify the function. We provide efficiency estimates based on a new implementation in C and on the implementations of de Meulenaer et al. [19] in hardware. Additionally, we study the function as a MAC and we show how to further modify it in order to reach NIST's requirements [1].

We start by describing ZesT's hash and key generation algorithms.

3 ZesT hash function

3.1 ZesT hash algorithm

ZesT hash algorithm takes as entry a key made of an irreducible binary polynomial $P_n(X)$ and of a starting point $(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}) \in \mathbb{F}^2_{2^n} \setminus (\begin{smallmatrix} 0 & 0 \end{smallmatrix})$, and a bitstring $m = m_0 m_1 ... m_{\mu-1}$ of arbitrary length. It is defined by

$$\textrm{ZesT}(P_n(X)||(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}), m) := H^{vec}_{ZT}(P_n(X)||(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}), (m||H^{vec}_{ZT}(P_n(X)||(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}), m))) .$$

ZesT algorithm is made of two rounds of the vectorial Zémor-Tillich hash function: after the first round, the intermediary result $(\begin{smallmatrix} a & b \end{smallmatrix}) := H^{vec}_{ZT}(P_n(X)||(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}), m)$ is seen as a bit sequence of $2n$ bits that are processed as a continuation of the message bits.

[1] We denote by $\mathbb{P}^1(\mathbb{F}_{2^n})$ the projective space of dimension 1 on $\mathbb{F}_{2^n}$, which is the set of equivalence classes of $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ that results from identifying two vectors $(\begin{smallmatrix} a_1 & b_1 \end{smallmatrix})$ and $(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix})$ if and only if $(\begin{smallmatrix} a_2 & b_2 \end{smallmatrix}) = \lambda (\begin{smallmatrix} a_1 & b_1 \end{smallmatrix})$ for some $\lambda \in \mathbb{F}^*_{2^n}$. We denote by $[a : b]$ the projective point that is the equivalence class of a vector $(\begin{smallmatrix} a & b \end{smallmatrix})$.

3.2 ZesT key generation algorithm

The key of ZESᴛ is made of an irreducible polynomial $P_n(X)$ and of a vector $(\, a_0 \; b_0 \,) \in \mathbb{F}_{2^n}^2 \setminus (\, 0 \; 0 \,)$. Both elements are randomly chosen by the key generation algorithm: if the polynomial is fixed, collision resistance cannot be reached: an adversary can simply store a collision for the original Zémor-Tillich hash function to produce collisions for any starting point $(\, a_0 \; b_0 \,)$. On the other hand, if the starting vector is not chosen randomly, the collision resistance of ZESᴛ is no longer equivalent to the collision resistance of the Zémor-Tillich hash function. In particular, some keys are weaker than others, for example if the starting point is $(\, a \; aX \,)$ for any $a \in \mathbb{F}_{2^n}^*$.

If the person who generates the key is trusted, the degree of the polynomial must be prime in order to avoid subgroup attacks against the Zémor-Tillich hash function, and the starting vector must be chosen randomly among all possible vectors for the equivalence between the original and vectorial Zémor-Tillich to hold. If the person generating the key is not trusted, it is necessary to choose the polynomial $P_n(X)$ and the initial vector in a way that clearly discards trapdoor attacks. This protection can be achieved by standard techniques, resorting either to universal constants like π or e, to a pseudorandom number generator or to a cryptographic hash function H (which can even be ZESᴛ with some fixed key).

4 Security reduction for ZesT

ZESᴛ has exciting flavors of provable security. Its collision, preimage and second preimage resistances follow from the hardness of the balance problem corresponding to the Zémor-Tillich hash function.

4.1 Collision resistance

ZESᴛ is collision resistant if and only if the balance problem corresponding to the Zémor-Tillich hash function is a hard problem.

Proposition 1 *There exists a* PPT *algorithm that breaks the collision resistance of* ZESᴛ *if and only if there exists a* PPT *algorithm that solves the balance problem corresponding to the Zémor-Tillich hash function.*

PROOF: We show how to construct a collision for the vectorial Zémor-Tillich with key $P_n(X)||(\, a_0 \; b_0 \,)$ from a collision for ZESᴛ with the same parameters and vice-versa; the result then follows from the equivalence between the vectorial and original versions of Zémor-Tillich (Proposition 4 of [36]). Let (m, m') be a collision on ZESᴛ: we have $m \neq m'$ and

$$H_{ZT}^{vec}\left(P_n(X)||(\, a_0 \; b_0 \,), (m||H_{ZT}^{vec}\left(P_n(X)||(\, a_0 \; b_0 \,), m\right))\right)$$
$$= H_{ZT}^{vec}\left(P_n(X)||(\, a_0 \; b_0 \,), (m'||H_{ZT}^{vec}\left(P_n(X)||(\, a_0 \; b_0 \,), m'\right))\right).$$

The messages $m||H_{ZT}^{vec}\left(P_n(X)||(\, a_0 \; b_0 \,), m\right)$ and $m'||H_{ZT}^{vec}\left(P_n(X)||(\, a_0 \; b_0 \,), m'\right)$ collide for the vectorial version and are distinct. On the other hand, it is clear that any collision on the vectorial version is also a collision on ZesT. □

The equivalence result of Proposition 1 is nearly tight. On one side, a solution to the balance problem immediately gives a collision on ZesT. On the other side, $\log_2 n$ bits of security are "lost" from the vectorial to the matrix version of Zémor-Tillich in the proof of Proposition 4 of [36]. The collision resistance of ZesT is not optimal as its output has $2n$ bits while the collision attacks of [36] will find collisions for the vectorial version of Zémor-Tillich in time $2^{n/2}$. In Section 7.3, we will suggest a modification of ZesT that reaches optimal collision resistance.

4.2 Preimage resistance up to the collision resistance level

The preimage resistance of ZesT follows from the hardness of the balance problem corresponding to the Zémor-Tillich hash function. We give here a proof that provides a preimage resistance guarantee but only up the $n/2$ bits of the collision resistance level.

Proposition 2 *If there exists a PPT algorithm that breaks the preimage resistance of* ZesT*, then there exists a PPT algorithm that solves the balance problem corresponding to the Zémor-Tillich hash function.*

Proof: The result is immediate as ZesT processes arbitrary-length bit sequences and it is collision resistant if the balance problem is hard (see [38] and Proposition 1). The informal argument is as follows. Suppose there exists an efficient algorithm A computing preimages, then there exists an efficient algorithm B that finds collisions: B chooses a random message m, computes its hash value, gives the hash to A and receives m' from A. As each hash value has a lot of preimages on average, the messages m and m' are likely to be different hence to form a collision. □

The result is not tight as there does not currently exists any algorithm able to compute preimages for ZesT in time $2^{n/2}$. Indeed, we argue in Section 5 that the actual preimage resistance level of ZesT seems closer to $2n$ bits.

4.3 Second preimage resistance up to the collision resistance level

The second preimage resistance of ZesT also follows from the hardness of the balance problem corresponding to the Zémor-Tillich hash function.

Proposition 3 *If there exists a PPT algorithm that breaks the second preimage resistance of* ZesT*, then there exists a PPT algorithm that solves the balance problem corresponding to the Zémor-Tillich hash function.*

Proof: Identical to the proof of Proposition 2. □

This result is tight as there exists an algorithm computing second preimages in time $2^{n/2}$. Indeed, given a message m, there exists an algorithm computing a preimage of $(\,a\ b\,) := H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right)$ in time $2^{n/2}$: this algorithm first computes a matrix $M \in SL(2, \mathbb{F}_{2^n})$ such that $(\,a_0\ b_0\,)\,M = (\,a\ b\,)$ and it then applies the preimage algorithm of [36] to M. To compute a second preimage of ZesT, it suffices to give $(\,a\ b\,)$ to this algorithm; as ZesT processes arbitrary-length inputs the message m' returned is likely to be different from m. Moreover,

$$\begin{aligned}
&H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, (m||H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right))\right)\\
&\qquad\qquad = H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right) H_{ZT}\left(P_n(X), H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right)\right)\\
&\qquad\qquad = H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m'\right) H_{ZT}\left(P_n(X), H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m'\right)\right)\\
&\qquad\qquad = H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, (m'||H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m'\right))\right).
\end{aligned}$$

5 Other security aspects of ZesT

ZesT is Zémor-Tillich with Enhanced Security inside. In this section, we give security properties of ZesT that cannot be proved based on the hardness of the balance problem corresponding to Zémor-Tillich, but that still appear very likely.

5.1 Output distribution

We argue that for long messages, the output distribution of ZesT is close to uniform. Given a message m, ZesT outputs the value

$$\begin{aligned}
&H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, (m||H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right))\right)\\
&\qquad\qquad = H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right) H_{ZT}\left(P_n(X), H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right)\right)
\end{aligned}$$

which can be seen as the result of two consecutive walks determined by the bits of m and of $(\,a\ b\,) := H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right)$ in a graph $\mathcal{ZT}^{vec}$ corresponding to the vectorial version[2]. This graph is a quotient graph of the graph $\mathcal{ZT}$ corresponding to the original Zémor-Tillich hash function. As random walks converge in $\mathcal{ZT}$ [42], they also converge in $\mathcal{ZT}^{vec}$ [28], and the uniform distribution of $(\,a\ b\,)$ for long messages follows. The second walk of $2n$ bits performed from $(\,a\ b\,)$ should not affect this distribution because the bits of $H_{ZT}^{vec}\left(P_n(X)||\,(\,a_0\ b_0\,)\,, m\right)$ are expected to be reasonably random and independent of those of m. Under this independence assumption, the output distribution of ZesT on long messages is provably close to the uniform distribution.

5.2 Preimage resistance

The preimage resistance of ZesT is much better than the $n/2$ bit security that can be proved based on the hardness of the representation problem. As preimages of the first round can be computed in time $2^{n/2}$, we may try to fix the value h' after this first round and to recover the message with this additional constraint. However, finding an h' value that may satisfy the equation

[2] See [36] for the definition of this graph.

$h = h' H_{ZT}(P_n(X), h')$ when h is given seems to be a hard problem, that cannot be solved faster than by exhaustive search methods.

A preimage on ZesT implies a preimage on H_{ZT}^{vec} in the second round that has the form $m||H_{ZT}^{vec}(P_n(X)||(a_0\ b_0), m)$. The preimage algorithm of Section [36] can be easily modified to compute some kinds of particular preimages on the vectorial version. For example, there exists an algorithm finding preimages that start and end with some given constant bitstrings. However, computing preimages of the form $m||H_{ZT}^{vec}(P_n(X)||(a_0\ b_0), m)$ faster than by exhaustive search seems to be out of reach.

The preimage attacks of [36] cannot be extended to messages of the form $m||H_{ZT}^{vec}(P_n(X)||(a_0\ b_0), m)$. For generic messages, we could concatenate various messages colliding for the projective version into a collision for the vectorial version. The approach does not work here because the concatenation of two messages of the form $m||H_{ZT}^{vec}(P_n(X)||(a_0\ b_0), m)$ is not a message of the form $m||H_{ZT}^{vec}(P_n(X)||(a_0\ b_0), m)$ in general.

For generic messages, we could also follow a "meet-in-the-middle" strategy to get preimages at the price of collisions. In the second round of ZesT, this approach is no longer possible because of the redundancy between the left-most and the right-most bits of the message that is given to the second round. More generally, it seems impossible to exploit the mathematical structure of ZesT (in particular the associativity of the matrix product) to improve generic preimage attacks against the second round because of the redundancy introduced between the bits of m and those of $H_{ZT}^{vec}(P_n(X)||(a_0\ b_0), m)$.

The actual preimage resistance of ZesT is therefore of $2n$ bits, hence much better than the $n/2$ bits security obtained from the hardness of the balance problem. For applications that only require 60 bits of preimage resistance (without requiring collision resistance), parameters as small as $n = 31$ will therefore be safe.

5.3 Issues in Zémor-Tillich that are removed in ZesT

ZesT does not present any apparent malleability property nor any apparent predictable behavior contradicting the intuition of pseudorandomness.

Unlike Zémor-Tillich and its vectorial variant, ZesT cannot be inverted on short messages. In these functions, the invertibility comes from the absence of modular reductions when the message size is only slightly larger than n. The issue is removed in ZesT because at least $2n$ bits are hashed in the second round.

ZesT is not malleable. As an example, let us consider a simple malleability issue of H_{ZT}^{vec} that is a relation between the hash values of m and $m' = m||0$:

$$\text{if } H_{ZT}^{vec}(m) = h_1||h_2 \text{ then } H_{ZT}^{vec}(m') = (h_1 X + h_2)||h_1.$$

We point out that the malleability of the vectorial Zémor-Tillich is limited to addition and/or suppression of bits on the right side of unknown messages. In particular, it is not possible to modify a hash value according to a change in the middle bits of an unknown message.

Let us now consider $\text{Z}\textsc{es}\text{T}(m) = H_{ZT}^{vec}(m||H_{ZT}^{vec}(m))$. Although they are strongly correlated, the hash values $H_{ZT}^{vec}(m)$ and $H_{ZT}^{vec}(m')$ differ in many middle bits in general, so $\text{Z}\textsc{es}\text{T}(m)$ and $\text{Z}\textsc{es}\text{T}(m')$ are completely uncorrelated. Of course, there exist some particular values (m, m') such that $H_{ZT}^{vec}(m)$ and $H_{ZT}^{vec}(m')$ are very close, for example differ by only the last bit, but finding such a pair without inverting H_{ZT}^{vec} already seems a hard problem. Moreover, any such m and m' that we could find would differ in many bits, so again $\text{Z}\textsc{es}\text{T}(m)$ and $\text{Z}\textsc{es}\text{T}(m')$ would be completely uncorrelated.

In Section 7.2 below, we provide further evidence that ZesT has no apparent weakness, based on analysis carried out with the pseudorandom tests of the Dieharder [2].

5.4 Security as a MAC

ZesT can be used as a message authentication code $\mathcal{M}\text{Z}\textsc{es}\text{T} = (Gen, Mac, Ver)$. The *Gen* and *Mac* algorithms of $\mathcal{M}$ZesT are just the key generation and the hash algorithms of ZesT. Of course, the key remains secret here, and it is important that both elements $P_n(X)$ and $\left(a_0 \; b_0 \right)$ remain secret. On input (s, m, t), the verification algorithm simply checks whether $t = \text{Z}\textsc{es}\text{T}(s, m)$.
$\mathcal{M}$ZesT is essentially HMAC used with the vectorial version of the Zémor-Tillich hash function. Although the weaknesses of this last function are not present in the functions usually employed with HMAC, we argue that $\mathcal{M}$ZesT is a secure MAC algorithm.

Key recovery against $\mathcal{M}$ZesT seems to be a hard problem. A ZesT key is made of two components, an irreducible polynomial $P_n(X)$ and an initial vector $\left(a_0 \; b_0 \right)$. We argue that recovering the whole key has a cost 2^{2n} even if the polynomial $P_n(X)$ can be recovered in time 2^n.

Let us first suppose that the polynomial $P_n(X)$ is not known to the adversary. If the adversary knew the initial vector $\left(a_0 \; b_0 \right)$, he could easily recover $P_n(X)$ as follows. The adversary would send the void message to the *Mac* algorithm and receive an answer that equals $H_{ZT}^{vec}(P_n(X)||\left(a_0 \; b_0 \right), \left(a_0 \; b_0 \right))$. The adversary could also compute the hash value by itself without performing the reductions and subtract the hash value returned by the *Mac* algorithm to obtain a vector $\left(a \; b \right) \in \mathbb{F}_2[X]$. The polynomial $P_n(X)$ would be an irreducible factor of $\gcd(a, b)$ with degree n. If needed, the adversary would make an additional query to the *Mac* algorithm to discriminate between alternative possible factors of degree n.

If the adversary does not know the initial vector, he can still recover the polynomial $P_n(X)$ with 2^n $\mathcal{M}$ZesT queries as follows. After 2^n queries, the adversary is likely to find two messages with the same MAC value. With a probability of about 50%, these two messages provide a collision (m, m') on the vectorial version of Zémor-Tillich. The adversary then computes the Zémor-Tillich hash values M, M' of m and m' without performing the modular reductions. The polynomial $P_n(X)$ is an irreducible factor of degree n of $\det(M + M')$. As this determinant may have more than one polynomial factor of degree n and as the adversary does not know whether the collision he obtained for the MAC was a collision for the first round, he needs a few MAC collisions to identify the right polynomial.

Let us now suppose that the polynomial $P_n(X)$ is known to the adversary who wants to recover the initial vector. From messages m_i of his choice and the corresponding hash values $h_i := \text{ZesT}(P_n(X) || \begin{pmatrix} a_0 & b_0 \end{pmatrix}, m_i)$, the adversary tries to recover $\begin{pmatrix} a_0 & b_0 \end{pmatrix}$. This is equivalent to finding any $\begin{pmatrix} a_i & b_i \end{pmatrix}$ such that $\begin{pmatrix} a_i & b_i \end{pmatrix} = H_{ZT}^{vec}(P_n(X) || \begin{pmatrix} a_0 & b_0 \end{pmatrix}, m_i)$ because $\begin{pmatrix} a_0 & b_0 \end{pmatrix} = \begin{pmatrix} a_i & b_i \end{pmatrix} (H_{ZT}(P_n(X), m_i))^{-1}$. The task of the adversary now consists in solving the equation

$$\begin{pmatrix} a_i & b_i \end{pmatrix} H_{ZT}(P_n(X), \begin{pmatrix} a_i & b_i \end{pmatrix}) = h_i$$

for one of the h_i. Solving this equation seems hard because of the redundancy in the unknown; the preimage attack of Section [36] does not extend to this case. We believe that when the polynomial is known, the adversary cannot recover the initial vector faster than in time 2^{2n}.

Message-extension attacks against $\mathcal{M}$ZesT are defeated by the second round of ZesT. These attacks are possible for any iterative hash function, in particular all Merkle-Damgård-based hash function and all expander hashes. The attack is prevented in ZesT in exactly the same way as in the HMAC construction: the second round destroys the block structure and prevents the adversary from having access to the result of an iterative hash function.

Forging $\mathcal{M}$ZesT seems to require 2^n MAC queries plus a computation time $2^{n/2}$. As soon as the polynomial $P_n(X)$ is known by the adversary, a forgery for this MAC is feasible in time $2^{n/2}$: the forger can compute a collision (m, m') for the Zémor-Tillich hash function, query the Mac algorithm on m to receive t, and return (m', t) as a valid forgery. When the polynomial is not known, the adversary cannot compute collisions for the Zémor-Tillich hash function. Moreover, its oracle access to $\mathcal{M}$ZesT does not help it to attack the Zémor-Tillich hash function as he only accesses the output of the second round. The malleability of the first round is not useful either to the adversary for the same reason. We have found no forgery algorithm faster than our best partial key recovery algorithm on $P_n(X)$ followed by a collision attack on the Zémor-Tillich hash function.

The trapdoor attacks and weak keys issues present in ZesT do also affect $\mathcal{M}$ZesT. However, when the key generation algorithm is not trusted, the techniques sketched out in Section 3.2 to protect ZesT will also protect $\mathcal{M}$ZesT.

5.5 Connections with HMAC and other iterative designs

The design of ZesT is inspired by HMAC [8,23] and by traditional block cipher and compression function designs: the mathematical structure remaining after the first round of ZesT is destroyed in its second round. However, most existing security results on HMAC assume hypotheses on the hash function that are clearly not satisfied by the vectorial Zémor-Tillich hash function, and block ciphers and compression functions usually have much more than just two rounds. The collision resistance of ZesT is guaranteed with a single round; a second round is necessary to obtain "extra" pseudorandom properties; the second round is also sufficient because the round function is already very strong.

ZesT looks very similar to NMAC and HMAC. The collision resistance transfers from the vectorial Zémor-Tillich hash function to ZesT, exactly like in these constructions. On the other hand, existing security results on the pseudorandom and MAC security of HMAC and NMAC cannot be used for ZesT. NMAC and HMAC were built for iterative hash functions whose compression functions have no apparent weaknesses. In contrast, the vectorial version of Zémor-Tillich is highly malleable and any "compression function" we could define from it by fixing a block size would also be malleable. In particular, this compression function would definitely not be pseudorandom nor a secure fixed-length MAC. Unlike its collision resistance, the pseudorandomness of ZesT follows from the iteration and not from the single rounds.

The iterative design of ZesT also appears in many block ciphers and in compression functions, typically with 16 to 64 rounds. In contrast, ZesT only has 2 rounds. Block ciphers or traditional hash functions would become invertible if their round number was decreased. In contrast, a single round of ZesT is already preimage and collision resistant because it is a whole hash function and it is therefore much stronger than the simple components used in block ciphers and compression functions.

6 Efficiency estimations

ZesT provides great recipes for many diets: it is really practical in a wide range of applications. We describe software implementations in Section 6.1, FPGA implementations in Section 6.2 and lightweight implementations in Section 6.3. Finally, we show in Section 6.4 how to exploit in ZesT the inherent parallelism of the Zémor-Tillich hash function.

For FPGA and lightweight implementations, we will base our performance estimations on the implementations in [19] of the function introduced in [37]. This function, that we will call ZT' in this paper, is very similar to ZesT. The only difference is the introduction of an XOR by a constant between the first and the second round:

$$\begin{aligned} &ZT'\left(P_n(X)||\begin{pmatrix} a_0 & b_0 \end{pmatrix}, m\right) \\ &\quad := H_{ZT}^{vec}\left(P_n(X)||\begin{pmatrix} a_0 & b_0 \end{pmatrix}, \left(m||H_{ZT}^{vec}\left(P_n(X)||\begin{pmatrix} a_0 & b_0 \end{pmatrix}, m\right) \oplus c\right)\right). \end{aligned} \tag{1}$$

The constant c is equal to the binary representation of pi in [37].

6.1 Efficiency of ZesT in software

ZesT recursively uses a very simple operation on a state $\begin{pmatrix} a & b \end{pmatrix}$. Depending on the next message bit, the state is updated to $\begin{pmatrix} a & b \end{pmatrix} A_0 = \begin{pmatrix} aX+b & a \end{pmatrix}$ or to $\begin{pmatrix} a & b \end{pmatrix} A_1 = \begin{pmatrix} aX+b & aX+b+a \end{pmatrix} = \begin{pmatrix} aX+b & (aX+b)+a \end{pmatrix}$. After processing all the message bits, the result is seen as a bitstring and processed in turn. Messages of μ bits therefore require to process $\mu + 2n$ bits for the first and the second rounds together.

The arithmetic is in a field of characteristic 2 and is thus very efficient. Like in [37], we may analyze the running time as follows. Any element $a = a_{n-1}X^{n-1}+a_{n-2}X^{n-2}+...a_1X+a_0$ is represented by an array of $L := \lceil \frac{n}{32} \rceil$ 32-bit integers ($L := \lceil \frac{n}{64} \rceil$ in 64-bit architectures). An addition requires only L XORs, and a multiplication a by X requires L SHIFTs by one bit and one polynomial modular reduction. The operations $aX + b$ and $aX + a$ can be performed with L SXORs and a modular reduction. The polynomial reduction in the computation of $aX + b$ can be done by testing the left-most bit of a: if this bit is equal to 1, we need L XORs operations of the bits of $aX + b$ with the bits of $P_n(X)$. For long messages, the TEST instruction will return 1 half of the times.

Let t_0 and t_1 be the average times needed respectively to process a bit 0 and 1, let t_{XOR} and t_{SXOR} be the times needed to perform a word XOR and SXOR, and let t_{TEST} be the time needed to perform a TEST instruction. According to our analysis, t_0 and t_1 are respectively

$$t_0 = Lt_{SXOR} + \frac{L}{2}t_{XOR} + t_{TEST} + C_0,$$
$$t_1 = Lt_{SXOR} + \frac{3L}{2}t_{XOR} + t_{TEST} + C_1.$$

where C_0 and C_1 are constant overhead times. If we neglect the TEST instruction and the overhead times and if we approximate $t_{SXOR} \approx t_{XOR}$, processing one bit requires on average $2Lt_{XOR}$ instructions. The total time needed to evaluate the ZesT hash value of a message of length μ is therefore

$$2(\mu + 2n)Lt_{XOR}.$$

For long messages, this time is essentially proportional to the message length and inversely proportional to the architecture size.

ZesT can be cut into small pieces; it is very scalable to any granularity. Implementing ZesT on an 8-bit processor or on the other hand on a graphical accelerator with a 512-bit data bus is just as easy as implementing ZesT on standard 32 or 64-bit processors, and the implementation speed will be directly proportional to the architecture size. If the architecture is larger than n, ZesT only requires two XORs per message bit, plus $4n$ XORs for the second round.

ZesT algorithm was implemented in C to get running time estimations for various parameters sizes. All tests were performed on a 64-bit Intel Xeon E5420 2.5GHz 16Go DDR2 Ram. The OS was Debian using 32-bit kernel 2.6.26. Test vectors for performance evaluation were 500Mo random files generated using `/dev/urandom`. The values chosen for the parameter n were the smallest primes smaller than 32, 64, 128, 160, 224, 256, 384, 512 and 1024, and for each n a random polynomial of degree n was selected for $P_n(X)$.

Performance results are presented in Table 1. The results differ significantly from the above analysis. First, the analysis seems to become valid only for large values of the parameter n. This may be due to various overheads, in particular to *for* loops present in the code for scalability reasons that should better be unrolled for short parameters. Second, we observe that ZesT-61 and ZesT-251 are respectively more efficient than ZesT-31 and ZesT-157: this might be due to the subjacent 64-bit architecture of our evaluation platform and to scalability options taken in our code.

In Table 2, the performances of ZesT are compared to the SHA algorithm evaluated with the `sha1sum`, `sha256sum` and `sha512sum` functions of the linux kernel. At comparable collision resistances, ZesT-127, ZesT-251 and ZesT-509 are respectively 10, 10 and 4 times less efficient than SHA-1, SHA-256 and SHA-512. At comparable preimage resistances (which is $2n$ bits for ZesT), ZesT-127 and ZesT-251 are respectively 7 and 2 times less efficient than SHA-256 and SHA-512. If we only rely on the $n/2$ "provable" bits of preimage resistance, then at comparable preimage resistances, ZesT-509 and ZesT-1021 are respectively 17 and 7 times less efficient than SHA-256 and SHA-512.

ZesT	Time (s)
ZesT-31	43
ZesT-61	33
ZesT-127	43
ZesT-157	65
ZesT-223	76

ZesT	Time (s)
ZesT-251	62
ZesT-383	82
ZesT-509	103
ZesT-1021	183

Table 1. Estimated running time (seconds) of ZesT with parameters of various sizes in a 32-bit architecture

CR	SHA	Time (s)	ZesT	Time(s)
$\approx 2^{64}$	SHA-1	4	ZesT-127	43
$\approx 2^{128}$	SHA-256	6	ZesT-251	62
$\approx 2^{256}$	SHA-512	27	ZesT-509	103

Table 2. Comparison of SHA and ZesT at the same collision resistance levels

The basic algorithm described above can be improved by grouping the computation of 2 or 4 consecutive message bits. This was done in [37] with the help of a code-generation program. The ZEST algorithm is so simple that code-generation programs may be easily tuned to any computer architecture and may also include more elaborate grouping strategies, for example based on Huffman coding. Finally, we point out that the performances of ZEST will be greatly improved by using the multimedia sets of instructions available on modern processors.

6.2 FPGA implementation

In this section, we provide performance estimations of high-speed implementations of ZEST on FPGA. These implementations are of interest in applications where several messages are to be hashed and a high throughput is required, such as for virtual network servers. The throughput per area metric makes sense here since the goal is to jointly minimize the execution time and the area: if the throughput of the resulting implementation is too low, any superior throughput can be reached by simply gathering several identical circuits.

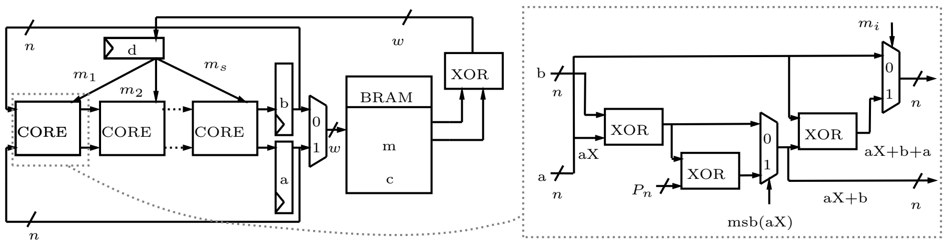

Fig. 1. FPGA architecture for ZT' proposed in [19]

Figure 1 presents the architecture proposed in [19] for ZT' which is essentially ZEST with an additional XOR between the first and the second round (see Equation 1). It is made of a central core processing one bit of message, and of storage elements. The core can be replicated s times in order to process s consecutive message bits. The throughput achieved can be approximated by the product of the frequency and s if the processing of the final phase is negligible, which is the case for long messages.

The implementation results for ZT' provide fair estimates of the performances of ZEST. Due to the absence of the constant c in ZEST, the usage of BRAM and of control logics will slightly decrease. The frequency should be on the same order as the longest data path remains unchanged. FPGA implementations of ZEST will therefore have slightly better but comparable throughput/area ratio as ZT'.

	Collision Resistance	Area [Slices]	Frequ. [Mhz]	Through. [Mbps]	Through./Area [Mbps / Slice]
SHA-1 [15]	2^{63}	533	230	1435	2.7
ZT'-127	2^{64}	597	160	800	1.34
SHA-256 [14]	2^{128}	797	150	1184	1.49
ZT'-251	2^{126}	1044	140	700	0.67
SHA-512 [14]	2^{256}	1666	121	1534	0.92
ZT'-509	2^{255}	1850	135	675	0.36

Table 3. Comparison of the implementation results of ZT' with SHA.

The best performance results of [19] are presented in Table 3 together with the best results for SHA. These results were obtained for $s = 5$. The impact of n on the frequency is moderate as increasing n does not add logic operators to the longest path. The small frequency drop is likely due to larger routing delays. The area may be approximated by a linear function of n:

$$\text{Area} = 3.3n + 200.$$

The area is nearly proportional to n as the only constant parts of the circuits are the control logics and the BRAM.

The results show that ZT' and hence ZesT have comparable performances with respect to the SHA hash function in terms of throughput per slice. Table 3 provides a comparison of the efficiency and collision resistance of ZT' with SHA. At comparable level of collision resistance, ZT'-127, ZT'-251 and ZT'-509 are about twice less efficient than the state-of-the-art implementations of SHA-1, SHA-256 and SHA-512 respectively. These performances might still be improved by 50% by introducing pipeline stages between the s cores [19].

6.3 Lightweight implementations

We now study the performances of ZesT in constrained environments such as for RFID tags authentication. Like in the previous section, we give estimations based on the lightweight implementations of ZT' that we proposed in [19]. The FPGA architecture with $s = 1$ can be modified to obtain the lightweight implementation of Figure 2. The first main change introduced consists in computing the entries a and b one bit at the time instead of all bits in parallel in order to save area by replacing n-bit gates by 1-bit ones. The second main change involves the storage elements. In lightweight implementations, large blocks of memory like BRAM are no longer available and are therefore replaced by two registers (labeled $a + c$

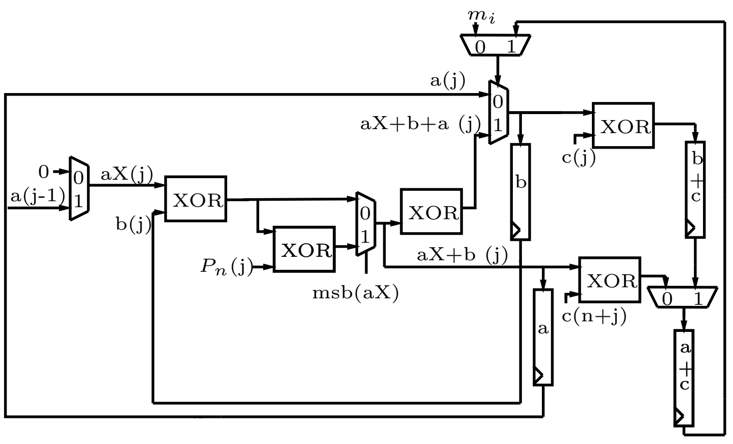

Fig. 2. Lightweight architecture for ZT' proposed in [19]

and $b + c$) that store the result of the XOR operation between the intermediary result and the constant c, which is hardcoded.

ZesT is very efficient in terms of occupied area with respect to current hash functions. Table 4 summarizes the results concerning ZT' and other hash functions [19]. shows that both the lightweight and high-speed (with $s = 1$) versions of ZT'-127 already outperform the hash functions SHA-1 and MD5. Lightweight ZT'-127 is a little smaller than the state of the art implementation of the AES block cipher proposed in [21]. The area requirements and collision resistances of ZT' and SHA are compared in Table 5, illustrating the inferior area costs for ZT' at a comparable collision resistance. ZT'-127 requires roughly one third of the area of SHA-1 while ZT'-251 needs half of the area of SHA-256. ZT'-127 is a little less compact than H-PRESENT-128, the hash function recently proposed in [10] based on the block cipher PRESENT.

ZesT-127 is comparable to DM-PRESENT-80 and it outperforms even H-PRESENT-128 for the same collision resistance. Based on our results for n=127 and n=251, the area required for the lightweight ZT' may be approximated by the function Area $= 20n + 300$. The area for ZesT can therefore be roughly approximated by Area $= 10n + 300$ as half of the registers are removed. This leads to approximations of 1600 and 2900 gates equivalents for ZesT-127 and ZesT-251 respectively.

For some applications, collision resistance is not required and a moderate level of security is sufficient (60-bit or 80-bit security) [39]: for example, many RFID protocols only rely on preimage resistance. ZesT turns out to be a very interesting candidate for these applications. As explained in Section 5.2, the

	Output size	Through. at 100kHz [kbps]	Through/Area [bps/GE]	Logic process	Area [GE]
MD5 [22]	128	83.7	10	0.13μm	8400
SHA-1 [22]	160	40.2	4.9	0.35μm	8120
SHA-256 [22]	256	45.4	4.2	0.35μm	10868
SQUASH [27]	32	< 0.1	< 0.02	estimate	<6000
AES-128 [21]	128	12.4	3.7	0.35μm	3400
DM-PRESENT-80 [10]	64	14.6	9.1	0.18μm	1600
H-PRESENT-128 [10]	128	11.5	4.9	0.18μm	2330
ZT'-127 (lightweight)	254	0.52	0.18	65nm	2945
ZT'-251 (lightweight)	502	0.20	0.04	65nm	5517
ZT'-127 (s = 1)	254	66.7	17.8	65nm	3752
ZT'-251 (s = 1)	502	66.7	9.2	65nm	7267

Table 4. Comparison of the performances of the lightweight implementation of ZT' with other hash functions and the AES block cipher [19]

	Collision Resistance	Area [GE] (rel.)
SHA-1 [21]	2^{63}	8120 (1)
ZT'-127 (lightweight)	2^{64}	2945 (0.36)
ZesT-127 (approximation)	2^{64}	1600 (0.20)
SHA-256 [21]	2^{128}	10868 (1)
ZT'-251 (lightweight)	2^{126}	5517 (0.51)
ZesT-251 (approximation)	2^{126}	2900 (0.27)

Table 5. Comparison of the collision resistance and area cost of SHA with the lightweight implementation of ZT' and the approximation for ZesT [19].

preimage resistance of ZesT-n is at least $2^{n/2}$ based on current knowledge on the balance problem, but it actually seems to be 2^{2n}.

In Table 6, the lightweight implementation of ZT'-127 is compared to SQUASH and DM-PRESENT-80 in terms of preimage resistance and lightweight implementations. ZT'-127 is twice as small as SQUASH. The function DM-PRESENT-80 [10] is nearly twice smaller than ZT'-127 but (according to our estimations above) comparable to ZesT-127. If we only rely on the "provable" preimage resistance of ZesT, lightweight implementations of ZesT-127 are therefore comparable to those of DM-PRESENT-80. However, based on the "heuristic" preimage resistance argued in Section 5.2, DM-PRESENT-80 should be compared with a version of ZT' four times smaller (for example, ZesT-31). According to our estimations for ZesT-127 and ZesT-251, ZesT-31 will probably require less than 1000GE, beating by far even DM-PRESENT-80.

As pointed out above, ZesT already occupies a small area if the high-speed design of Section 6.2 is used with $s = 1$. In practice, this implementation will probably be more suitable for area-constrained applications than the lightweight version presented in this section. As our design choice here was to minimize the

	Preimage Resistance	Area [GE] (rel.)
SQUASH [27]	2^{32}	<6000 (1)
DM-PRESENT-80 [10]	2^{64}	1600 (0.27)
ZT'-127 (lightweight)	2^{64} - 2^{256}	2945 (0.49)

Table 6. Comparison of the preimage resistances and area costs of lightweight implementations of ZT' and other one-way hash functions [19].

area, our implementation has a low throughput resulting in a long latency and an important energy consumption. However, the flexibility of the ZesT function allows to raise the throughput easily by increasing the number of bits of a and b processed in parallel at the cost of little additional logic. The two extreme points of this tradeoff between area and throughput are our first implementation with $s = 1$ and our lightweight implementation; the first one has a throughput 128 times as high for only 30% more area. The optimal point in practice will probably be closer to the the first one but the results of this section may be understood as a lower bound for area. Wherever the tradeoff is set, ZesT is a very interesting hash function in the context of lightweight applications.

6.4 Exploiting parallelism

ZesT can be cut into small pieces (see Figure 3). It is particularly well-suited for parallelism in the message computation. We point out that unlike many hash functions recently proposed, ZesT has both a serial and a parallel mode that describe exactly the same function. Indeed, let us suppose that we have N computing units for computing the ZesT hash value of a long message.

Fig. 3. Citrus' ZesT in serial and parallel modes

For any $\begin{pmatrix} a_0 & b_0 \end{pmatrix} \in \mathbb{F}_{2^n} \setminus \{\begin{pmatrix} 0 & 0 \end{pmatrix}\}$ and for any bitstrings $m_1, m_2, ..., m_{N'} \in \{0,1\}^*$, we have

$$\begin{aligned} &H_{ZT}^{vec}\left(P_n(X) || \begin{pmatrix} a_0 & b_0 \end{pmatrix}, m_1 || m_2 || ... || m_{N'}\right) \\ &\qquad = H_{ZT}^{vec}\left(P_n(X) || \begin{pmatrix} a_0 & b_0 \end{pmatrix}, m_1\right) H_{ZT}\left(P_n(X), m_2\right) ... H_{ZT}\left(P_n(X), m_{N'}\right). \end{aligned}$$

Moreover, the matrix version of Zémor-Tillich can be implemented as two vectorial versions starting from $\begin{pmatrix} 1 & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 & 1 \end{pmatrix}$:

$$H_{ZT}\left(P_n(X), m_i\right) = \begin{pmatrix} H_{ZT}^{vec}\left(P_n(X) || \begin{pmatrix} 1 & 0 \end{pmatrix}, m_i\right) \\ H_{ZT}^{vec}\left(P_n(X) || \begin{pmatrix} 0 & 1 \end{pmatrix}, m_i\right) \end{pmatrix}.$$

This structure can be exploited to distribute the computation of the first round of ZesT on a long message among N computing units. The messages are divided into $N+1$ blocks of equal sizes $m_0 || m_1 || ... || m_N$, the computation of $H_{ZT}^{vec}\left(P_n(X) || \begin{pmatrix} a_0 & b_0 \end{pmatrix}, m_0 || m_1\right)$ is given to the first unit and the computations of both $H_{ZT}^{vec}\left(P_n(X) || \begin{pmatrix} 1 & 0 \end{pmatrix}, m_i\right)$ and $H_{ZT}^{vec}\left(P_n(X) || \begin{pmatrix} 0 & 1 \end{pmatrix}, m_i\right)$ are given to the ith unit (see Figure 4).

The exploitation of the parallelism has two costs: first, the total computation cost of the N computing units is $\frac{2N-1}{N}$ times the computation cost of one single unit in a serial mode (it is nearly doubled when N is large). Second, $N-1$ vector by matrix multiplications must be performed at the end to combine the partial hash values. Each of these vector by matrix products requires 4 full modular multiplications and 2 additions. As the cost of a modular multiplication in $\mathbb{F}_{2^n}^*$ is about $2n$ additions [18] (a bit less if an advanced algorithm like Schönhage or Karatsuba is used), the time required to compute the ZesT hash value of a message of length μ using this method is roughly

$$2Lt_{XOR}\left[\frac{2\mu}{N+1} + (4n+1)(N-1) + 2n\right] \qquad (2)$$

where like in Section 6.1, L is the number of words needed to store n bits and t_{XOR} is the time needed to compute the XOR of two words.

The time required to compute a ZesT hash value in parallel is minimized when $N \approx \sqrt{\frac{\mu}{2n}}$. For large messages (like data disks of 100GB$\approx 2^{40}$), this would require too many computing units in practice hence the time is essentially inversely proportional to $N+1$.

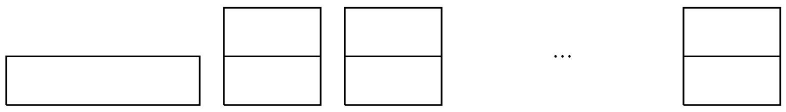

Fig. 4. Distributing computation among N computing units

A second kind of parallelism can be exploited in ZesT software computation. ZesT computes XORs, SXORs and SHIFTs on bitstrings of length n which in ANSI C are decomposed into corresponding instructions on words of length 32 or 64 bits. Using the SIMD instructions (Single Instruction, Multiple Data) that are commonly found on modern microprocessors, the function computation would be considerably sped up as 128 bits would be treated in parallel. If the implementation is performed using recent graphical accelerators with 512-bit data BUS, then even the computation of ZesT-509 will require only 2 XOR or SXOR per bit on average.

7 Adding ZesT into NIST's cooking pot

ZesT has very interesting properties from both security and efficiency points of view. It is provably collision resistant, arguably non-malleable, reasonably efficient in software, very efficient in hardware with respect to both high speed and low area metrics and it is parallelizable. However, it does not completely fulfil NIST's requirements in its call for a new hash function standard [1]. In particular, ZesT is a keyed hash function, it has suboptimal preimage, second preimage and collision resistances and it cannot be used with the output sizes required by NIST because of subgroup attacks.

In this section, we present a fixed-key variant of ZesT and we study its pseudorandom behavior. We then propose modifications of the function in order to satisfy all NIST's requirements and we discuss the efficiency and security implications of these changes. The resulting function is less efficient and considerably less simple than ZesT, but it satisfies all NIST's requirements. The various changes suggested in this section can be added independently into ZesT's recipe; we suggest that the reader selects his first-rate ingredients to satisfy his personal taste for security, efficiency, simplicity and conformity with the NIST's requirements.

7.1 Fixing all parameters

Although the formal notion of collision resistance requires defining a family of hash functions with a key as parameter, all standardized hash functions are unkeyed functions.

The key of ZesT is made of n, of an irreducible binary polynomial $P_n(X)$ of degree n, and of an initial vector $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) \in \mathbb{F}_{2^n}^2 \setminus \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix} \right)$. Besides the primality constraint on n, our choice of parameter must be clearly non-cheating because of the trapdoor attacks possible for the person who chooses $P_n(X)$ or $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) \in \mathbb{F}_{2^n}^2 \setminus \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix} \right)$. A traditional solution to this problem would be to use a cryptographic hash function (for example SHA-1) and a universal constant (for example pi) to generate the parameters.

We suggest a different approach based on an LFSR, that might present an advantage in area constrained applications. Our goal is to select a key value that is "reasonably random" (say apparently random for Joe-the-Plumber) and obviously non-cheating, and that can be recomputed efficiently to avoid its storage in memory constrained environments. We refer to Appendix A for the details of this approach.

7.2 Use of unkeyed ZesT in standardized applications

In Section 5.3 we argued that any malleability property that was present in the Zémor-Tillich hash function and its vectorial variant was removed in ZesT. To further assert "pseudorandomness", we use the Dieharder suite of pseudorandom tests [2] on the outputs of the unkeyed version of ZesT used in a counter mode. The results were very similar to what we may expect from truly random bits, hence they reveal no apparent weakness of the function. This suggests that the unkeyed version of ZesT may be used in all standardized applications including DSA [3], key derivation [5], HMAC [4] and random bit generation [6]. Details of our experiments are provided in Appendix B.

7.3 Reaching optimal (provable) collision resistance

ZesT-n is provably collision resistant up to $n/2$ bits but its output has $2n$ bits. Optimal collision resistance can be reached by replacing the vectorial version of Zémor-Tillich by its projective version in the second round. However, this change has non-negligible efficiency and portability costs.

As the projective and vectorial versions of Zémor-Tillich have the same collision resistance if n is not too large [36], we define

$$\begin{aligned}&\text{ZesT}_1(P_n(X)||\left(a_0\ b_0\right), m)\\ &\qquad := H_{ZT}^{proj}\left(P_n(X)||\left(a_0\ b_0\right), m||H_{ZT}^{vec}\left(P_n(X)||\left(a_0\ b_0\right), m\right)\right).\end{aligned}$$

ZesT$_1$ returns projective points $[a:b] \in \mathbb{P}^1(\mathbb{F}_{2^n})$ that can be represented in slightly more than n bits. Following the same reasoning as for ZesT, the security properties of ZesT$_1$ are easily derived. ZesT$_1$ has provable preimage, second preimage and collision resistance up to $n/2$ bitsBASED ON THE HARDNESS OF FINDING COLLISIONS FOR H_{ZT}^{proj} (FOR PARAMETERS NOT TOO LARGE, BASED ON THE HARDNESS OF FINDING COLLISIONS FOR H_{ZT}). This is optimal for collision resistance but suboptimal for preimage and second preimage resistance. The actual second preimage security of ZesT$_1$ is indeed not larger than $n/2$ bits but its preimage resistance is arguably as large as n bits because meet-in-the-middle attacks are impossible. Finally, ZesT$_1$ has no particular malleability weakness because the existence of such a weakness would imply the existence of a corresponding weakness in ZesT.

ZesT$_1$ has $2^n + 1$ possible outputs while it would be more convenient if the output could fit into n bits. For this reason, we suggest the following additional change in ZesT:

$$\text{ZesT}_2(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m) := \pi\left(a_0, \text{ZesT}_1(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m)\right)$$

where $\pi : \mathbb{F}_{2^n} \times \mathbb{P}^1(\mathbb{F}_{2^n}) \to \{0,1\}^n$ is defined by

$$\pi(a_0, [a : b]) = \begin{cases} b/a & \text{if } a \neq 0, \\ b_0/a_0 & \text{if } a = 0 \text{ and } a_0 \neq 0, \\ X & \text{if } a = 0 \text{ and } a_0 = 0, \end{cases}$$

for $\left(\begin{smallmatrix} a & b \end{smallmatrix}\right) := \text{ZesT}(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m)$.

ZesT$_2$ is collision resistant if H_{ZT}^{proj} is collision resistant and for parameters not too large, if and only if H_{ZT} is collision resistant. Indeed, let (m, m') be a collision for ZesT$_2$: then either the ZesT$_1$ hash values of m and m' are the same, either the ZesT$_1$ hash value of one of them (let us say m) is equal to $[a_0 : b_0]$ (if $a_0 \neq 0$) or to $[1 : X]$ (if $a_0 = 0$). Defining $h := H_{ZT}^{vec}(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix}\right), m)$, the message $m||h$ (if $a_0 \neq 0$) or $m||h||0$ (if $a_0 = 0$) collides with the void message for H_{ZT}^{proj}.

The function ZesT$_2$ is less efficient than ZesT due to the final division b/a that must be performed most of the times. Divisions in $\mathbb{F}_{2^n}^*$ can be done either with extended versions of the Euclidean algorithm or with a modular exponentiation. Algorithm 7.1 in [18] performs a division in the field with about $4n$ additions. Therefore, the time cost of performing this division is roughly the time needed to process $2n$ bits of the vectorial Zémor-Tillich. For long messages, the division time represents a small overhead but for short messages it will be significant.

The division has another significant drawback: implementations become much more complex. Although the division can be decomposed into additions, the XOR gates of our implementations cannot be reused for the division: this would require additional control logics, which would be as expensive as simply duplicating the XOR gates. The control part of both high-speed and low-area implementations will also considerably increase. Finally, the maximal frequency of the circuit is likely to decrease. The hardware performances of ZesT$_2$ should therefore be further examined in the future. In some applications, it might be interesting to compute ZesT in hardware and derive the ZesT$_2$ value in software.

7.4 Reaching optimal (heuristic) second preimage resistance

The second preimage resistance of ZesT, ZesT$_1$ and ZesT$_2$ is limited to the collision resistance level. Second preimages can be computed at the price of collisions for the first round, and they suffice to compute second preimages on the whole function. To reach an optimal level of second preimage resistance, distinct n values may be used in the first and second round, with a value about twice as large in the second round as in the first round.

For primes n and $n' \approx 2n$, irreducible polynomials $P_n(X)$, $P_{n'}(X)$ and initial vectors $\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right) \in \mathbb{F}_{2^n}^2 \setminus \{\left(\begin{smallmatrix} 0 & 0 \end{smallmatrix} \right)\}$, $\left(\begin{smallmatrix} a'_0 & b'_0 \end{smallmatrix} \right) \in \mathbb{F}_{2^{n'}}^2 \setminus \{\left(\begin{smallmatrix} 0 & 0 \end{smallmatrix} \right)\}$,

$$\begin{aligned} &\textsc{ZesT}_3(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right)||P_{n'}(X)||\left(\begin{smallmatrix} a'_0 & b'_0 \end{smallmatrix} \right), m) \\ &\qquad := \pi\left(a_0, H_{ZT}^{proj}\left(P_n(X)||\left(\begin{smallmatrix} a_0 & b_0 \end{smallmatrix} \right), m||H_{ZT}^{vec}\left(P_{n'}(X)||\left(\begin{smallmatrix} a'_0 & b'_0 \end{smallmatrix} \right), m\right)\right)\right). \end{aligned}$$

The collision and preimage resistances of $\textsc{ZesT}_3$ are identical to the previous functions. Moreover, we argue that $\textsc{ZesT}_3$ has optimal second preimage resistance. Indeed, let us suppose that on input m, there exists an algorithm that finds m' colliding with m for $\textsc{ZesT}_3$. Either m and m' have the same intermediate hash value after the first round, either not. The first case clearly reduce to the second preimage resistance of H_{ZT}^{vec} in the first round, which has a complexity of $n'/2 \approx n$ bits. The second case clearly gives a collision in the second round, but this argument only provides a security level of $n/2$ bits.

The second preimage resistance cannot be proved up to n bits based on the balance problem, but heuristic arguments fill in the gap. Indeed, a proof must assume that given m, it is infeasible to find m' such that the vectorial Zémor-Tillich hash values of m and m' are collisions for the projective Zémor-Tillich hash function. This assumption is actually a non-malleability assumption on the vectorial Zémor-Tillich function. We know that the last function is not non-malleable with respect to relations involving concatenations, but the relation here seems completely unrelated to the hash structure and the assumption therefore seems reasonable.

$\textsc{ZesT}_3$ is three times less efficient than ZesT because a vectorial hash value of m is computed with a polynomial of degree $n' \approx 2n$ in the first round and with a polynomial of degree n in the second round. The software time performances, the high speed performances and the lightweight performances will therefore decrease roughly by a factor 3.

7.5 Tweaking the function for NIST's output sizes

To avoid Steinwandt et al.' subgroup attacks [41], the parameter n must be prime in ZesT and its variants. However, standardized hash functions usually have output sizes multiple of 32 or even 64. In particular, the output sizes required by NIST for the SHA-3 standard are 224, 256, 384 and 512 [1]. In this section, we propose two alternative modifications of the function to reach these output sizes and we discuss their respective advantages and drawbacks. For each of these alternatives, we define four instances of the function that we call kumquat, lemon, orange and grapefruit, corresponding to the NIST output sizes.

The first alternative chooses n as the smallest prime larger than the targeted size and truncates the result by a few bits. The second alternative chooses n as the largest prime smaller than the targeted size, adds a third round to the function and concatenates the result of the third round with a few bits coming from the second round. The resulting parameters n and the numbers of truncated or added bits ϵ for NIST's output sizes are shown in Table 7.

Name	Output	Truncate		Extend	
		n	ϵ	n	ϵ
kumquat	224	227	3	223	1
lemon	256	257	1	251	5
orange	384	389	5	383	1
grapefruit	512	523	11	509	3

Table 7. Parameters for kumquat, lemon, orange and grapefruit in the two alternatives

Both changes have (small) negative impact on the efficiency of the function. The first variant uses a larger state and will require an additional word XOR for each addition. For kumquat and in 32-bit architecture, this is not negligible as it represents 14% additional XORs. In hardware, the effect may be considered as negligible. The second variant has a third round hence it requires about $2n$ additional full additions. This effect is negligible for long messages but not for short ones.

The first alternative is more natural and conceptually simpler. However, the collision resistance of a hash function does not imply that the function remains collision resistant if some of its bits are truncated. Of course, we believe that the resulting function is still collision resistant: the opposite would contradict in a strong sense our intuition that ZesT has no non-random behavior. Nevertheless, with this alternative we lose the benefit of provable security for collision resistance. On the other hand, preimage and second preimage resistance (up to the birthday bound) still follow from the hardness of the balance problem.

The second variant is a little more elaborate but its preimage, second preimage and collision resistances are implied by the collision resistance of ZesT. A third round is added in this function because the output of the first round is not random enough to be used for the ϵ additional bits. The function is provably preimage, second preimage and collision resistant up to $n/2$ bits which for collision resistance is only $\epsilon/2$ bits worse than the birthday bound.

8 Open problems

In this section, we discuss possible improvements in ZesT that require further study.

8.1 Use of special polynomials

Special polynomials, in particular sparse polynomials, may significantly improve the efficiency of ZesT in software [36]. Indeed, if $P_n(X) = X^n + P_{31}(X)$, a modular reduction only requires one word XOR instead of L word XORs. For large n and small architectures, this results in an efficiency improvement of nearly 25%. The efficiency in hardware, at least for our designs of Section 6.2 and 6.3, is unchanged. However, the use of sparse polynomials impacts the pseudorandomness behavior of ZEST as the bit mixing is achieved through the modular reductions.

We heuristically observed that “the images of messages with a lot of zeroes also have a lot of zeroes”. This fact may be explained from the properties of the powers of A_0 that are described in [7]. Similarly, we observed that “the images of messages with a lot of ones have a lot of zeroes”. Further study should demonstrate whether this weakness can be turned into an actual collision attack against the Zémor-Tillich hash function when sparse polynomials are used, or if its damage is limited to pseudorandom properties. In the second case, we solved the problem in [37] with an appropriate intermediate permutation between the first and the second rounds of ZEST. The purpose of this permutation is to mix the bits produced after the first round; it may be as simple as an XOR by a constant whose bits do not follow any repetition pattern, for example the bits of pi [37].

8.2 Other graph generators

From both efficiency and security points of view, the generators chosen in the Zémor-Tillich hash function are better than other Cayley hash proposals like LPS and Morgenstern [11,43,35]. There may exist other generators sets that still improve the function, but further study is required to assert their security. Balance and representation problems are not classical problems but in the case of the Zémor-Tillich hash they have at least been studied from 15 years. If the generators change, the confidence that we may have on the Zémor-Tillich hash function will not transfer automatically to the new function because the hardness of these problems seems to depend a lot on the generators choice.

From an efficiency point of view, the group $SL(2,\mathbb{F}_{2^n})$ is definitely better than the group $SL(2,\mathbb{F}_p)$ used in [11]. The function efficiency will however improve a lot if sets of 4, 8 or 16 generators can be used instead of the two generators A_0 and A_1 of the Zémor-Tillich hash function.

We may also try to change the generators to protect against the trapdoor attack on the vectorial version. The choice of A_0 and A_1 was particularly unlucky with this respect, because the attack will likely be unpractical for randomly chosen generators. Indeed, let $M, M' \in SL(2,\mathbb{F}_p)$ satisfy $\begin{pmatrix} a_0 & b_0 \end{pmatrix} M = \begin{pmatrix} a_0 & b_0 \end{pmatrix} M'$. This is equivalent to $\det(M+M') = 0$, hence to $\det(I+M'M^{-1}) = 0$ and finally to $\mathrm{Tr}(M'M^{-1}) = 0$. There are about 2^{3n} matrices in $SL(2,\mathbb{F}_p)$, among which about 2^{2n} matrices with 0 trace, hence for randomly chosen generators we would need about 2^n random products of them in order to find two matrices of the correct form. Choosing as generators the two matrices

$$A'_0 = \begin{pmatrix} X^2 & 1 \\ 1 & 0 \end{pmatrix} \qquad A'_1 = \begin{pmatrix} X & X+1 \\ 1 & 1 \end{pmatrix}.$$

seems safe with respect to the vectorial trapdoor attack. The change of X by X^2 in A_0 will not affect too much the efficiency. Moreover, this change would have the advantage to lessen greatly the density of the subset Ω generated in $SL(2, \mathbb{F}_2[X])$, which could be benefic against lifting attacks.

Since balance and representation problems in groups $SL(2, .)$ are badly known in general and since their hardness seems to depend a lot on the choice of generators, any change in the generators should be followed by its own careful security study.

8.3 Number of rounds

Heuristic reasoning and pseudorandomness tests on the outputs of ZesT used in a counter mode tend to assert that two rounds are enough for its security as a general-purpose hash function. However, we might have missed some way to exploit the group structure or even some hidden quasi-group structure in the function. In particular, we believe that the number of rounds necessary in ZesT should be better examined, especially in the light of standard attacks against traditional hash functions. We leave this question as an interesting open problem.

9 Conclusion

In this paper, we transformed a provable hash function with its inherent malleability weaknesses into a practical, all-purpose hash function. We started from the Zémor-Tillich hash function because of its elegant Cayley hash design, its potentially great efficiency and the possibility to parallelize the hash computation. Although the collision resistance of this function relies on a problem that is not classical in Cryptography, the problem has resisted 15 years of cryptanalytic attempts and we believe that it deserves broader interest in the cryptographic community.

We call ZesT our modification of the Zémor-Tillich hash function. ZesT is provably collision, preimage and second preimage resistant. Our first implementations show that it is reasonably fast in software and efficient in FPGA and that it admits ultra-lightweight implementations. In particular, it is only 4 to 10 times as slow as SHA in software, comparable to SHA on FPGA and better than any other known hash function for area constrained applications. Moreover, the hash computation can be distributed easily without affecting the result.

The ZesT function does not fill all NIST requirements in its call for a new hash algorithm, but it can be easily modified to comply with all of them. Therefore, as soon as the security of the Zémor-Tillich hash function is better trusted by further studies on balance and representation problems in non-Abelian groups, ZesT will definitely be an interesting hash candidate ... for SHA-4.

References

1. http://csrc.nist.gov/groups/ST/hash/documents/SHA-3_FR_Notice_Nov02_2007%20-%20more%20readable%20version.pdf.

2. Dieharder. http://www.phy.duke.edu/ rgb/General/dieharder.php.
3. FIPS 186-2 digital signature standard (DSS).
4. FIPS 198 the keyed-hash message authentication code. `http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf`.
5. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. `http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf`.
6. Recommendation for random number generation using deterministic random bit generators. `http://csrc.nist.gov/publications/nistpubs/800-90/SP800-90revised_March2007.pdf`.
7. K. S. Abdukhalikov and C. Kim. On the security of the hashing scheme based on SL2. In *FSE '98: Proceedings of the 5th International Workshop on Fast Software Encryption*, pages 93–102, London, UK, 1998. Springer-Verlag.
8. M. Bellare, R. Canetti, and H. Krawczyk. Message authentication using hash functions—the HMAC construction. *CryptoBytes*, 2, 1996.
9. M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT*, pages 163–192, 1997.
10. A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, and Y. Seurin. Hash functions and RFID tags: Mind the gap. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 283–299. Springer, 2008.
11. D. X. Charles, E. Z. Goren, and K. E. Lauter. Cryptographic hash functions from expander graphs. To appear in *Journal of Cryptology*.
12. C. Charnes and J. Pieprzyk. Attacking the SL2 hashing scheme. In *ASIACRYPT '94: Proceedings of the 4th International Conference on the Theory and Applications of Cryptology*, pages 322–330, London, UK, 1995. Springer-Verlag.
13. D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 470–484. Springer, 1991.
14. R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Improving SHA-2 hardware implementations. In *CHES*, pages 298–310, 2006.
15. R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Cost-efficient SHA hardware accelerators. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 16(8):999–1008, August 2008.
16. S. Contini, A. K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2006.
17. I. Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987.
18. G. M. de Dormale. *Destructive and Constructive Aspects of Efficient Algorithms and Implementation of Cryptographic Hardware*. PhD thesis, Université catholique de Louvain, 2008.
19. G. de Meulenaer, C. Petit, and J.-J. Quisquater. Hardware implementations of a variant of Zémor-Tillich hash function: Can a provably secure hash function be very efficient ? Preprint, 2009.
20. S. Even and O. Goldreich. The minimum-length generator sequence problem is NP-hard. *J. Algorithms*, 2(3):311–313, 1981.
21. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. pages 357–370. 2004.

22. M. Feldhofer and C. Rechberger. A case against currently used hash functions in rfid protocols. pages 372–381. 2006.
23. FIPS. *The Keyed-Hash Message Authentication Code (HMAC)*, Mar. 2002.
24. W. Geiselmann. A note on the hash function of Tillich and Zémor. In D. Gollmann, editor, *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 51–52. Springer, 1996.
25. S. Goldwasser, S. Micali, and R. L. Rivest. A "paradoxical" solution to the signature problem (extended abstract). In *FOCS*, pages 441–448. IEEE, 1984.
26. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, 1988.
27. F. Gosset, F.-X. Standaert, and J.-J. Quisquater. FPGA implementation of SQUASH. In *Proceedings of the 29th Symposium on Information Theory in the Benelux*, 2008.
28. S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.
29. M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theor. Comput. Sci.*, 36(2-3):265–289, 1985.
30. A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.
31. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Provably secure FFT hashing. In *NIST 2nd Cryptogaphic Hash Workshop*, 2006.
32. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In Nyberg [34], pages 54–72.
33. M. Morgenstern. Existence and explicit construction of $q+1$ regular Ramanujan graphs for every prime power q. *Journal of Combinatorial Theory*, B 62:44–62, 1994.
34. K. Nyberg, editor. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*. Springer, 2008.
35. C. Petit, K. Lauter, and J.-J. Quisquater. Full cryptanalysis of LPS and Morgenstern hash functions. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2008.
36. C. Petit, J.-P. Tillich, G. Zémor, and J.-J. Quisquater. Hard and easy components of collision search for the Zémor-Tillich hash function: new attacks and reduced variants with the same security. To appear in CT-RSA 2009, 2008.
37. C. Petit, N. Veyrat-Charvillon, and J.-J. Quisquater. Efficiency and Pseudo-Randomness of a Variant of Zémor-Tillich Hash Function. In *IEEE International Conference on Electronics, Circuits, and Systems, ICECS2008*, 2008.
38. P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. K. Roy and W. Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.
39. A. Shamir. SQUASH - a new MAC with provable security properties for highly constrained devices such as RFID tags, 2008.
40. A. Shamir and Y. Tauman. Improved online/offline signature schemes. In J. Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.
41. R. Steinwandt, M. Grassl, W. Geiselmann, and T. Beth. Weaknesses in the $SL_2(\mathbb{F}_{2^n})$ hashing scheme. In *Proceedings of Advances in Cryptology - CRYPTO 2000: 20th Annual International Cryptology Conference*, 2000.

42. J.-P. Tillich and G. Zémor. Hashing with SL_2. In Y. Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 1994.
43. J.-P. Tillich and G. Zémor. Collisions for the LPS expander graph hash function. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2008.
44. Wikipedia. Linear feedback shift register, October 2008.
45. G. Zémor. Hash functions and graphs with large girths. In *EUROCRYPT*, pages 508–511, 1991.
46. G. Zémor. Hash functions and Cayley graphs. *Des. Codes Cryptography*, 4(4):381–394, 1994.

A Fixing the key with an LFSR

To generate the $3n$ bits of $P_n(X)$ and $(\,a_0\ b_0\,)$, we use the maximal Fibonacci LFSR defined by the polynomial $x^{16}+x^{14}+x^{13}+x^{11}+1$. As n must be at least 1024 to provide the 512 bits of preimage resistance required by NIST for sensitive applications [1], the degree of the LFSR polynomial must be at least 12. We chose a polynomial of degree 16 because it might help in some applications if the LFSR register has an exact number of bytes. The polynomial $x^{16}+x^{14}+x^{13}+x^{11}+1$ has a maximal period 65535; it is the polynomial of degree 16 proposed in [44].

From this LFSR we construct the following (cryptographically weak but good enough for our purpose) pseudorandom bit generator that outputs the first bit of the state.

```
uint16_t reg = INIT ;
for (i =1;i<=65535;i++)
{
    bit = (reg & 0x0001)^((reg & 0x0002) >> 1)
            ^((reg & 0x4000) >> 14) ;
    reg = (reg >> 1) | (bit << 15);
    cout<<bit;
}
```

Let us write $g(INIT, i)$ for the ith bit output by this generator if the state is initially set to INIT. For any n value, we generate our parameters $P_n(X)$ and c as follows:

1. Fix INIT to the binary representation of pi that is 1100100100001111.
2. Build a polynomial from g as follows
   a Set $P_n(X) = X^n + 1$
   b Set the i^{th} bit of $P_n(X)$ to the value $g(INIT, i)$
3. Check wether $P_n(X)$ is irreducible. If yes go to point 6.
4. If $P_n(X)$ is not irreducible, modify it as follows:
$$P_n(x) \leftarrow X^{-2}\left(P_n(X) + (P_n(X) \bmod X^2)\right) + X^n + 1 + bX^{n-1}$$
   where b is the next bit output by the LFSR.
5. Check wether $P_n(X)$ is irreducible. If yes go to point 6, otherwise go back to point 4.
6. Take the $2n$ following consecutive bits of the LFSR to define the initial vector $(\,a_0\ b_0\,)$, from the degree 0 coefficient of b_0 to the degree $n-1$ coefficient of a_0.

B Results of Dieharder

The tests included in the version 2.28.1 of Dieharder we used are listed in Table 8. Each test returns a p-value and a diagnostic according to this p-value. The test is considered as PASSED if the p-value returned is larger than 5%. The byte sequence analyzed is considered as POTENTIALLY WEAK if the p-value is between 1% and 5%, and as POOR if it is smaller than 1%. Many of these tests are probabilistic. On perfectly random bits they will return POTENTIALLY WEAK 4% of the times and POOR 1% of the times.

```
        Diehard Tests
-d 1  Diehard Birthdays test
[sus: -d 2  Diehard Overlapping Permutations test]
-d 3  Diehard 32x32 Binary Rank test
-d 4  Diehard 6x8 Binary Rank test
-d 5  Diehard Bitstream test
-d 6  Diehard OPSO test
-d 7  Diehard OQSO test
-d 8  Diehard DNA test
-d 9  Diehard Count the 1s (stream) test
-d 10 Diehard Count the 1s (byte) test
-d 11 Diehard Parking Lot test
-d 12 Diehard Minimum Distance (2D Spheres) test
-d 13 Diehard 3D Spheres (minimum distance) test
-d 14 Diehard Squeeze test
[sus: -d 15 Diehard Sums test]
-d 16 Diehard Runs test
-d 17 Diehard Craps test
-d 18 Marsaglia and Tsang GCD test
[dev: -d 19 Marsaglia and Tsang Gorilla test]
```

```
           RGB Tests
-r 1 RGB Timing test (times the rng)
-r 2 RGB Bit Persistence test
-r 3 RGB Ntuple Bit Distribution test suite (-n ntuple)
-r 4 RGB Generalized Minimum Distance test
-r 5 RGB Permutations test (new, partial replacement for operm tests)]
[rft: -r 6 RGB Lagged Sums test
      (do not use the following as tests yet)
[dev: -r 7 RGB L-M-Ntuple Distribution test suite (quite long)]
[dev: -r 8 RGB Overlapping Permutations test]

   Statistical Test Suite (STS)
-s 1 STS Monobit test
-s 2 STS Runs test
-s 3 STS Serial test

         User Tests
-u 1 User Template (Lagged Sum Test)
```

Table 8. Tests implemented into Dieharder version 2.28.1 [2]. The tests preceded by “rft” are ready for testing (the test may - or may not - work correctly) in that version, the tests preceded by “sus” are suspect (they consistently fail “good” generators) and the tests preceded by “dev” are under development.

ZesT was used in a counter mode with n in 31, 61, 127, 157, 223, 251, 383, 509, 1021 and the parameters fixed in the previous section. More precisely, we computed the hash values of $1, 2, ..., 1.000.000$, we truncated them by a few bits to fit perfectly into bytes, we concatenated the results and we analyzed the resulting byte sequence with the Dieharder suite.

The results are shown in Table 9. Each column is a version of ZesT labeled by its parameter n and each row is a test of Dieharder labeled as in Table 8. As some tests are repeated, each entry shows the number of tests for which the result was PASSED, POTENTIALLY WEAK and POOR. The results for ZesT are very similar to what we may expect from random bytes: for each n value, a few tests are failed below 5% and very few tests are failed below 1%.

Test	31	61	127	157	223	251	383	509	1021
- r 3	31a1b	11a1c	11a1b	11a1b	11a1b	12a	12a	12a	11a1b
- r 4	4a	4a	3a1b	4a	4a	4a	4a	3a1b	3a1b
- r 5	5a1c	5a1c	6a	6a	5a1b	6a	6a	6a	6a
- r 6	33a	33a	33a	31a2b	33a	33a	32a1b	31a1b1c	33a
- d 1	1a	1a	1a	1a	1a	1a	1a	1b	1a
- d 3	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 4	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 5	1a	1a	1b	1a	1a	1a	1a	1a	1a
- d 6	1c	1a	1a	1a	1a	1b	1a	1c	1a
- d 7	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 8	1a	1a	1a	1b	1a	1a	1a	1a	1a
- d 9	1a	1b	1a	1a	1a	1a	1a	1a	1a
- d 10	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 11	1a	1a	1a	1a	1a	1a	1a	1b	1a
- d 12	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 13	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 14	1a	1a	1a	1a	1a	1a	1a	1a	1a
- d 16	2a	2a	2a	2a	2a	2a	2a	2a	2a
- d 17	2a	2a	2a	2a	2a	2a	2a	2a	2a
- d 18	2a	2a	2a	2a	2a	2a	2a	2a	2a
- s 1	1a	1a	1a	1a	1a	1a	1a	1a	1a
- s 2	1a	1a	1a	1a	1a	1a	1a	1a	1a
- s 3	29a1b	29a1b	30a	29a1c	30a	28a1b1c	30a	30a	27a3b
- u 1	1a	1a	1a	1a	1a	1a	1a	1a	1a

Table 9. Results of Dieharder for ZestT. The tests are identified as in Table 8. As some tests are repeated, each entry gives the number of tests whose verdicts were PASSED, POTENTIALLY WEAK and POOR. a=PASSED (p-value larger than 5%); b=POTENTIALLY WEAK (p-value between 1% and 5%); c=POOR (p-value smaller than 1%).