Hardware Implementations of a Variant of the Zémor-Tillich Hash Function: Can a Provably Secure Hash Function be very efficient ?

Giacomo de Meulenaer
Christophe Petit and Jean-Jacques Quisquater

UCL Crypto Group
Université catholique de Louvain
Place du Levant, 3
B-1348 Louvain-la-Neuve - Belgium
{giacomo.demeulenaer,christophe.petit, jjq}@uclouvain.be

Abstract. Hash functions are widely used in Cryptography, and hardware implementations of hash functions are of interest in a variety of contexts such as speeding up the computations of a network server or providing authentication in small electronic devices such as RFID tags. Provably secure hash functions, the security of which relies on the hardness of a mathematical problem, are particularly appealing for security, but they used to be too inefficient in practice. In this paper, we study the efficiency in hardware of ZT', a provably secure hash function based on the Zémor-Tillich hash function. We consider three kinds of implementations targeting a high throughput and a low area in different ways. We first present a high-speed implementation of ZT' on FPGA that is nearly half as efficient as state-of-the-art SHA implementations in terms of throughput per area. We then focus on area reduction and present an ASIC implementation of ZT' with much smaller area costs than SHA-1 and even than SQUASH, which was specially designed for low-cost RFID tags. Between these two extreme implementations, we show that the throughput and area can be traded with a lot of flexibility. Finally, we show that the inherent parallelism of ZT' makes it particularly suitable for applications requiring high speed hashing of very long messages. Our work, together with existing reasonably efficient software implementations, shows that this variant of the Zémor-Tillich hash function is in fact very practical for a wide range of applications, while having a security related to the hardness of a mathematical problem and significant additional advantages such as scalability and parallelism.

Keywords: Hash function, Zémor-Tillich, Hardware, FPGA, ASIC.

1 Introduction

Hash functions are widely used in cryptographic applications such as digital signatures schemes, message authentication codes, commitment schemes or password storage. Typically, a hash function is required to have nearly uniform output distribution and to be both preimage and collision resistant: it must be computationally hard to find a preimage to a given randomly chosen hash value, as well as to find two messages hashing to the same hash value. Additionally, a hash function is often required to behave "like a random function". Due to the importance of cryptographic hash functions, the SHA family was designed as a standard [2].

Hardware implementations of standard cryptographic hash functions like MD5 or SHA have been achieved to accelerate computations in specific applications [14, 37, 23, 11, 19, 10]. As an example, a virtual network server typically uses a hash function both to verify the identities of its clients (with a digital signature protocol) and the integrity of the messages they send (with an authentication protocol). If the server has many clients, a cryptographic hardware accelerator may help to quickly compute the hash values of many, potentially long messages. For cost reasons, such an accelerator is often implemented in an FPGA rather than an ASIC. Very compact ASIC implementations of hash functions are also required to provide security for small low-cost devices such as low-power wireless sensors or RFID tags. However, standard hash functions like SHA are not suited for lightweight applications [17, 5]. In many RFID applications, collision resistance is not needed but preimage resistance is a sufficient property, hence other hash functions like SQUASH [34] were specially designed, satisfying preimage resistance but not collision resistance. If collision resistance is needed for compact implementations, the authors of [5] recommend the use of block cipher-based hash functions but they point out that this approach presents scalability problems.

Recently discovered vulnerabilities in SHA-1 [38] prompted NIST to launch a competition for a new Secure Hash Algorithm [1] and underlined the need for secure and efficient hash functions. Particularly appealing from a theoretical point of view, some hash functions are *provably secure*, in the sense that their collision resistance relates to the hardness of some mathematical problems [29, 12, 7, 36]. A good reduction to a simply formulated mathematical challenge facilitates the evaluation process and increases the confidence once the function has resisted the first cryptanalytic attempts. However, it has been a common believe in the cryptographic community that provably secure hash functions cannot reach the efficiency level of dedicated algorithms like SHA.

The Zémor-Tillich hash function was proposed by Tillich and Zémor at CRYPTO'94 and remains essentially unbroken today [36, 33]. The collision and preimage resistance of this function depend on the hardness of the representation and balance problems in a non-Abelian group. These kinds of problems are well-known in expander graphs theory; for cyclic Abelian groups, they are equivalent to discrete logarithm problems [4] and for generic groups, they are known to be hard [26, 15]. Although provably secure in the sense of collision resistance, the Zémor-Tillich hash function would present several security issues if it was used as a general-purpose hash function. A variant of this function was therefore introduced by Petit, Veyrat and Quisquater [33] both to solve these issues and to speed up the hash computation in software. It follows from a result of Petit, Quisquater, Tillich and Zémor [32] that the collision resistance of this variant, after a small modification in the key generation algorithm, is equivalent to the collision resistance of the original function, resulting in a provably secure hash function with the same level of collision resistance and with increased efficiency and security. In software, the function is faster than or comparable to other hash functions with provable security [20, 21, 13, 6, 9, 12], except the very efficient SWIFFT algorithm [30]. It is currently 10-50 times slower than SHA [33], but the use of graphical instructions such as SIMD should improve these performances greatly. In [36, 33], the authors suggested that the Zémor-Tillich hash function and its variant would perform reasonably well in hardware since the operations involved in the hash computation are very simple.

In this paper, we show that the provably secure hash function described in [33] is indeed very interesting for a wide range of hardware implementations, including implementations that focus on a high throughput per area, a small area, or a high speed efficiency metric. We first provide a FPGA implementation reaching a throughput per slice about half as good as very optimized FPGA implementations of SHA-1 and SHA-2 such as [10, 11, 19]. We then present an ASIC implementation requiring much less area than any currently used hash function. Particularly, our compact implementation requires less than half of the area of the lightweight implementation of SHA-1 presented in [17] and even less than the SQUASH implementation of [22]. Furthermore, the function allows developing designs that are very flexible in the sense that they can be easily modified to produce different throughputs. Finally, we show that the inherent parallelism of the function makes it particularly suitable for applications requiring high-speed hashing of single long messages. Our work, together with reasonably efficient existing software implementations described in [33], shows that the variant of the Zémor-Tillich hash function proposed by Petit et al. is in fact very practical for a wide range of applications, while having a security related to the hardness of a mathematical problem and significant additional advantages such as scalability and parallelism.

This paper is organized as follows: Section 2 presents the Zémor-Tillich hash function and its variant introduced by [33]; Section 3 describes a high-speed FPGA implementation that is half as efficient as SHA in terms of area-time efficiency; Section 4 details a lightweight implementation more compact than SHA and even than SQUASH; Section 5 discusses potential uses of the parallelism of ZT' for the high-speed hashing of very long messages and Section 6 concludes the paper.

2 A Variant of the Zémor-Tillich Hash

We now describe the Zémor-Tillich (ZT) hash function, following [36]. Let $m = m_1 m_2 ... m_k$ be the bit string representation of the message to be hashed. Let $P_n(X)$ be an irreducible polynomial of degree n and let us see the field $\mathbb{F}_{2^n}$ as $\mathbb{F}_2[X]/(P_n(X))$. Let A_0 and A_1 be the following matrices

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

Define the following mapping

$$\begin{aligned} \pi : \{0,1\} &\rightarrow \{A_0, A_1\} \\ 0 &\rightarrow A_0 \\ 1 &\rightarrow A_1. \end{aligned}$$

The hashcode of m is just the matrix product

$$h_{ZT}(m) := \pi(m_1)\pi(m_2)...\pi(m_k)$$

where the arithmetic is made modulo $P_n(X)$, that is in the field $\mathbb{F}_{2^n}$. As for the parameter size, Tillich and Zémor proposed to take $n \in [130, 170]$.

The security of the ZT hash can be related to both algebraic and graph-theoretical problems [36, 7]. More exactly, it relies on the hardness of a balance and a representation problem in a matrix group. Representation problems are equivalent to discrete logarithm problems in cyclic Abelian groups [4] and they are known to be hard in generic groups [26, 15]. For non-Abelian groups, representation problems are not known to reduce to any classical problem in Cryptography but they have been long studied in the theory of expander graphs [25]. Recent results in this field are still far from any explicit solution [27, 24]. The Zémor-Tillich has remained essentially unbroken since its publication at CRYPTO'94. The attacks of Charnes and Pieprzyk and of Steinwandt et al. only target particular parameters [8, 3, 35]. They are discarded if n is prime and if the polynomial is well-defined to avoid the insertion of trapdoors. Geiselmann's attack [18] produces messages of expected length larger than 2^n hence it cannot be considered as practical. Recently, Petit et al. [32] have described collision and preimage attacks of complexity $2^{n/2}$ that can be applied for all values of the parameters n and $P_n(X)$.

Although fundamentally unbroken in the sense of preimage and collision resistance, the Zémor-Tillich hash function has two main security problems: it is invertible when short messages are hashed and it is malleable. The malleability property directly results from the associativity of the matrix product: given the hash $h(m)$ of an unknown message m, the hash value of any message $x_1||m||x_2$ can easily be computed. As a consequence, the use of ZT hash must be avoided in applications requiring some form of pseudo-randomness.

To solve these issues, Petit, Veyrat and Quisquater introduced ZT', the variant of Zémor-Tillich hash [33] described below. Their basic idea consists in XORing the result of the ZT function with a constant and to consider the result as if it was the continuation of the original message. Moreover, for efficiency purposes, they restricted the hash result to the first row of the ZT hash result, halving therefore the computations. As a consequence, their variant is:

$$H(m) := h_{ZT}^{vec}(m||\sigma(h_{ZT}^{vec}(m)))$$

where $h_{ZT}^{vec}(m)$ is the concatenation of the entries $(1,1)$ and $(1,2)$ of $h_{ZT}(m)$ computed with the same parameters n and $P_n(X)$, and $\sigma : \{0, ...2^{2n}-1\} \to \{0, ...2^{2n}-1\} : x \to x \oplus c$. The parameter c is some fixed constant the bits of which "look like random"; they suggest to use c equal to the binary representation of pi.

As $h_{ZT}^{vec}(m_1...m_{k+1}) = h_{ZT}^{vec}(m_1...m_k)A_{m_{k+1}}$, the two entries a and b of the function h_{ZT}^{vec} may be computed bit by bit with the formulae

$$(a_{k+1}, b_{k+1}) = \begin{cases} (a_k, b_k)A_0 = (a_kX + b_k, a_k) & \text{if } m_{k+1} = 0 \\ (a_k, b_k)A_1 = (a_kX + b_k, a_kX + a_k + b_k) & \text{if } m_{k+1} = 1 \end{cases} \quad (1)$$

where the additions and the multiplications by X are in the field $\mathbb{F}_{2^n} \approx \mathbb{F}_2[X]/(P_n(X))$. The *initial vector* (a_0, b_0) is set to $(0, 1)$. In the following, we informally use the term *final phase* to refer to the processing of the additional $2n$ bits of $\sigma(h_{ZT}^{vec}(m)))$.

It is shown in [32] that if the initial vector (a_0, b_0) is chosen randomly in $\mathbb{F}_{2^n}^2 \setminus \{(0,0)\}$ the collision resistance of h_{ZT}^{vec} is equivalent to the collision resistance of h_{ZT}. Therefore, our only change with respect to the function of [33] is that the initial vector is no more fixed to $(1, 0)$ but can be more generally specified. The initial vector is chosen randomly in $\mathbb{F}_{2^n}^2 \setminus \{(0,0)\}$ during the key generation process and is part of the key.

Using the results of [32], it is easy to check that the resulting variant of the ZT function, which has an output size of $2n$ bits, is provably collision resistant up to $2^{n/2}$ bits and that a collision attack with this complexity can indeed be mounted. It is also provably preimage resistant up to $n/2$ bits, but as the preimage attack of [32] cannot compute preimages of h_{ZT}^{vec} with the very special form $m||\sigma(h_{ZT}^{vec}(m))$, the actual security level of h_{ZT}^{vec} is of $2n$ bits today. In [33] the authors suggest using the following sparse irreducible polynomials $P_n(X)$ to improve efficiency in software:

n	Polynomial
127	$X^{127} + X + 1$
251	$X^{251} + X^7 + X^4 + X^2 + 1$
509	$X^{509} + X^8 + X^7 + X^3 + 1$
1021	$X^{1021} + X^5 + X^2 + X + 1$
2039	$X^{2039} + X^{10} + X^9 + X^8 + X^7 + X^5 + X^4 + X^2 + 1$

These polynomials are safe with respect to the attacks developed in [18, 3, 35]. The value $n = 127$, which has a security level comparable to SHA-1, could be considered as too small for high security applications at the light of the attacks in [32]. We use the same polynomials in our hardware implementations, although more general irreducible polynomials would not decrease performances.

3 High-Speed Implementation

In this section, we describe a high-speed implementation of ZT' on FPGA. High-speed 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. In the following, we first describe our architecture, then we compare our implementation performances to state-of-the-art implementations of well-known hash functions, and we finally discuss possible improvements.

3.1 Proposed Architecture

The main feature of ZT' is that it considers the bits of the message one after the other. The dependency between the intermediary results of the hash function suggests an iterative architecture where the message is processed one bit at the time. The efficiency can be improved by processing s consecutive bits at each step, which amounts to partially unroll the main loop. The optimal value for s (the unroll factor) is determined empirically. When processing one bit of the message, the bits of the two entries a and b of the matrix can be efficiently computed in parallel. Indeed, the operations involved in the hash function are simple: they essentially consist in XOR operations between two or three operands and 1-bit left shifts (see Equation 1). These bitwise operations allow computing the bits of a and b in parallel without decreasing the frequency, which would not be the case it there was a carry propagation for instance. The parallel approach is better than the serial one: the latter alternative would involve a wordwise processing and would therefore require extra control logic to select and route the appropriate words during the execution of the function, resulting in a lower throughput per slice efficiency.

Our architecture is depicted in Figure 1. It is a partially unrolled architecture as described in [28], which processes s bits per clock cycle. It is made of s identical cores connected in a serial manner and of storage elements. Each core processes one message bit by achieving the

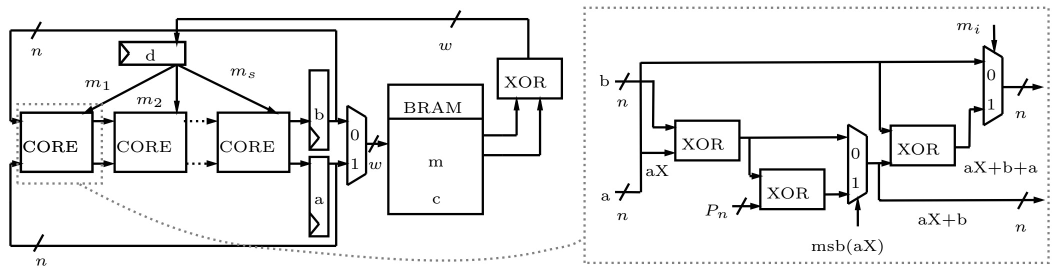

Fig. 1. Overview of the proposed architecture of the variant of ZT on FPGA (left) and core details (right).

operations corresponding to Equation 1. The s consecutive message bits, m_i with $1 \leq i \leq s$, come from a w-bit shift register denoted d containing a word of the message to hash. The partial results are stored in two n-bit registers, denoted a and b, and are used as inputs when processing the next s bits of the message. They loop in the circuit until the end of the message is reached. Then they are loaded in the block ram (BRAM) and are reused as if they were the continuation of the message in the final phase. The BRAM also stores the message, denoted m, and the pseudorandom constant c (a pi binary representation) by words of $w = 32$ bits. The two output ports of the BRAM are connected to a w-bit XOR that is used to implement the function σ, i.e., the XOR operation with c preceding the final phase. Before the final phase, zero is outputted in place of c to route correctly the words of m to the core of the circuit through the w-bit XOR gate.

The throughput achieved by our partially unrolled architecture can be approximated by the product of s and the frequency if the processing of the final phase is negligible. This is valid for sufficiently long messages with respect to the $2n$ additional bits of the final phase. In the following of this section, we assume that the messages fulfill this condition.

The unroll factor s must be properly chosen in order to maximize the throughput per slice. A small s ensures a large operation frequency together with a small occupied area. Increasing s is interesting in the sense of the considered metric, since it is proportional to the throughput of the circuit but only to the core which is only a fraction of the total area. On the other hand, increasing s also adds logic operators to the longest path, resulting in a lower maximum operating frequency. The optimal value for s was therefore found by an empirical study.

3.2 Implementation Results

The function was implemented on the Xilinx Virtex-2 XC2V2000-6. ISE 8.2 was used for synthesis and place and route. Testing and debugging were performed with Modelsim SE 6.1. The first implementation results determined the optimal unroll factor s. As shown in Table 1 for the function with $n = 127$, the highest throughput per slice was obtained with $s = 5$. This was also the case for the other parameters that we investigated, $n = 251$ and 509. In the following, the implementation results are therefore given for $s = 5$. We point out that the evolution of the area and the frequency in function of s in Table 1 is not as regular as one would expect when considering the unrolled architecture of Figure 1. This is due to the underlying structure of the Virtex-2 FPGA that uses 4-input look-up tables (LUT). The optimal value for the unroll factor is likely to differ on other types of FPGAs, such as the 6-input LUT based Virtex 5.

Table 1. Implementation results of ZT'-127 in function of the unroll factor s.

Unroll Factor s	Area [Slices]	Frequ. [Mhz]	Through. [Mbits]	Through/Area [Mbits / Slice]
1	262	220	220	0.84
2	377	215	430	1.14
3	515	185	555	1.08
4	596	170	680	1.14
5	597	160	800	1.34
6	647	130	780	1.21
7	794	120	840	1.06
8	901	110	880	0.98

The implementation results concerning the first three values of the n parameter (127, 251 and 509) are given in Table 2. The impact of n on the frequency is moderate as increasing n does not add logic operators to the longest path (as s does). The small frequency drop is likely due to larger routing delays.

Table 2. Implementation results for ZT'(with $s = 5$).

n	Area [Slices]	Frequ. [Mhz]	Through. [Mbits]	Through/Area [Mbits / Slice]
127	597	160	800	1.34
251	1044	140	700	0.67
509	1850	135	675	0.37

In Table 3, we compare these results to the very optimized implementations of SHA proposed in [11] and [10] representing the current state-of-the-art in terms of achieved throughput per occupied area. The results show that the performances of ZT' and of SHA in terms of throughput per slice are in the same order. At comparable level of collision resistance, our implementations of ZT'-127, ZT'-251 and ZT'-509 are about half as efficient as state-of-the-art implementations of SHA-1, SHA-256 and SHA-512 respectively.

3.3 Future Improvements

Our implementations already reach the level of performances of SHA, but we believe that they can be further improved by introducing pipeline stages between the s cores. This technique should allow increasing the operating frequency while still processing s bits per clock cycle, these bits being from different messages this time. This would significantly increase the

Table 3. Comparison of the performances of the high-speed implementations of ZT' with SHA. The collision resistance of the hash functions is also given. The FPGA used is the Virtex-2 XC2V for all designs except for SHA-1 that was implemented on the Virtex-2 Pro XC2VP30-7.

	Collision Resistance	Area [Slices]	Frequ. [Mhz]	Through. [Mbps]	Through/Area [Mbps / Slice]
SHA-1 [11]	2^{63}	533	230	1435	2.7
ZT'-127	2^{64}	597	160	800	1.34
SHA-256 [10]	2^{128}	797	150	1184	1.49
ZT'-251	2^{126}	1044	140	700	0.67
SHA-512 [10]	2^{256}	1666	121	1534	0.92
ZT'-509	2^{255}	1850	135	675	0.36

throughput at a relatively small area cost, i.e., mainly a $2n$-bit register per pipeline stage. Since the designs with large s leave many slices of the Virtex 2 FPGA with unused flip-flops, the cost of additional pipeline stages could even be much lower, depending on the routing constraints. In theory, a single pipeline stage could increase the frequency of the design of ZT'-127 with $s = 8$ (110 MHz, see Table 1) to the frequency of the design with $s = 4$ (170 MHz, see Table 1), resulting in a throughput of roughly 1300 Mbps in place of 880, i.e., roughly 50% improvement.

The control part of the pipelined architecture will be more complicated in the case of messages of different sizes as ZT' does not process the message by fixed size blocks. Indeed, when one message in the pipeline will come to the processing of the final phase or when it will be fully hashed, the computations for all messages will be irregularly interrupted. In applications where the message sizes are integer multiples of some block size, the control of the pipelined architecture is simplified as the interruptions for the final phase and the final result happen at the same processing steps for each block.

4 Lightweight Implementation

In this section, we study an architecture of ZT' minimizing the area, which is of interest in area-constrained environments such as RFID tags. As shown in Table 1, ZT'-127 with $s = 1$ (i.e., not unrolled) already occupies a relatively small area: we now modify this architecture to focus on the area reduction.

Figure 2 presents the architecture of our lightweight implementation of ZT'. 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 (this is illustrated on the figure with the presence of the j suffix, with $0 \leq j \leq n-1$). Therefore, processing one message bit m_i takes n clock cycles instead of one previously.

The second main change involves the storage elements. For lightweight implementations, large blocks of memory like BRAM are no longer available and they are therefore replaced by two registers (labeled $a+c$ and $b+c$) that store the result of the XOR operation between the intermediary result and the constant c, which is hardcoded. The outputs of these registers are used as the continuation of the message during the final phase. This architecture assumes

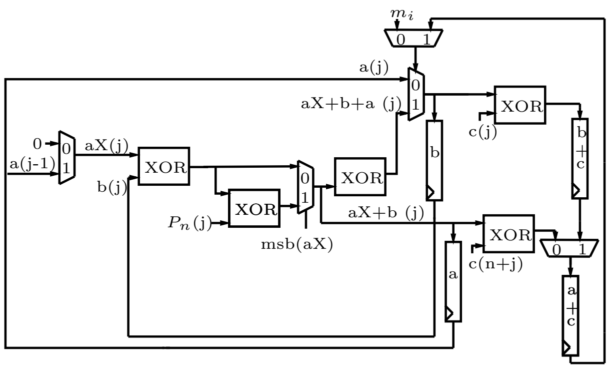

Fig. 2. Proposed lightweight architecture of the variant of ZT

that the message is loaded in the circuit one bit at the time and therefore does not require a dedicated storage element. The four n-bit registers in this architecture are shift registers that do not have a parallel load since the circuit operates on 1-bit signals. Their 1-bit inputs and outputs are represented as wires respectively at their tops and bottoms. At each clock cycle, the register a outputs two bits $a(j-1)$ and $a(j)$, the register b gives $b(j)$ and both registers take as inputs the two bits computed by the circuit. The registers $a+c$ and $b+c$ also input these bits but only after a XOR operation with the constant bits $c(n+j)$ and $c(j)$. During the final phase, the output of the register $b+c$ is redirected toward the register $a+c$, which provides the bits of the continuation of the message. At the end of the final phase, the hash values, stored in the registers a and b, are also retrieved one bit at the time.

The latency of this circuit when hashing a message m containing k bits is $(k+2n).n$ clock cycles. The corresponding throughput cannot be approximated any more by leaving aside the processing of the additional $2n$ bits of the final phase since the messages are likely to be small in the context of lightweight implementations. In the following, it is computed based on the formula of the latency given above with $k = 512$, i.e., assuming consecutive messages of 512 bits. This size is arbitrary as there is no fixed block size in the ZT' function.

The design was synthesized using Synopsys Design Analyzer version Y-2006.06 with the CMOS65 library of STMicroelectronics. To provide a comparison with the architecture of the preceding section, it was also implemented on FPGA. For n=127, it requires only 73 slices with a frequency of 145 MHz. This particularly small area requirement is due to the use of compact SRL16 registers to implement the shift registers (with no parallel load), which significantly diminishes the number of slices used.

Table 4 summarizes the results concerning ZT' and other hash functions (based on the comparison performed in [5]). Results concerning the AES block cipher are also given as a reference. The results for SQUASH are based on the estimate performed in [22], which describes a lightweight implementation on the Xilinx Virtex-4 LX FPGA. This estimate must be seen as an upper bound. Finally, we also synthesized the designs ZT'-127 and ZT'-251 with $s = 1$ of the preceding section to evaluate the area reduction obtained with our lightweight design. Although the implementations listed in Table 4 involve different logic processes, the comparison is fair since the throughput is given at a fixed frequency and the area is expressed in terms of Gate Equivalents (GE).

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

	Output size	Through. at 100kHz [kbps]	Through/Area [bps/GE]	Logic process	Area [GE]
MD5 [17]	128	83.7	10	0.13μm	8400
SHA-1 [17]	160	40.2	4.9	0.35μm	8120
SHA-256 [17]	256	45.4	4.2	0.35μm	10868
SQUASH [22]	32	< 0.1	< 0.02	estimate	<6000
AES-128 [16]	128	12.4	3.7	0.35μm	3400
DM-PRESENT-80 [5]	64	14.6	9.1	0.18μm	1600
H-PRESENT-128 [5]	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

ZT' is very efficient in terms of occupied area with respect to current hash functions. Both its lightweight and high-speed (with $s = 1$) versions with the smallest parameter n= 127 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 [16]. It is however a little less compact than H-PRESENT-128, the hash function recently proposed in [5], based on the block cipher PRESENT. The area requirements and collision resistances of ZT' and SHA are compared in Table 5, illustrating the lower area costs for ZT' at a comparable collision resistance. At comparable levels of 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.

Table 5. Comparison of the collision resistance and area cost of the lightweight implementation of ZT' with SHA.

	Collision Resistance	Area [GE] (rel.)
SHA-1 [17]	2^{63}	8120 (2.8)
ZT'-127 (light.)	2^{64}	2945 (1)
SHA-256 [17]	2^{128}	10868 (2)
ZT'-251 (light.)	2^{126}	5517 (1)

Table 6. Comparison of the preimage resistance and area cost of the lightweight implementation of ZT' with preimage-resistant hash functions.

	Preimage Resistance	Area [GE] (rel.)
SQUASH [22]	2^{32}	<6000 (3.8)
ZT'-127 (light.)	$\in [2^{64}, 2^{254}]$	2945 (1.8)
DM-PRESENT-80 [5]	2^{64}	1600 (1)

For some applications, collision resistance is not required and a moderate level of security is sufficient (64-bit or 80-bit security) [34]: for example, many RFID authentication protocols only rely on preimage resistance. ZT' turns out to be a very interesting candidate for these applications. As shown in Table 6, its lightweight version with $n = 127$ is twice as small as the upper bound given for SQUASH and about twice as large as the function DM-PRESENT-80 [5] based on the compact block cipher PRESENT. However, the comparison with DM-PRESENT-80 is not very fair since the preimage resistance of ZT'-127 today seems to be 2^{254} instead of its 2^{64} "provable part" as explained in Section 2. Parameter four times smaller should therefore be considered in the comparison, resulting in an area cost of the lightweight ZT' equivalent or smaller than DM-PRESENT-80, and possibly even smaller than 1000 GE. It should therefore easily fit in the 2000 GE available for security in low-cost RFID tags, as stated in [5].

As pointed out above, our first implementation of ZT'-127 with $s = 1$ already occupies a small area. 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 area, our implementation has a low throughput implying a long latency and an important energy consumption. However, the flexibility of the ZT' function allows raising 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, our results show that ZT' is a very interesting hash function in the context of lightweight applications.

5 Very Fast Hashing of Long Messages Using Parallelism

If only one (long) message needs to be hashed, the usual throughput per area metric looses sense since replicating hardware to process more messages in parallel is pointless. For example, a large data server may need to compute the hash value of its content to guaranty its integrity, and update this hash value every time a legitimate user modifies the data. For this application, a hardware implementation will need to primarily satisfy a time constraint before caring about area. In this section, we point out that ZT' is very well-suited for such an application since its first round can be computed in parallel. Indeed, for any $m_1, m_2, ...m_N \in \{0,1\}^*$, we have

$$h_{ZT}^{vec}(m_1||m_2||...||m_N) = h_{ZT}^{vec}(m_1)h_{ZT}(m_2)...h_{ZT}(m_N).$$

Moreover, as the matrix version of Zémor-Tillich can be implemented as two vectorial versions starting from $(1,0)$ and $(0,1)$, the computation can be easily distributed into $2N-1$ computing units using our architecture of Section 3 (possibly with a different s value). The exploitation of the parallelism has two costs: first, the total computation cost of the $2N-1$ 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, requiring $4(N-1)$ full modular multiplications. Depending on the exact size of the messages, it might be more efficient to use hardware/software co-design consisting in computing partial products in hardware and using software to combine them and compute the final phase: this technique will be investigated in a further work. We point out that unlike other hash functions, the inherent parallelism of ZT' allows trading area for speed in the hash computation of *one* single message.

6 Conclusion

In this work, we studied the efficiency in hardware of the provably secure hash function based on Zémor-Tillich introduced by Petit et al. [33]. In particular, we presented a high-speed FPGA implementation, we provided an ASIC implementation focusing on reducing the area costs, and we discussed how to compute the hash value of long messages.

Our results show that this function exhibits high performances in a wide range of implementation contexts, which is usually unexpected from a hash function with an algebraic structure. The efficiency of our high-speed FPGA implementation is about one half of the efficiency of very optimized SHA implementations and it could be further optimized by introducing pipeline. Our lightweight ASIC implementation requires less area than currently used hash functions such as SHA-1 or CBC-AES and even than SQUASH, which was specially designed for low-cost RFID tags. For protocols that only rely on preimage resistance, it should easily fit in the 2000 GE available for security in low-cost RFID tags, as stated in [5]. Finally, the inherent parallelism of the function makes it an ideal candidate for hashing long messages very quickly. Besides the hardware implementations considered, we stress that other tradeoffs between area and throughput for differently constrained applications can also be easily achieved. The lower software performances with respect to SHA may not have too much practical consequences because the hash function computation [33] still outperforms by far the asymmetric cryptographic operations used in many protocols. Moreover, applications requiring high performance hashing can resort to more efficient processing units such as graphical accelerators or typically to hardware implementations.

Our work, together with reasonably efficient existing software implementations described in [33], shows that the variant of the Zémor-Tillich hash function proposed by Petit et al. is in fact very practical for a wide range of applications, while having significant advantages over dedicated and block cipher-based hash functions including scalability, parallelism and a collision resistance reducing to the hardness of a mathematical problem.

References

1. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.
2. FIPS 180-2 secure hash standard.
3. 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.
4. M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT*, pages 163–192, 1997.
5. 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.
6. S. A. Brands. An efficient off-line electronic cash system based on the representation problem. In *246*, page 77. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1993.
7. D. X. Charles, E. Z. Goren, and K. E. Lauter. Cryptographic hash functions from expander graphs. To appear in *Journal of Cryptology*.
8. 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.

9. 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.
10. R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Improving SHA-2 hardware implementations. In *CHES*, pages 298–310, 2006.
11. 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.
12. 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.
13. I. Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987.
14. J. Deepakumara, H. Heys, and R. Venkatesan. FPGA implementation of MD5 hash algorithm, 2001.
15. S. Even and O. Goldreich. The minimum-length generator sequence problem is np-hard. *J. Algorithms*, 2(3):311–313, 1981.
16. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. pages 357–370. 2004.
17. M. Feldhofer and C. Rechberger. A case against currently used hash functions in RFID protocols. pages 372–381. 2006.
18. 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.
19. R. Glabb, L. Imbert, G. Jullien, A. Tisserand, and N. Veyrat-Charvillon. Multi-mode operator for SHA-2 hash functions. *J. Syst. Archit.*, 53(2-3):127–138, 2007.
20. S. Goldwasser, S. Micali, and R. L. Rivest. A "paradoxical" solution to the signature problem (extended abstract). In *FOCS*, pages 441–448. IEEE, 1984.
21. 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.
22. 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.
23. T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott. Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. In *ISC '02: Proceedings of the 5th International Conference on Information Security*, pages 75–89, London, UK, 2002. Springer-Verlag.
24. H. A. Helfgott. Growth and generation in SL2(Z/pZ), 2005.
25. S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43:439–561, 2006.
26. M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theor. Comput. Sci.*, 36(2-3):265–289, 1985.
27. M. Kassabov, A. Lubotzky, and N. Nikolov. Finite simple groups as expanders, 2005.
28. R. Lien, T. Grembowski, and K. Gaj. A 1 gbit/s partially unrolled architecture of hash functions sha-1 and sha-512. In T. Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2004.
29. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Provably secure FFT hashing. In *NIST 2nd Cryptogaphic Hash Workshop*, 2006.
30. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In Nyberg [31], pages 54–72.
31. 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.

32. 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 at CT-RSA 2009, 2009.
33. 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.
34. A. Shamir. SQUASH - a new MAC with provable security properties for highly constrained devices such as RFID tags. In *FSE*, pages 144–157, 2008.
35. 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.
36. 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.
37. K. K. Ting, S. C. L. Yuen, K. H. Lee, and P. H. W. Leong. An FPGA based SHA-256 processor. In *FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*, pages 577–585, London, UK, 2002. Springer-Verlag.
38. X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.