ॐ भूर्भुवस्व तत् सवितूर्वरेण्यम् भर्गो देवस्य धीमहि धियो यो न प्रचोदयात्

EnRUPT Hash Function Specification

Sean O'Neil
VEST Corporation
France

Karsten Nohl, PhD
University of Virginia
USA

Luca Henzen
ETH Zürich University
Switzerland

www.EnRUPT.com

1. Introduction

This document specifies the ïrRUPT stream hashing mode of the EnRUPT family of cryptographic algorithms first published at SASC 2008 [ER007]. ïrRUPT is an iterative stream hash function that processes a message one w-bit word at a time to produce a condensed representation of the message called message digest.

Seven stream hash functions are proposed herein for the general-purpose use: ïrRUPT32-128, ïrRUPT32-160, ïrRUPT32-192, ïrRUPT64-224, ïrRUPT64-256, ïrRUPT64-384 and ïrRUPT64-512. The seven proposed algorithms differ mostly in the number of bits of security that are provided for the data being hashed, which is half of the message digest size. Other algorithms may require the use of a secure hash algorithm that provides a certain number of bits of security. For example, ïrRUPT64-256 providing 128 bits of security can be used if a message is being signed with a 128-bit secure digital signature algorithm requiring the use of a secure hash function that also provides 128 bits of security.

Additionally, the seven proposed algorithms differ in their word size and size of internal state. Basic properties of the seven algorithms are presented in Table **1**.

Hash	ïrRUPT-128	ïrRUPT-160	ïrRUPT-192	ïrRUPT-224	ïrRUPT-256	ïrRUPT-384	ïrRUPT-512
Message Size (bits)	$<2^{64}$	$<2^{80}$	$<2^{96}$	$<2^{112}$	$<2^{128}$	$<2^{192}$	$<2^{256}$
Word Size (bits)	32	32	32	64	64	64	64
State Size (bits)	256+96	320+96	384+96	512+192	512+192	768+192	1024+192
Hash Size (bits)	128	160	192	224	256	384	512
Security (bits)	64	80	96	112	128	192	256
Rounds per Word	8	8	8	8	8	8	8
Additional Rounds per Message	104–112	128–136	152–160	104–112	104–112	152–160	200–208

Table 1 Basic properties of the proposed EnRUPT stream hash functions

All seven proposed general-purpose algorithms use the same security and parallelization parameters defined in section **2.1**, s=4 and P=2. All the specified parameters can be varied to obtain hash functions with desired properties.

1.1 Definitions

1.1.1 Glossary

Bit	A binary digit having values of either 0 or 1.
Byte	A group of eight bits.
Word	A group of either 32 bits (4 bytes) or 64 bits (8 bytes), depending on the algorithm parameter w.

1.1.2 Symbols and operations

mod	Remainder of division, the result of x mod n is a remainder of division of x by n; % operator in C programming language.
+	Arithmetic addition operation limited to w-bit integers, also known as addition mod 2^w or depicted as ⊞; + operator in C.
*	w-bit arithmetic multiplication ($\cdot \bmod 2^w$); * operator in C.
⊕	w-bit bitwise XOR (exclusive-OR) operation; ^ operator in C.
&	w-bit bitwise AND operation, also known as ∧; & operator in C.
|	w-bit bitwise OR operation, also known as ∨; | operator in C.
¬	w-bit bitwise complement (NOT) operation; ~ operator in C.
<<	w-bit bitwise shift left operation; x << n discards the leftmost n bits of the word x padding the result on the right with n zero bits, identical in its operation to $x*2^n$; << operator in C.
>>>	w-bit bitwise rotation right, where **rotr**(x,n) = x >>> n is computed by cyclic rotation of the word x to the right by n bits.
/	Rounded integral division operation; / operator in C.

1.1.3 Terms and parameters

c	Word of ciphertext, in big-endian format.
d	A set of P delta accumulators.
d_i	i^{th} (i mod P) delta accumulator.
(ïr1)	Irreversible EnRUPT round function; returns d_{P-1}.
f	Temporary variable containing intermediate result of **(ïr1)**.
h	Size of the final hash value in bits; $s \cdot w \cdot P/2 \le h \le 3 \cdot s \cdot w \cdot P$.
H	Number of words in the internal state x; $s \cdot P \le H \le 6 \cdot s \cdot P$, $2 \cdot h \le H$.
m	Number of bits in the input message.
N	Number of sealing rounds; $N=2 \cdot s \cdot H$ for all EnRUPT stream modes.
o_i	i^{th} word of hash output or keystream, in big-endian format.
M or p	message or 'plaintext' to be hashed or encrypted or both.
p_i	i^{th} word of plaintext input, in big-endian format.
P	Parallelization parameter, generally $1 \le P \le 4$.
r	Round number, starts with 0.
s	Security parameter, $s \ge 4$ is recommended.
w	Word width, 32 or 64 bits.
x	H words of internal state, recommended H=(h+P*w–1)/w/P*P.
x_i	i^{th} (mod H) word of internal state.

1.2 Bit strings, *w*-bit words and integers

The following terminology related to bit strings and integers will be used:

1. A hex digit is an element of the set {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}. Each hex digit represents a 4-bit string corresponding to the same number in binary form. For example, the hex digit "7" represents the 4-bit string "0111", and the hex digit "B" represents the 4-bit string "1011".

2. A word is a *w*-bit string that may be represented as a sequence of hex digits. To convert a word to hex digits, each 4-bit string is converted to its hex digit equivalent, as described in (1). For example, the 32-bit string

   1010 1101 0101 1011 0101 0001 1010 1110

   can be expressed as "AD5B51AE", and the 64-bit string

   1110 1101 0000 1101 1110 1100 0010 1011
   1010 1101 0101 1011 0101 0001 1010 1110

   can be expressed as "ED0DEC2BAD5B51AE".

   Throughout this specification, the "*big-endian*" convention is used when expressing both 32-bit and 64-bit words, according to which the most significant bit is stored in the left-most bit position within each word.

3. An integer may be represented as a word or a number of words. An unsigned integer between 0 and $2^{32}-1$ inclusive may be represented as a 32-bit word (from 00000000 to FFFFFFFF). The least significant four bits of the integer are thus represented by the rightmost hex digit of the word representation. For example, the integer $4680 = 2^{12} + 2^9 + 2^6 + 2^3 = 4096+512+64+8$ is represented by the hex word 00001248. Similarly, an unsigned integer between 0 and $2^{64}-1$ inclusive may be represented as a 64-bit word (from 0000000000000000 to FFFFFFFFFFFFFFFF).

4. Only single-word *w*-bit unsigned integers are used in EnRUPT hash functions. The words of internal state *x*, the delta accumulators *d*, the round number *r* and the temporary variable f are all *w*-bit unsigned integers. For EnRUPT32, the message is represented in 32-bit words, and for EnRUPT64, the message is represented in 64-bit words.

2. EnRUPT Structure and Operation

ïrRUPT*w*xP-*h*/*s*	input m bits of message p and location for h bits of hash o; set $p_{m/w} = (1 << (\neg m \,\&\, (w-1))) \mid p_{m/w} \,\&\, (-1 << (\neg m \,\&\, (w-1)))$; set H = (2*$h$+P*$w$–1)/$w$/P*P; set $x_{0..H-1} = d_{0..P-1} = r = 0$; for i = 0 to m/w execute **ïr2s**(p_i), set i += 1; execute **ïr2s**(h); for i = 0 to H–1 execute **ïr2s**(0), set i += 1; for i = 0 to (h–1)/w set o_i = **ïr2s**(0), set i += 1; Return h bits of o as the final hash value.
ïr2s(p)	execute **(ïr1)** 2*s times; set d_{P-1}⊕=p; return d_{P-1};
(ïr1), P=2	set x_{r+2} ⊕= f = rotr(2*$x_{r\oplus1}$ ⊕ x_{r+4} ⊕ $d_{r\&1}$ ⊕ r, w/4)*9, set $d_{r\&1}$ ⊕= f ⊕ x_r, set r += 1;

Fig. 1 Complete EnRUPT stream hashing mode specification in pseudocode.

(ïr1), P=1	set x_{r+1} ⊕= f = rotr(2*x_r ⊕ x_{r+2} ⊕ d_0 ⊕ r, w/4)*9, set d_0 ⊕= f ⊕ $x_{r+H/2+1}$, r += 1;
(ïr1), P=4	set x_{r+4} ⊕= f = rotr(2*$x_{r\oplus1\oplus(r\&1)*2}$ ⊕ x_{r+8} ⊕ $d_{r\&3}$ ⊕ r, w/4)*9, set $d_{r\&3}$ ⊕= f ⊕ x_r, r += 1;
(ïr1), any P	set x_{r+P} ⊕= f = rotr(2*$x_{r/P*P+((r+1)\%P)}$ ⊕ x_{r+2*P} ⊕ $d_{r\%P}$ ⊕ r, w/4)*9, set $d_{r\%P}$ ⊕= f ⊕ $x_{r+H*P/2+(P\&1)}$, r += 1;

Fig. 2 (ïr1) round function pseudocode for other P values.

Note: All x indexes in the pseudocode are evaluated as modulo H.

2.1 EnRUPT parameters

Parameter	Description	Acceptable values
W	Word size	w = 32 for h=128, 160, 192, 224, 256, … 768 w = 64 for h=256, 320, 384, 448, 512, … 1536
P	Parallelization level	P = 2^n, P=2 for general-purpose applications
h	Final hash bits	$h = s{\cdot}w \le h \le 6{\cdot}s{\cdot}w$
s	Security level	s = 1 for non-cryptographic applications s = 2 to resist passive distinguishers s = 3 to resist non-adaptive attacks s = 4 to resist all adaptive attacks s = 4 for general-purpose applications $s > 4{\cdot}sw$ for high-assurance applications

Fig. 3 ïrRUPT[w]x[P]-[h]/[s] required parameters

2.2 EnRUPT round function (ïr1)

The following two figures demonstrate the structure of one **(ïr1)** round function and the structure of two **(ïr1)** round functions executed in parallel.

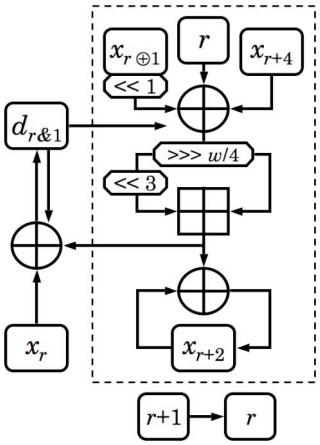

Fig. 4 Single **(ïr1)**, P=2.

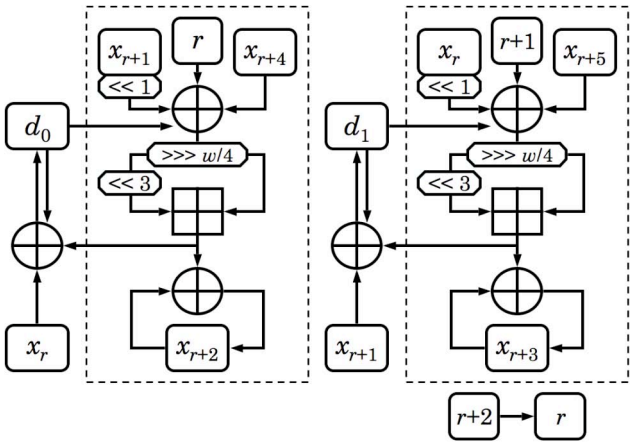

Fig. 5 Two parallel **(ïr1)**, P=2.

In pseudocode, the irreversible EnRUPT round function **(ïr1)** is defined as

(ïr1)

set f = rotr($2*x_{r/P*P+((r+1)\bmod P)} \oplus x_{r+2*P} \oplus d_{r\bmod P} \oplus r$, $w/4$)*9
set $x'_{r+P} = x_{r+P} \oplus$ f
set $d'_{r\bmod P} = d_{r\bmod P} \oplus$ f $\oplus x_{r+H*P/2+(P\&1)}$
set $r' = r+1$,

where x', d' and r' represent next-round values of the updated word of internal state, delta accumulator and the round number. **ïr2s**(p) defines $2*s$ rounds of **(ïr1)** linearly combining d_{P-1} with the word p and returning d_{P-1} as output.

2.3 EnRUPT structure

Each ïrRUPT[w]x[P]-[h]/[s] algorithm consists of H w-bit words of internal state $x_0..x_{H-1}$, P delta accumulators $d_0..d_{H-1}$ and the round function **(ïr1)**, which also uses round number r in its operation. The number of words in x is calculated as

$$H = (2*h+P*w-1)/w/P*P$$

H should not be less than $s \cdot P$, although smaller states can be used for research, cryptanalysis and non-cryptographic applications.

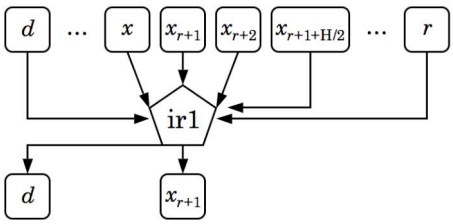

Fig. 6 EnRUPT structure, P=1.

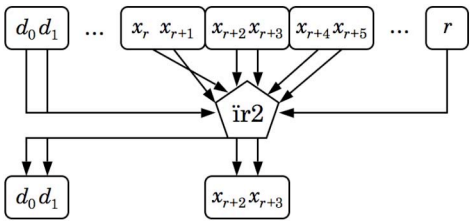

Fig. 7 EnRUPT structure, P=2.

At the high level, EnRUPT structure is almost identical to that of XXTEA [XXT]. The unparallelizable **(ïr1)** with P=1 is identical to **(ir1)** [ER007] round function. All seven proposed general-purpose hash functions use 2x parallelization.

2.4 EnRUPT operation in ïrRUPT mode

The basic (unkeyed/unrandomized/etc) ïrRUPT stream hashing mode of operation consists of four stages: preprocessing, computation, finalization and hash output.

1. During the preprocessing stage, a message is padded and parsed into w-bit words, and the internal state is reset.
2. The computation stage updates the internal state with word operations using words of the padded message and the round number.
3. The finalization stage seals the internal state by diffusing the last input bits securely throughout the state.
4. The final value generated by the hash output stage is the message digest.

2.4.1 Preprocessing

1. Append one bit **1** to the end of the message, followed by $w - ((m+1) \bmod w)$ bits **0**. Then pad the message in a multiple of w-bit words as the protocol requires, where randomization, message length, hash bit length, message number and any other variables and parameters may be included.
2. Set all H=(2*h+P*w–1)/w/P*P words of the internal state x to 0. Set N = 2*s*H. Set r = 0. Set d_0 = 0. Set d_1 = 0.

2.4.2 Message processing

1. Execute **ïr2s**(p_i) once for each word of the message.

2.4.3 Finalization

1. Execute **ïr2s**(h) once.
2. Iterate **ïr2s**(0) H times.

2.4.4 Output

1. Save o_i=**ïr2s**(0) as the i^{th} word of the final hash value.
2. Repeat the above step $(h+w-1)/w$ times.
3. Truncate the last bits of o for the odd-size hash values.

Being a stream hash, ïrRUPT processes the message one w-bit word at a time:

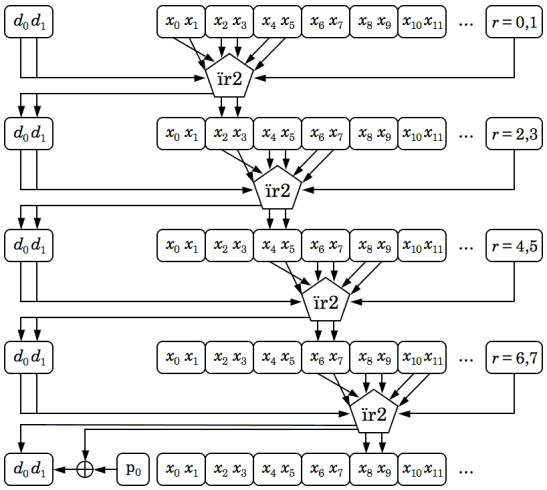

Fig. 8 Hashing the first p_0 word of the message (s=4).

2.5 PRF, MAC, HMAC, AE and randomized hashing

Construction of PRF, T/PRNG, MAC or HMAC, authenticated encryption or randomized hashing from ïrRUPT is straightforward for any of the variants:

1. Initialise the state according to 2.4.1. The key/IV/salt and the plaintext must be padded according to 2.4.1.
2. If necessary, hash the secret key according to 2.4.2.
3. Hash the IV/salt according to 2.4.2.
4. Seal the internal state according to 2.4.3.
5. Further continuous output of **ïr2s**(0) is a PRF/PRNG (RUPT mode), which can be used as keystream for binary additive encryption (a stream cipher); continuous output of **ïr2s**(p_i) can be used as ciphertext for authenticated encryption (aeRUPT mode) or as a true RNG continuously loading entropy as p_i; the message stream is also processed with **ïr2s**(p_i) for randomized hashing or HMAC (mcRUPT mode) but disregarding the output.
6. After processing the message with **ïr2s**(p_i) and sealing the internal state according to 2.4.3 for the second time, the final hash value from the step 2.4.4 can serve as a MAC, HMAC or a randomized hash value.

2.6 Parallel modes

ïrRUPT can also be safely used in any of the parallel hashing modes such as tree hashing [TREE], 4-time XOR hashing proposed by Bernstein in [XOR4] or MD6 [MD6] by dividing the stream into blocks or chunks of fixed size and hashing them in parallel according to **2.4**.

3. Design Principles

The design of cryptographic functions remains largely a trial-and-error process in which known attacks are tested against arbitrarily guessed candidate functions until constructs that withstand all attacks are found. The AES competition has found the current standard in block ciphers by soliciting constructs that are hard to break, but did not require the design methodology of the constructs to be reproducible or even comprehensible. While the functions that come out of a cipher competition are usually strong, little insight is gained into how the different components of a function interact to provide its strength. Furthermore, the evaluation of new cryptographic functions is often limited to the analysis of its different parts, all of which must be strong for a cipher to be accepted. This design approach leads to algorithms unnecessarily complex and expensive. We demonstrate that equally strong while significantly cheaper designs can be found by evaluating candidate functions as a whole with automated cryptanalysis tools.

3.1 Kerckhoffs desiderata

In 1883 Auguste Kerckhoffs has published six desiderata for cryptographic systems in his work La Cryptographie Militare [LCM]. We translate these desiderata to the modern age cryptography as:

1. The cost of breaking the cryptosystem should be higher than the benefit.
2. Compromise of one system should not affect [the users of] other systems.
3. Keys should be as short as possible and rekeying should be as fast as possible.
4. It should store and process data in the most conveniently aligned binary form.

5. The algorithm should occupy as little area, code and memory as possible.
6. **The algorithm should be of minimal complexity, as fast as possible, and its implementation should not cause any mental strain.**

The real users of modern cryptosystems are software and hardware developers. While "no security through obscurity" has become a popular slur misquoting the second desideratum, the sixth one requiring simplicity seems to have been so far overlooked. EnRUPT is designed to be as simple as possible. Although security was never sacrificed, some of its performance was traded in favour of simplicity.

3.2 Design methodology

In order to find functions that are simple yet strong, we have developed tests that measure the cryptographic strength of candidate functions by evaluating randomness of distribution of monomials in all the polynomial relationships between bits. We use these tests to find the strongest constructs from a design space containing a vast number of simple functions. Randomness of the polynomial structure of a function ensures sufficiently high complexity and absence of a detectable bias, guaranteeing unbiased pseudorandom output.

To test a candidate function for polynomial randomness, we reconstruct the weakest possible parts of its polynomial structure and run standard randomness tests on those parts. Given a set of sufficiently independent round functions that are indistinguishable from random, the Luby-Rackoff theory [LR] provides us with a straightforward way of building functions verifiably resistant to all statistical attacks. Luby-Rackoff theorem states that a Feistel Network construction iterating four independent round functions, all of which are indistinguishable from random in polynomial time, will resist all adaptive and non-adaptive statistical attacks. Therefore, finding verifiably secure cryptographic functions that are immune to statistical attacks can be reduced to finding functions indistinguishable from random and combining them in Feistel network constructions. Ciphers with a sufficiently random structure also seem to be highly resistant to algebraic attacks since these attacks rely on the sparseness of a cipher's polynomial structure.

By design, any function selected by this process is resistant to all attacks considered in the design methodology, as long as the right heuristics are chosen in making the design tools practical. Anchoring the security claims in the automated design methodology and not in any concrete function has the additional advantage that further cryptanalysis is significantly accelerated and its value is increased because the number of structural flaws that may have been overlooked is significantly reduced. Cryptanalysts can concentrate on the simple tools used to create a cipher instead of having to cut through the (often unnecessary) layers of complexity inherent in many cryptographic primitives. Should new statistical attacks be discovered through cryptanalysis, useful feedback is gained for improving the design tools' heuristics to consider the new attacks as well. Our approach, therefore, provides a positive feedback loop that converts new cryptanalysis techniques into improved tools to create ciphers that are inherently resistant to the new attacks. All known statistical attacks such as linear and differential cryptanalysis as well as algebraic attacks were already considered in the design tools used to find EnRUPT.

We demonstrate the effectiveness of our design methodology by identifying EnRUPT from a large design space predominantly populated with weak design choices. Our design space is defined by all round functions that only use a small number of shifts, rotations, arithmetic additions and XOR operations.

The following sections discuss the mathematical foundation of our design methodology demonstrating their direct correlation with all statistical attacks including but not limited to, linear cryptanalysis, differential cryptanalysis and Mod n cryptanalysis [MODN].

3.2.1 Monomial distribution

A function can be defined by a set of polynomials that express output bits in terms the input bits. The i^{th} bit of a string x is denoted x_i. We consider functions that take one input string, x consisting of *all* the input bits including both, the key and the data bits, and produce an output string: $y \leftarrow f(x)$. Each bit of the output string, y_i, can be expressed as a binary function of the input bits. Each term, known as a *monomial*, in the algebraic normal form (ANF) of these binary functions is a conjunction of one or more input bits. Each ANF of a function with n input bits (key bits plus data bits) has the general form:

$$y_i = a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 + \ldots + a_{1,n} \cdot x_n$$
$$+ a_{2,1} \cdot x_1 \cdot x_2 + \ldots + a_{2,(n-1)n} \cdot x_{n-1} \cdot x_n$$
$$+ \ldots + a_{n,1} \cdot x_1 \cdot x_2 \cdot \ldots \cdot x_n$$

where the j^{th} monomial of degree i has a coefficient $a_{i,j} \in \{0,1\}$ and the terms correspond to the powerset of the input bits. The string formed by these coefficients, $a_{1,1} \| a_{1,2} \| \ldots \| a_{n,1}$ completely describes the function. Next we show that if the combined set of ANFs of those functions is sufficiently random, the complete function is indistinguishable from a random function and can be used for construction of cryptographically strong algorithms.

3.2.2 Luby-Rackoff theory

A *pseudorandom function family* (PRF) is a family of functions for which the output of a randomly chosen member of the family is indistinguishable from random for any efficient (i.e., polynomial time) distinguisher. The randomness of the output is a much weaker property than the randomness of the ANF of a function: a random ANF always leads to an output that is indistinguishable from random (Proposition 1 in [SAAR]), while a random-looking output can also be produced by a complex, but not random ANF that can be distinguished from random by either statistical or algebraic attacks.

Given a PRF (or a function that is indistinguishable from a PRF), we rely on the Luby-Rackoff theory to build a more globally pseudorandom permutation that is resistant to adaptive and non-adaptive cryptanalysis [LR]. The Luby-Rackoff theorem states that after three iterations of a Feistel network with independent PRFs, the mapping between (random) inputs to the Feistel network and its output cannot be distinguished from a random function by a polynomial time attacker. If the inputs to the Feistel network are not random, a fourth iteration is required to make the function indistinguishable from random by an adaptive polynomial time attacker with access to the decryption process.

Let r be the number of rounds after which a construct becomes a PRF or LRF. Normally, $2r$ Feistel rounds resist passive distinguishers from random without known plaintext, $3r$ rounds resist non-adaptive statistical attacks by known or chosen plaintext or ciphertext attacks, and $4r$ rounds resist all adaptive attacks (including square, rectangle, boomerang, etc.). Patarin, Naor, and Reingold provided similar proofs for Benes networks and for unbalanced Feistel networks [PRP], [BBSR].

EnRUPT is a source-heavy unbalanced Feistel network. Although the exact number of subrounds required for provable resistance to adaptive attacks is smaller than $4r$ for unbalanced Feistel networks, we have adopted the highest value $4r$ for its simplicity, since it is not significantly higher than the actual number of rounds that will resist all attacks and cryptographers always appreciate an extra security margin.

3.2.3 Partial analysis of large ANF

To test conclusively how well a given design implements a function that is indistinguishable from a PRF we would have to compute its complete ANF and test it for randomness. However, exhaustively computing and analysing ANFs of arbitrarily large functions is intractable. Instead, we apply heuristics designed to provide good coverage for all known cryptographic attacks. These heuristics can show immunity to statistical and (known) algebraic attacks that operate on small sets of inputs and outputs.

We argue that it is the non-randomness of the ANF detectable for a sufficient number of rounds that is responsible for the success of all the known statistical and algebraic attacks on symmetric cryptographic primitives, although the function's outputs may appear perfectly random. Differential and linear cryptanalysis, for instance, exploit the strong connection between small groups of input and output bits [DC], [LR], and algebraic attacks exploit the sparseness of ciphers in their polynomial representation [ALG], which is detectable as sparseness, non-randomness or otherwise "compressability" of their ANF.

As our heuristic, we choose to compute the monomials for each subset of input bits up to a certain size. For a cipher with n input bits, we generate the polynomial relations between the output bits and all sets of p input bits for all the values of p up to an upper bound m. This upper bound is usually constrained by computational power, since computing the ANF of p input bits of an arbitrary function takes $p \cdot 2^{(p-1)}$ executions of the function by Algorithm 1 in [SAAR]. The algorithm takes the truth table of a function (or an oracle that generates it) and computes the monomials starting from the smallest ones and iteratively xors these smaller monomials into the truth table as more inputs are set to **1**. All input bits that are not part of the p inputs under consideration are set to **0** so that the polynomials can have at most 2^p monomials.

Values up to m=40 can be computed on commodity hardware for simple ciphers within minutes. During our tests, we find that all detectable flaws are reflected in subsets of lower degrees (around p=16..20), while the higher degree tests can almost never distinguish a function that is passed as pseudorandom by the lower-degree tests. The tests that operate on large numbers of inputs often do not even detect the flaws that are detected by testing smaller subsets since the noise in

the tests is increased as more bits are included that do not contribute to the flaw. Therefore, it is important that smaller sets of inputs are tested first.

Our observation that most weaknesses are caused by small sets of inputs is consistent with results from linear and differential cryptanalysis, which always lead to attacks involving only small sets of inputs and outputs. Differential cryptanalysis is a class of attacks where a known difference in the input bits produces a known difference in the output bits with an exploitably high probability. Linear cryptanalysis is a related attack that finds the best affine approximations to the Boolean functions iterated for many rounds and exploits the higher probability of approximation to find some of the key bits faster than by brute-force. Linear and differential analysis, hence, exploit a strong connection between small sets of input and output bits.

Algebraic attacks are another example of attacks that exploit only local non-randomness since they rely on the sparsity of monomials. Low sparsity has led to successful attacks on a number of stream ciphers [ALG], but is detected by our tests automatically. We therefore conjecture that testing ANF of all low degree subsets is sufficient for detecting exploitable cryptographic weaknesses.

3.2.4 Randomness tests

Counting monomials of ciphers to detect statistical weaknesses a [STAT]. His *d-monomial* test computes the number of monomials present in the polynomials of a given function and requires that roughly half of all possible monomials are present. The same test was applied by Saarinen to the candidates of the eSTREAM competition for stream ciphers [SAAR]. Many of the analysed ciphers were found to have polynomial structures that are too sparse and would hence potentially be vulnerable to statistical and algebraic attacks. While any overly sparse (or overly dense) polynomial is distinguishable from random, the inverse does not hold. We propose to test polynomial structures for properties beyond the average monomial count by applying standard randomness tests.

Our analysis is constrained by the general infeasibility of conclusive randomness tests. None of the strings we generate are truly random (since we know a way of generating them), but they may still be indistinguishable from random in polynomial time. We define a sufficient level of local randomness in a practical way: any bias that the attacker could possibly exploit can also be tested for with the right randomness tests. It should, therefore, be possible to prevent all statistical attacks through sufficient randomness testing of the algebraic structure of all the relationships between all the input and output bits.

As shown by the Luby-Rackoff theorem, once a function is indistinguishable from random after a certain number of rounds, a Feistel Network built with that function becomes resistant to all statistical attacks after only four times that number of rounds. If the round functions are independent as required, slide attacks [SLID] also do not apply. Our tests also verify that property.

The choice of ordering of the monomials matters very little for the randomness tests. The amount of entropy is not altered by any fixed transpositions, thus randomness tests distinguish differently ordered ANF sequences equally well. Simple accumulation of all the monomials into a single stream in the order they are generated (by Algorithm 1 in [SAAR], for instance) is sufficient to detect local non-randomness.

Our tests rely on well-known randomness tests such as the NIST and `diehard` statistical test suites. For example, the `diehard` bitstream test takes a string of 2^n binary symbols and counts how often different n-bit words are found as subsets of the string. In a random 2^n-bit long string, the number of missing n-bit words should be normally distributed with mean $\frac{2^n}{e}$. The test is performed on a number of different 2^n-bit strings from the source under the test and is passed if all of them have a distribution reasonably close to the random distribution according to a Z-test.

By applying these standard tests to monomial distributions, weaknesses in the polynomial structure of several cryptographic primitives have been found. Randomness tests from the diehard suite successfully detect the vulnerability of the TEA cipher to related-key attacks for any number of rounds, detect large classes of weak keys in complete IDEA, as well as distinguishing up to 4 rounds of AES and up to 27 rounds of SHA-1 from random [47].

3.3 Design

Formally speaking, the structure of EnRUPT is a consistent incomplete source-heavy heterogenous UFN (unbalanced Feistel network) [UFN]. Despite its apparent simplicity, it seems to be an optimal structure preserving its strength with states of any size. Simplicity of every aspect of the complete design was our primary goal and turned out to be the hardest thing to achieve while maintaining high performance with a single simple set of rules for all the possible sizes. It is actually very easy to save a few clock cycles per byte at the cost of simplicity.

3.4 Parameters

3.4.1 s, security, set to 4

A certain minimal amount of message expansion is required for ïrRUPT to hash data securely. Therefore the proportion between the word size and the state size must be sufficiently high. The internal state should not be less than H=s·P words and cannot be less than 2·h, which makes the smallest supported hash to be s=4 words in size for the proposed ïrRUPTx2.

It is not advised to reduce the security parameter below s=4 for general-purpose applications. However, some applications may benefit from the higher speeds if security against certain classes of attacks is not required. Protocol designers wishing to modify ïrRUPT parameters outside the recommended ranges must consider the security implications presented in sections **4.6**, **4.10** and **4.11** below.

3.4.2 P, parallelisation, set to 2

The parallelisation parameter P=2 is proposed for the general-purpose hashing. This offers the most optimal balance between constrained environments and extra large microchips, pipelined parallel CPUs and GPUs.

EnRUPT with P=4 is of course two times faster in FPGA and ASIC, but it allows only at least P·4 (16-word) internal states with 4-word increments, suitable only for the 256-bit and 512-bit hashes. While P=1 lets small RFID chips implement even 64-bit or 96-bit hashes with small 128-bit or 192-bit states, its lack of

parallelisability makes it too inflexible for a wide range of pipelined software processors and too slow for the unconstrained hardware.

Different parallelization options for different hash sizes would confuse developers making it difficult to reuse the round function. Therefore we chose P=2 sacrificing the top hardware speed to preserve simplicity and balanced efficiency.

3.4.3 r, round number

EnRUPT is not the first and certainly not the last algorithm to use the round index as means to break self-similarity of the round function, at least for each group of 2^w rounds, which is more than enough to render slide attacks infeasible.

Our tests described in **3.2** and the previously discovered collision attacks against hash functions make it clear that pseudo-random round constants contribute nothing to the security of the algorithm other than breaking self-similarity of the round function and *possibly* also removing fixed points. The round number or any other long counter is a much more reliable way to achieve both those objectives.

3.4.4 w, word width

The word width is set to 32 for ïrRUPT-128, -160 and -192 and to 64 for ïrRUPT-224, -256, -384 and -512 to support the register sizes of the modern and the immediate future processors to make current software implementations efficient. However, all of the EnRUPT variants can safely use either 32-bit or 64-bit words.

EnRUPT32 and EnRUPT64 cannot be directly compared with EnRUPT16 or EnRUPT128 or any other word-width scaled variants. The word width plays a significant role in the stability of the algorithm. Odd word widths require careful adjustment of the rotation and shift amounts and may not be secure at all.

4. Security

EnRUPT is an ADD-ROL-XOR algorithm that owes its security to the non-linearity provided by the rotated arithmetic addition operation when it is interleaved with linear addition (XOR).

	PRF rounds / input words								
State Size:	2	4	5	6	8	10	12	14	16
TEA	∞	–	–	–	–	–	–	–	–
XTEA	4.75	–	–	–	–	–	–	–	–
XXTEA	5.25	–	–	–	–	–	–	–	–
MD5-3	–	1.69	–	–	–	–	–	–	–
MD5-1	–	1.5	–	–	–	–	–	–	–
SHA-0/1	–	–	1.69	–	–	–	–	–	–
irRUPT32	4	2.5	2.2	1.83	1.75	1.6	1.58	1.57	1.56
irRUPT64	4.5	2.75	2.4	2	1.88	1.8	1.67	1.64	1.63
ïrRUPT32	4	4	3	2.67	2.25	2.1	2	2	2
ïrRUPT64	4.5	5	3.2	3	2.63	2.4	2.33	2.29	2.25

Table 2 Strengths of different ADD-ROL-XOR round functions (P=1 and P=2)

As can be seen from Table **2** above, the strength of the ïrRUPT round function is comparable to other ADD-ROL-XOR ciphers. The number of PRF rounds does not grow linearly for very small state sizes. Only states of at least H=8 words for P=2 are advised for cryptographic applications, at which size the number of full PRF rounds for ïrRUPT lowers sufficiently close to 2. It is consistent with the requirement for a sufficient state/input proportion discussed in Section 3.4.1. The state sizes of 12, 14 and 16 words proposed in this specification for SHA-3 can benefit from the lowest number of rounds, while the states with less than 8 words would obviously require more rounds to maintain the required security level.

ïrRUPT is a stream hash and there is currently no set of known attacks that can be applied directly. Therefore we can only speculate on the potential effectiveness of known attacks and explore other possible directions for cryptanalysis.

4.1 Linear and differential cryptanalysis, $\min(2^{\mathbf{H}\cdot w-h},2^h)$

The source heavy structure of **(ïr1)** makes differential attacks more effective than linear [UFN]. Our preliminary analysis shows absence of iterative characteristics in EnRUPT with blocks or states of any size. Our basic statistical attacks quickly grow beyond 2^h in complexity for any EnRUPT with $s\geq 3$, but advanced linear and differential attacks must be studied in more detail.

4.2 Length extension, $\min(2^{\mathbf{H}\cdot w-h},2^h)$

Being a stream hash with a state much larger than the output, ïrRUPT mode is naturally resistant to the length extension attacks unlike Mercle-Damgärd constructions. Knowledge of the hash output is insufficient to continue hashing. h+P·w bits of the internal state are missing. The attacker must guess those to continue hashing the stream correctly. Thus we can safely assume at least h-bit security against length extension attacks.

4.3 Collision resistance, $\min(2^{\mathbf{H}\cdot w/2},2^{h/2})$

We believe that hash function collision resistance is identical to resistance to adaptive statistical, algebraic and structural chosen key+plaintext+ciphertext attacks nondifferent from those applied to block ciphers or stream ciphers. Since collision attacks are new and largely unexplored, we will refrain from speculation leaving it to the cryptologic community to research collision resistance of EnRUPT stream hashing. Unlike the P delta accumulators $d_0..d_{P-1}$, the $2\cdot h$-bit state of ïrRUPT is updated bijectively preventing local collisions until more than $2\cdot h$ bits of input is hashed. All the recently discovered collision attacks depend on local collisions. Local collisions occur in MD5 after 5 rounds, in SHA-1 after 6 rounds and in SHA-2 after 9 rounds. In ïrRUPT-128, -160, -192, -224, -256, -384 and -512, local collisions occur only after $2\cdot s\cdot$(H+P) rounds (that is 80, 96, 112, 80, 80, 112 and 144 rounds respectively).

4.4 First and second preimage resistance, $\min(2^{\mathbf{H}\cdot w-h},2^h)$

We believe the different kinds of preimage resistance to be identical to finding an encryption key for a block/stream cipher with conditions ranging from one known plaintext-cipher text pair to a large number of chosen plaintext-ciphertext pairs. Finding ïrRUPT preimages is no different to breaking it as a stream cipher.

In order to find a valid preimage to a given hash value, the attacker must first reverse the hash output process using recursive backtracking discussed in section **4.12** below. Although it is normally impossible to perform without the missing more than h bits of information about the state, the attacker may choose their own values and find any state that could produce the given hash value. If the attacker did not need to guess the missing $h+\mathrm{P}\cdot w$ bits of the final state, finding preimages of short messages would be trivial with a meet-in-the-middle attack.

In order to find a preimage for a final state with the guessed $h+\mathrm{P}\cdot w$ missing bits, $2\cdot s\cdot(\mathrm{H}+\mathrm{P})$ rounds of **(ïr1)** must be attacked, which is 112 for ïrRUPT-384, 128 for ïrRUPT-224 and 144 for ïrRUPT-256 and ïrRUPT-512. The cheapest generic attack is a meet-in-the-middle search, complexity of which is $2^{h+\mathrm{P}\cdot w/2}$, providing a $\mathrm{P}\cdot w/2$-bit security margin for possible optimisations. The search is further constrained by the difficulty of backtracking the ïrRUPT round function. Knowledge of the first preimage should not provide the attacker a significant amount of additional information that would facilitate the 2nd preimage search.

4.5 K-way second preimage resistance, $\min(2^{\mathrm{H}\cdot w/2},2^{h})$

Unless significant shortcuts are discovered, finding k-way second preimages for ïrRUPT has complexity of no less than $O(\log_2(k)\cdot 2^{h+\mathrm{P}\cdot w/2}+2^{h})$ operations [MULT] due to its large internal state of $2\cdot h+\mathrm{P}\cdot w$ bits.

4.6 K-way multicollision resistance, $\min(2^{\mathrm{H}\cdot w/2},2^{(k^{-}1)\cdot h/k})$

A k-way multicollision search must require at least $2^{(k^{-1})\cdot h/k}$ hashing operations for a hash to be considered secure. Iterative constructions with h-bit states (chaining variables) were found by A. Joux to require only $O(\log(k)\cdot 2^{h/2})$ operations for any k-way multicollision search [MULT], which is much lower than what it should be for $k>2$. The same attacks applied to ïrRUPT with its large internal state require at least $O(\log(k)\cdot 2^{h+\mathrm{P}\cdot w/2})$ operations. It is unreasonable to expect any minor optimisations of these attacks to reduce that complexity below $O(2^{h})$, which is higher than the required $\geq 2^{(k^{-1})\cdot h/k}$ complexity for any k.

4.7 CTFP preimage resistance, $\min(2^{\mathrm{H}\cdot w-h},2^{h})$

Due to its large internal state of $2\cdot h+\mathrm{P}\cdot w$ bits, unless significant shortcuts are discovered, building the diamond structure with $2^{k+1}-2$ states for ïrRUPT herding should take no less than $2^{k/2+h+\mathrm{P}\cdot w/2+2}$ hashing operations [HERD] and the chosen target forced prefix search should take $2^{2\cdot h+\mathrm{P}\cdot w-k}$ operations.

4.8 HMAC, PRF, AE, randomized hashing, $\min(2^{\mathrm{H}\cdot w/2},2^{h})$

Besides the security parameter s, security of EnRUPT in these modes depends on the sizes and randomness of the key, IV, salt and other randomization material.

4.9 Guess-and-determine, $\min(2^{\mathrm{H}\cdot w/2},2^{h})$

Unlike block ciphers or block hash functions, the structural integrity of stream ciphers and stream hash functions requires additional verification to ensure their resistance to guess-and-determine attacks that care very little about complexity of the functions. Guess-and-determine attacks work by guessing a few bits of the

state and determining whether there is a configuration of the remaining state bits that is consistent with the guessed bits and the observed output bits.

Our tool set includes such automated verification for algorithms with any structure. The structure of ïrRUPT has also been verified to ensure that at least a half of its state must be guessed for a state of any size before the attacker can gain any advantage by learning more information about the state than was guessed. Table **3** below demonstrates the number of words for different state sizes, which the attacker must guess before benefitting from it.

Words:	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
s=1	0	2	1	1	3	3	4	4	5	5	5	6	6	6	7	7	7	8	8	8	8	9	9	10	9	10	10	11	10	12	11
s=2	1	2	3	3	4	4	5	6	6	7	7	8	8	8	9	9	9	10	10	11	11	11	12	12	12	13	13	14	14	14	15
s=3	2	2	3	3	4	4	6	6	7	7	8	8	8	9	9	10	10	11	11	12	12	12	13	13	13	14	14	15	15	16	16
s=4	2	2	3	3	4	6	6	7	7	8	8	9	9	9	10	10	10	11	11	12	12	13	13	14	14	15	15	16	16	16	17

Table 3 ïrRUPT resistance to GD attacks (words to guess), P=2

4.10 Linearization, min($2^{H \cdot w-h}$,2^{h})

The most straightforward way to attack EnRUPT algorithms is to approximate each arithmetic addition operation with XOR. It results in the rest of the cipher turning into an easily solved system of linear equations. ïrRUPT processes each word $2 \cdot s$ times. It means that in order to define the state [disregarding the first and the last full cycles], at least $2 \cdot (s-1) \cdot H$ rounds must be attacked, containing more than $2 \cdot (s-1) \cdot H$ consecutive arithmetic additions to approximate. Therefore, in order to be able to break ïrRUPT faster than 2^h, the attacker must be able to approximate each arithmetic adder at a cost of less than $w/4 \cdot (s-1)$ bits, which is highly unlikely even for s=2. Only the non-cryptographic applications using s=1 could be successfully attacked in this way.

4.11 Partial state collisions, min($2^{H \cdot w/2}$,$2^{h/2}$,$2^{2 \cdot s \cdot P \cdot w}$)

Each full cycle propagates a change in one word of the internal state only directly to the nearest 2·P neighbours. Therefore a controlled or a guessed change in those 2·P directly neighbouring words could terminate further propagation to the rest of the internal state. Each full cycle adds 2·P to the number of words that need to be guessed or controlled.

Thus, for instance, in order to attack 8 full cycles of ïrRUPTx2, a change in one word would have to be contained by at least 32 words around that word with at least 16 more words available to verify with any degree of certainty that the match was not random. The complexity of this search is at most $2^{2 \cdot s \cdot P \cdot w}$ time with $2^{2 \cdot s \cdot P \cdot w}$ memory.

Although it is highly unlikely that this property could be exploited to attack the complete hash even with reduced s values, users of parallel modes wishing to combinine in a weak way (e.g. linearly) the internal states themselves instead of their compressed outputs, must be aware that the internal states greater than 6·s·P words may not provide the expected higher security.

4.12 Backtracking

While direct execution of ïrRUPT round function in reverse should have no effect on its security, it is not a trivial task. Since unlike the rest of the internal state,

the P delta accumulators $d_0..d_{P-1}$ are updated non-bijectively by the **(ïr1)** round function, reversing their operation requires execution of a recursive search process known as 'backtracking'.

Although it is reasonably inexpensive, the backtracking process contributes a noticeable cost to the attacker. While it may prevent some adaptive attacks that require direct access to the "decryption" process (such as boomerang attacks or meet-in-the-middle attacks) from succeeding, it will most likely merely increase their cost or shift the "middle" forcing them to attack a larger number of rounds at the benefit of a slightly increased number of possible solutions. The authors have published their ïrRUPT backtracking C source code on www.enrupt.com.

4.13 Timing and side channel attacks

It is becoming increasingly important for cryptographic algorithms to resist timing and side channel attacks. EnRUPT does not use variable rotations, S-boxes or other data-dependent or key-dependent memory operations to make its software implementations naturally invulnerable to timing attacks. Hardware implementations, however, should take all the necessary precautions to ensure their resistance to side-channel attacks. EnRUPT simplicity should make implementation of side-channel attack countermeasures easier than for other more complicated algorithms.

4.14 Parallel brute-force

The cheapest generic attacks are the parallel brute-force attacks. One ïrRUPT circuit is equivalent to about $(H+P+5)\cdot w$ bits of memory. ïrRUPT updates P words of the state per clock cycle. Thus only the attacks requiring fewer clock cycles than would do as many ïrRUPT circuits as can fit in the same area as the code, all the processors and all the memory required for the attack, can be considered faster or cheaper than the brute force. Please measure your attacks correctly.

5. Software Performance

Only the H state words, the P accumulator words and the round number must be stored between iterations, however performance can be improved by caching a number of words to be hashed with a speed-optimised implementation. Table **4** below provides memory usage (in bytes) and speed (in CPB) of the three included 32-bit and 64-bit implementations demonstrating some of the possible time / code size / implementation flexibility tradeoffs. Properly optimized SIMD assembly implementations should show even higher speeds.

	ïrRUPT-224			ïrRUPT-256			ïrRUPT-384			ïrRUPT-512		
	ref	avg	opt	ref	avg	opt	ref	avg	opt	ref	avg	opt
Unrolled, times	1	6	32	1	6	32	1	6	48	1	6	64
RAM, bytes	92	160	120	92	160	120	152	256	192	184	320	240
ROM, bytes	0	0	0	0	0	0	0	0	0	0	0	0
Speed, CPB	217	32	5.18	217	32	5.18	217	32	5.18	217	32	5.18

Table 4 Submitted 32-bit/64-bit software implementations

The following table lists 32-bit and 64-bit Core-2-Duo performances of three implementations: an optimal generic implementation that supports hashes of any

size, an unrolled C implementation and an unrolled ANSI C implementation of ïrRUPT32 with MMX and ïrRUPT64 with SSE compiler intrinsics.

s=4	Compiler	ïrRUPT32 generic	ïrRUPT32 MMX	ïrRUPT64 generic	ïrRUPT64 SSE
X86 on x64, CPB	IC32	13.3	13.4	20.2*	8.37
	MSC32	15	13.5	23–30.7*	9.09
	GCC32	17	14.4	24	9.87
x64, CPB	IC64	30	25.4	5.18	14.2
	MSC64	13.4	–	7.86	9.71
	GCC64	27.5	14.4	10.9*	9.87
Finalization, clock cycles		478–691	482–697	373–704	602–1138

Table 5 ïrRUPT Intel performance with different compilers on 16K messages
(*) u32-only implementation

The small code size of ïrRUPT also makes it an ideal candidate for the high-level language implementations with Java, JavaScript, Perl, Python, Basic, PHP, etc. significantly reducing the processing and download overheads for web pages. For comparison, the smallest implementation of MD5 in JavaScript to support secure password authentication in web pages occupies 7 kilobytes of code to download.

Platform	Hash	1 byte	10 bytes	100 bytes	1,000 bytes	10,000 bytes	100,000 bytes
Total Rounds	ïrRUPT-224	112	184	904	8112	80112	800112
	ïrRUPT-256	112	184	904	8112	80112	800112
	ïrRUPT-384	160	232	952	8160	80160	800160
	ïrRUPT-512	208	280	1000	8208	80208	800208
IC64 on x64, Plain C, Total Clock Cycles	ïrRUPT-224	718	76.3	12.1	5.81	5.21	5.14
	ïrRUPT-256	718	76.3	12.0	5.81	5.21	5.14
	ïrRUPT-384	955	100	14.5	6.09	5.23	5.15
	ïrRUPT-512	955	125	17.1	6.32	5.23	5.13
IC32 on x86, SSE intrinsics, Total Clock Cycles	ïrRUPT-224	1161	123	19.5	9.39	8.42	8.31
	ïrRUPT-256	1161	123	19.4	9.39	8.42	8.31
	ïrRUPT-384	1544	162	23.5	9.85	8.46	8.32
	ïrRUPT-512	1951	202	27.7	10.2	8.46	8.29

Table 6 Optimized ïrRUPT64x2/4 speed in C for messages of different sizes

5.1 Embedded 8-bit processor performance estimates

EnRUPT is designed to be smartcard-friendly with its tiny memoryless round function, a single **8-bit** or **16-bit** rotation and mostly XOR operations. Multiplication by 9 can be implemented as either a 3-bit shift across 32-64 bits and one 32-64-bit addition or as 4 calls to the adder implemented as 4-8 8-bit adders with carry. The rest of the cipher is trivially implemented with 8-bit XORs. Thus the total number of simple 8-bit operations to implement one round of ïrRUPT32 (64) is 50 (98). That is 21 (41) XORs, 8 (16) copy operations and either 21 (41) arithmetic additions or 16 (32) 1-bit shifts and 5 (9) arithmetic additions, with additional 4 (8) XOR operations once every 8 rounds to load each input word. That is 6 to 8 times faster than AES-128 and 10 times faster than SHA-1, with a much smaller code and with lower memory requirements as can be seen from Table **7** below:

Primitive	Code	RAM	ROM	CPB	Extra clock cycles / message
AES-128[6805]	1K	50	879	592 or 846	0 or 2278
ïrRUPT32-128	0.1K	44	0	101	3636 or 4040
SHA-1[6805]	2K	118	798	1058	0 or 67722
ïrRUPT32-160	0.1K	52	0	101	4444 or 4848
ïrRUPT32-192	0.1K	60	0	101	5252 or 5656
ïrRUPT64-224	0.2K	88	0	99	7128 or 7920
ïrRUPT64-256	0.2K	88	0	99	7128 or 7920
ïrRUPT64-384	0.2K	120	0	99	10296 or 11088
ïrRUPT64-512	0.2K	152	0	99	13464 or 14256

Table 7 8-bit processor performance comparison

Since 8-bit ïrRUPT32 implementations and 8-bit and 16-bit ïrRUPT64 implementations do not use bitwise rotations, we also anticipate significantly higher speeds on those processors comparing to other ADD-ROL-XOR designs.

6. Hardware Implementations

We evaluate several hardware efficient architectures of ïrRUPT stream hashing modes of EnRUPT for ASIC and FPGA target devices. The implementations described in Section **6.1** are meant for medium to high throughput applications. A minimum-size implementation is described in Section **6.2**.

6.1 High Efficiency Architectures

We simulated hardware implementations of ïrRUPT32-256 and ïrRUPT64-512 with the internal state of H=16 words and the security parameter s=4. We chose these two variants for closer comparability with the existing hash functions such as SHA-256 and SHA-512. The iterative mode of the round function **ïr2s** suggests a straightforward hardware implementation based on a round unit and a memory to store the internal state x and the delta accumulator d. For every parallelization degree, two distinct architectures have been investigated to evaluate the relation between speed and area.

6.1.1 Implementation

The 8-ïrRUPT designs are isomorphic architectures of the algorithm in which eight **(ïr1)** function units are instantiated. The sequential dependency between the words of x is exploited parallelizing the **(ïr1)** units according to the chosen parallelization parameter P. A full **ïr2s** round is computed in one clock cycle.

With P=1, the iterative decomposition of 8-ïrRUPT architecture leads to the implementation of only one **(ïr1)** unit. This architecture, referred to as 1-ïrRUPT, reuses the same hardware through eight cycles to compute a complete **ïr2s** round. The 2-ïrRUPT core is an efficient circuit for P=2, in which two parallel **(ïr1)** units operate on the state x in parallel to execute one **ïr2s** round in 4 clock cycles. The last architecture is the 4-ïrRUPT design. With P=4, four parallel **(ïr1)** units compute one **ïr2s** round every two clock cycles.

In the 8-ïrRUPT architectures, a new message word p_i is injected into the core on every cycle, while in the output phase every new d_{P-1} constitutes an element of the final hash value. In the three remaining P-ïrRUPT cores, p_i is injected into

the function every 2s/P clock cycles. Respectively, the final digest word o_i is generated every 2s/P clock cycles, in the output phase.

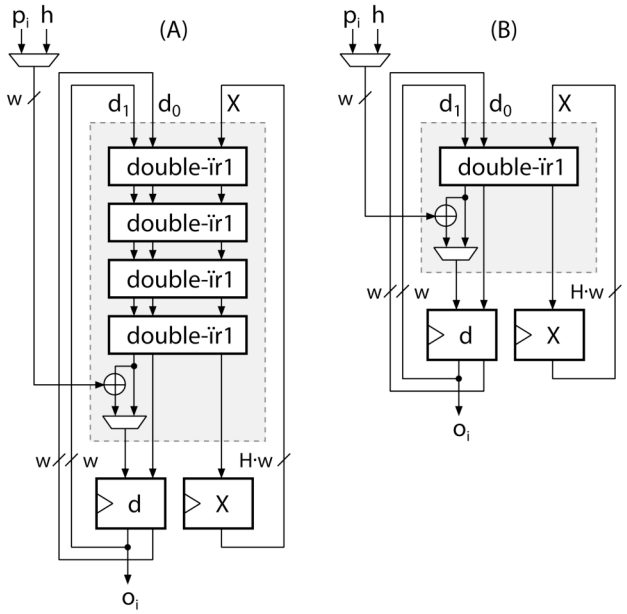

Fig. 9 8-ïrRUPT **(A)** and 2-ïrRUPT **(B)** architectures (P=2)

The block diagrams of the 8-ïrRUPT and 2-ïrRUPT architectures for P=2 are depicted in Figure **9**. The **double-ïr1** module hosts two parallel **(ïr1)** units.

The design strategy used in the implementation of the round function is based on a progressive word-shift of the state combined with P parallel **(ïr1)** units. Instead of using time-expensive multiplexers and demultiplexers, the **(ïr1)** modules take as inputs always the same x_i words to update x_{r+P} and d_r. Hence, according to the algorithm, every word is shifted by P words after the computation of one ore more parallel **(ïr1)** functions. This final word shift operation is equivalent to the increment of the index variable r. Figure **10** below shows the application of this strategy inside the **double-ïr1** module for the basic P=2 architectures.

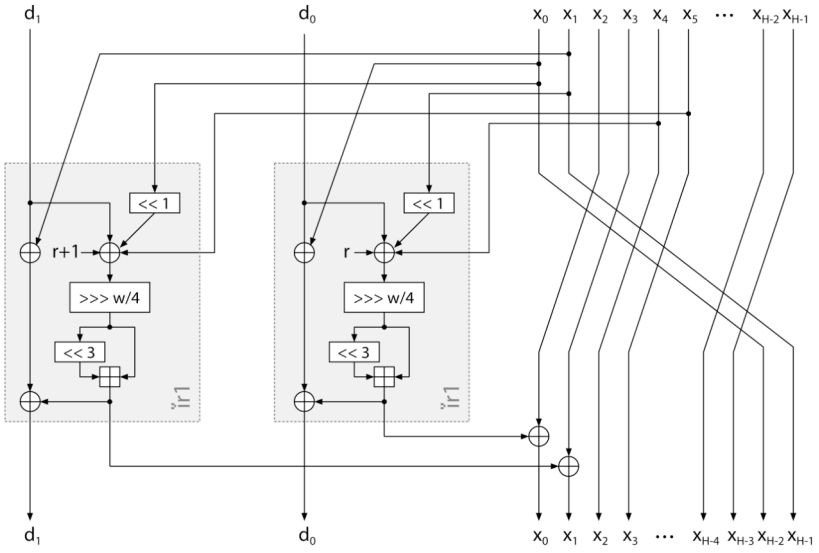

Fig. 10 Word-shift structure of the **double-ïr1** module
(all connections are w-bit wide)

Furthermore, all the explored architectures are controlled by a dedicated control unit, which computes the round number and routes the signal inside the cores. With the aid of enable signals, it switches between the message processing, the finalization, and the output phases. This unit relies on a finite state machine. Its contribution to the core area is estimated to be 400 to 800 GE depending on the ïrRUPT architecture.

6.1.2 Performance

Because of its stream hashing structure, the overall performance of the ïrRUPT hashing process depends on the length of the input message, while the processing time per word of input remains constant. Although the area and the operational frequency are constant characteristics of the circuit, the throughput varies with different message sizes. This is due to the additional clock cycles taken by the finalization and output phases. The throughput is computed as:

$$T = f \cdot \frac{m}{\frac{1}{k}\left(\left\lfloor \frac{m+w-1}{w} \right\rfloor + 1 + H + \left\lceil \frac{h-1}{w} \right\rceil\right)} \qquad (1),$$

where f is the circuit frequency and m is the message size. The constant k defines the iterative degree of the architecture. In 8-ïrRUPT, k is equal to one and in the P-ïrRUPT cores, k=P/(2s). Figure **11** below shows the m-dependency of the circuit's normalized throughput on frequency. Using long message sizes, the speed converges to $f \cdot k \cdot w$.

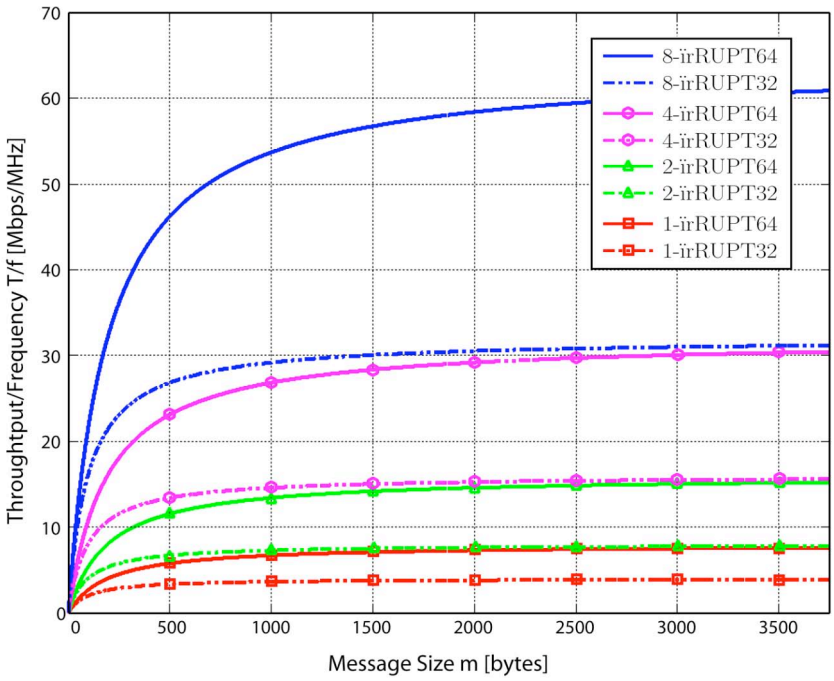

Fig. 11 Throughput/message size trade-off of the architectures.

The hashing cores have been coded in functional VHDL and synthesized with the Synopsys Compiler, using a 0.18 µm CMOS technology. The hardware analysis also includes the evaluation on a Xilinx Virtex-4 LX100 FPGA device. Table **8** and Table **9** below list the resource utilisation, the maximal clock frequency and the throughput of the architectures for ïrRUPT32 and ïrRUPT64. The efficiency

ratio gives a criterion to compare the hardware performances of the circuits. To illustrate the additional per-message latency, we provide the efficiency ratio for 16-word messages and for infinitely long messages. Figure **12** below depicts the relation between the area and processing time for 1Gb long messages.

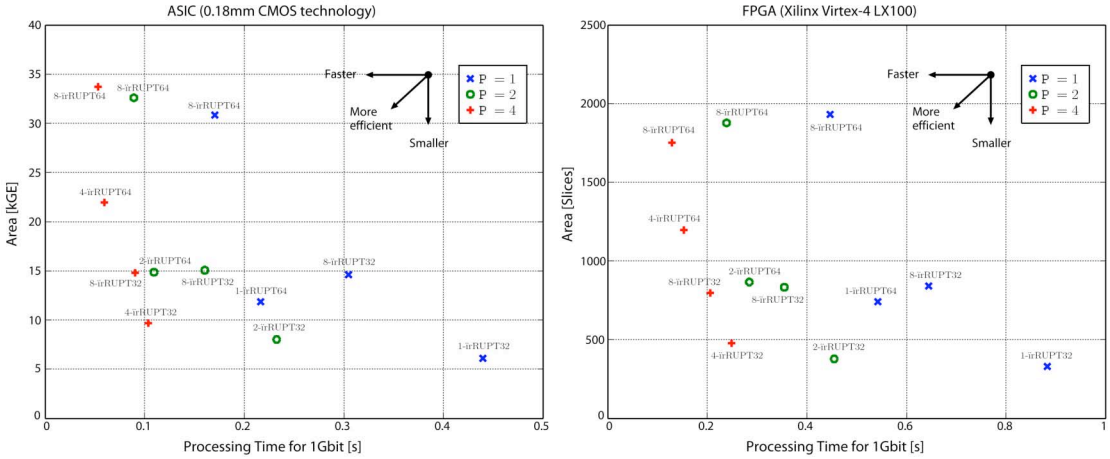

Fig. 12 Area vs. processing time for the ïrRUPT cores

6.1.3 Evaluation

As defined in Section **6.1.2** and seen on the Figure **11** above, the speed and the hardware efficiency of the circuits gradually increase with message size. The results illustrate the advantage of EnRUPT stream hashing. In ASIC and FPGA, the 8-ïrRUPT circuits are the fastest (especially for P=4) matching the needs of modern high-speed communication links, while the 1-ïrRUPT architecture is a more suitable alternative for the limited-resource applications.

Architecture		Area [kGE[1]]	Freq. [MHz]	Latency [Cycles]	Throughput [Gbps]		Efficiency [Kbps/GE[1]]	
					16w-bit m	Long m	16w-bit m	Long m
P=1	8-ïrRUPT32	14.70	103	m/w+25	1.281	3.282	87.1	223.3
	8-ïrRUPT64	30.89	92		2.287	5.861	74.0	189.7
	1-ïrRUPT32	6.12	568	8*(m/w)+200	0.887	2.273	145.0	371.6
	1-ïrRUPT64	11.91	578		1.805	4.624	151.5	388.2
P=2	8-ïrRUPT32	15.07	195	m/w+25	2.434	6.238	161.5	413.9
	8-ïrRUPT64	32.60	175		4.382	11.228	134.4	344.4
	2-ïrRUPT32	8.03	538	4*(m/w)+100	1.678	4.301	209.0	535.5
	2-ïrRUPT64	14.87	571		3.568	9.143	240.0	615.0
P=4	8-ïrRUPT32	14.79	346	m/w+25	4.321	11.073	292.1	748.4
	8-ïrRUPT64	33.73	292		7.303	18.713	216.5	554.8
	4-ïrRUPT32	9.73	602	2*(m/w)+50	3.761	9.639	386.6	990.6
	4-ïrRUPT64	21.97	526		6.573	16.842	299.1	766.4

Table 8 ASIC post synthesis results

[1] One GE corresponds to the area of a two-input drive-one NAND gate of size 9.7µm²

Architecture		Area [Slices]	Freq. [MHz]	Latency [Cycles]	Throughput [Gbps]		Efficiency [Mbps/Sl.]	
					16w-bit m	Long m	16w-bit m	Long m
P=1	8-ïrRUPT32	842	49	m/w+25	0.606	1.553	0.720	1.845
	8-ïrRUPT64	1930	35		0.877	2.248	0.455	1.165
	1-ïrRUPT32	343	283	8*(m/w)+200	0.442	1.133	1.289	3.304
	1-ïrRUPT64	739	230		0.719	1.843	0.973	2.494
P=2	8-ïrRUPT32	834	88	m/w+25	1.101	2.822	1.320	3.384
	8-ïrRUPT64	1878	65		1.635	4.188	0.870	2.230
	2-ïrRUPT32	379	275	4*(m/w)+100	0.858	2.198	2.263	5.799
	2-ïrRUPT64	867	220		1.372	3.516	1.583	4.055
P=4	8-ïrRUPT32	799	152	m/w+25	1.892	4.848	2.368	6.068
	8-ïrRUPT64	1754	121		3.025	7.752	1.725	4.420
	4-ïrRUPT32	479	251	2*(m/w)+50	1.565	4.010	3.267	8.372
	4-ïrRUPT64	1195	204		2.548	6.531	2.133	5.465

Table 9 FPGA post place & route results

Smaller designs synthesised with relaxed timing constraints can fit in much more compact areas. However, such solutions will have a lower global efficiency of the circuit, which is why their evaluation was omitted.

We have demonstrated the high degree of hardware flexibility of the ïrRUPT stream hashing mode. The proposed architectures are indeed able to reach a wide range of performance requirements and can be adapted to various environments.

6.2 Minimum size irRUPT MAC

We chose to implement one of the smaller irRUPT variants for use as a MAC, because of its specific application to RFID. Keyed one-way functions are needed on resource-constrained devices such as RFIDs for authentication or private identification [PRIV]. We find that irRUPT can be implemented smaller than any secure alternative that has been proposed so far for RFID tags such as AES. Our smallest implementation of irRUPT MAC uses 223 gate equivalents (GE), while the smallest known implementation of AES uses 3,800 GE [CAHF]. It is even smaller than some other stream ciphers that were specifically designed for RFID tags, but which were found to be insecure [MIF].

The small minimum size of a hardware implementation of irRUPT is made possible by (a) serializing all operations to work on single bit data, (b) realizing rotations and shifts (including multiplication) as delay elements, and (c) reusing one register to store all intermediate values.

The minimum size implementation of irRUPT32 round function for P=1 is shown on Figure **12** below. On each clock cycle, one bit of the current data word and its higher neighbour are loaded from memory. The lower neighbour is left in the internal state from the last round (the very first round only loads that data word). The higher neighbour is combined with the round counter, then delayed by one clock cycle and combined with the lower neighbour. The delay realizes the 1 bit shift between the neighbours. The data is then loaded in a 32-bit internal state. After the first 11 clock cycles (8 of which realize the rotation), an adder

consumes the data from two cells of the state (which realizes multiply by 9). The added values are then combined with the current data word and also loaded into the state. After 32 clock cycles, the state contains the new data word that is stored back to memory during the next round.

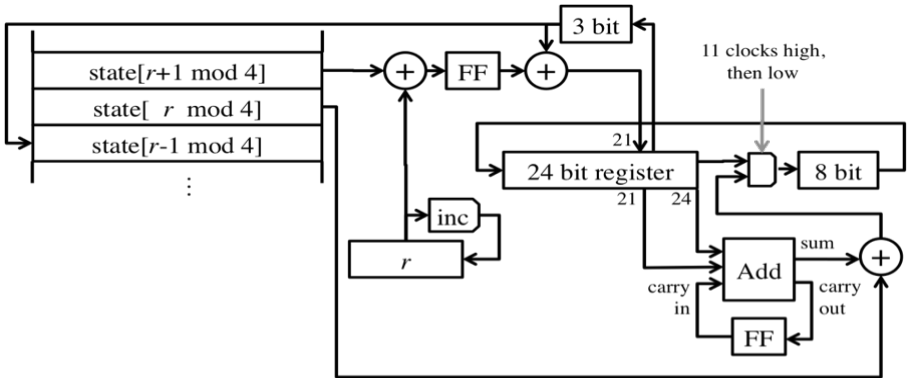

Fig. 12 Minimum size implementation of irRUPT32 round function, P=1.

Each execution of the round function takes 43 clock cycles (3 clock cycles for the data from the last round to propagate, 8 for rotation, and 32 to process the data). Assuming an RFID with 100 kHz, and a security level of 64 bits (64-bit key, 128-bit state, 32 sealing rounds), one complete MAC operation takes 17 ms.

The hardware implementation of the round function uses a total of 37 flip-flops (FF), 3 XOR, one adder, and one multiplexer. When implemented in the same technology that is used for some NXP RFID tags, the implementation uses 223 GE. The round counter needs another 5 FFs and adders and uses 58 GE. Therefore, an implementation of irRUPT can be more than an order of magnitude smaller than the smallest known implementation of the AES.

7. Probable Disadvantages

With two different word widths to support, EnRUPT implementations required to support hash values of all the different sizes are more complicated than we wished for. Actually, both ïrRUPT32 and ïrRUPT64 can be used for the SHA-3 hash sizes (224 to 512 bits). We chose ïrRUPT64 for its higher efficiency.

Different state sizes for different hash sizes may also be seen as a complication. Of course, ïrRUPT64 with H=16 (truncated ïrRUPT64-512) can be used to provide hash values of all the smaller sizes, but we chose the different state sizes for their higher hardware performance, especially in constrained environments.

ïrRUPTx4 is probably a better choice with its double hardware performance over the proposed ïrRUPTx2, but P=4 requires 16-word or larger states. That excludes ïrRUPT32-128, ïrRUPT32-160, ïrRUPT32-224 and ïrRUPT64-384, thus forcing the use of truncated ïrRUPT32-256 for SHA3-128, SHA3-160, SHA3-192 and SHA3-224, and either ïrRUPT32-384 or truncated ïrRUPT64-512 for SHA3-384. We leave it up to NIST to choose the most suitable variants.

EnRUPT's simplicity also makes it appear too simple to be a secure algorithm. It may hinder its public acceptance at first, but we hope that its resistance to cryptanalysis will build trust faster than for other more complex algorithms.

8. References

[XXT] R. Needham, D. Wheeler, *TEA extensions*, Technical Report 1997.

[ER007] S. O'Neil, *EnRUPT – First all-in-one symmetric cryptographic primitive*, SASC 2008.

[HAC] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Inc. 1997.

[TREE] R.C. Merkle, *A digital signature based on a conventional encryption function*, CRYPTO 1987

[XOR4] D.J. Bernstein, *What output size resists collisions in a xor of independent expansions?* http://cr.yp.to/rumba20/expandxor-20070503.pdf 2007

[MD6] C. Crutchfield, *Security Proofs for the MD6 Hash Function Mode of Operation*, MIT masters thesis 2008.

[LCM] A. Kerckhoffs, *La cryptographie militaire*, Journal des sciences militaires 1883.

[LR] M. Luby, C. Rackoff, *How to Construct Pseudorandom Permutations and Pseudorandom Functions*, SIAM Journal on Computing 1988.

[MODN] J. Kelsey, B. Schneier, D. Wagner, *Mod n Cryptanalysis, with Applications Against RC5P and M6*, FSE 1999.

[PRP] M. Naor, O. Reingold, *On the Construction of Pseudo-Random Permutations: Luby-Rackoff Revisited*, Journal of Cryptology 1999.

[BBSR] J. Patarin, A. Montreuil, *Benes and Butterfly Schemes Revisited*, ICISC 2005.

[DC] E. Biham, A. Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, Journal of Cryptology 1991.

[ALG] N. Courtois, W. Meier, *Algebraic Attacks on Stream Ciphers with Liner Feedback*, EuroCrypt 2003.

[SAAR] M.-J.O. Saarinen, *Chosen-IV Statistical Attacks on eSTREAM Stream Ciphers*, ECRYPT 2006.

[STAT] E. Filiol, *A New Statistical Testing for Symmetric Ciphers and Hash Functions*, ICISC 2002.

[SLID] A. Biryukov, D. Wagner, *Slide Attacks*, FSE 1999.

[CAHF] M. Feldhofer, C. Rechberger, *A case against currently used hash functions in RFID protocols*, Workshop on RFID Security 2006.

[ASD] S. O'Neil, *Algebraic Structure Defectoscopy*, Cryptology ePrint Archive 2007.

[UFN] B. Schneier, and Kelsey, J. *Unbalanced Feistel Networks and Block-Cipher Design*, FSE 1996.

[MIF] K. Nohl, D. Evans, Starbug, H. Plötz, *Reverse-Engineering a Cryptographic RFID Tag*, USENIX Security 2008.

[PRIV] D. Molnar, D. Wagner, *Privacy and Security in Library RFID: Issues, Practices, and Architectures*, ACM CCS 2004.

[MULT] A. Joux. *Multicollision on Iterated Hash Function*, Advances in Cryptology, CRYPTO 2004.

[6805] G. Keating, *Performance Analysis of AES candidates on the 6805 CPU core*, Proceedings of The Second AES Candidate Conference 1999.

[HERD] J. Kelsey, T. Kohno, *Herding Hash Functions and the Nostradamus Attack*, EUROCRYPT 2006.