Lightweight Cryptography for RFID Systems

Guang Gong

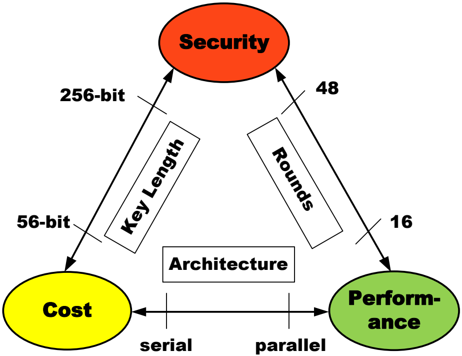
Department of Electrical and Computer Engineering
University of Waterloo
CANADA
<http://comsec.uwaterloo.ca/∼ggong>

Part II. Design of Lightweight Crypto primitives

- Overview of **lightweight** cryptographic primitives
- Design Principles
- Lightweight stream ciphers (Grain, Trivium and WG-7)
- Lightweight block ciphers (Present, KATAN/KTANTAN)
- **Hummingbird**: hybrid ultra-lightweight cryptographic algorithm

What is Lightweight Cryptography?

- Cryptography tailored to **(extremely) constrained devices**
- **Not** intended to **replace** classical cryptography
- Trade-off among **security**, **cost** and **performance**



Requirements of Security Solutions for RFID Tags

- Three **performance attributes**:
  - The **size** of an implementation, as measured by **gate equivalents (GE)**
  - The peak and the average **power consumption**
  - The **time** required to complete a computation
- What is available on a RFID tag?
  - Much depends on the security level, intended market, cost of fabrication and deployment, ...
  - A typical **rule-of-thumb** is that security features might occupy around **2,000 GE**.

Cryptographic Primitives

Symmetric-key Primitives	**Asymmetric-key Primitives**
Block ciphers	Encryption schemes
Stream ciphers	Digital signature schemes
Hash functions	Identification schemes
Message authentication codes	

- **Classical** cryptographic primitives designed for full-fledged computers might not be suited for resource-constrained RFID tags.
- **Designing** new lightweight cryptogrpahic primitives that can perform strong authentication and encryption for ultralow-power RFID applications is an emerging research area.

Lightweight Symmetric-Key Primitives

Block cipher	KATAN/KTANTAN Family [Cannière et al.'09], PRESENT [Rolfes et al.'08], DESXL [Leander et al.'07], AES [Feldhofer et al.'04]
Stream cipher	WG Family [Nawaz&Gong'08], Grain [Hell et al.'04], Trivium [Cannière&Preneel'06], Mickey [Babbage&Dodd'05]
Hybrid cipher	Hummingbird [Engels et al.'10]
Hash function and MAC	Quark [Aumasson et al.'10], PRESENT-based [Bogdanov et al.'08], AES-based

Lightweight stream ciphers

- **Feedback** Shift Registers (FSRs)
- Grain and Grain-like for a low bound of periods
- Trivium and Trivium-like generators
- WG generators

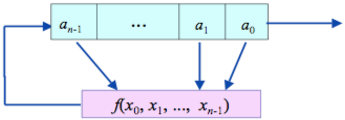
Feedback Shift Registers (FSRs)

Notation:

- $F = \mathbb{F}_2$, $K = \mathbb{F}_q = GF(q)$ where $q = p^m$ where p is a prime.
- **Boolean function:**

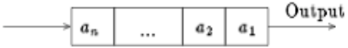
$$f(x_0, x_1, \cdots, x_{n-1}) = \sum c_{i_1 i_2 \cdots i_t} x_{i_1} x_{i_2} \cdots x_{i_t}, \; c_{i_1 i_2 \cdots i_t} \in \{0, 1\} \qquad (1)$$

where the sum runs through all subsets $\{i_1, \cdots, i_t\}$ of $\{0, 1, \cdots, n-1\}$.



A Block Diagram for an FSR

- An **initial state**: $(a_0, a_1, \cdots, a_{n-1})$.
- **State transition**: the next state of the shift register in the above FSR becomes



where

$$a_n = f(a_0, a_1, \cdots, a_{n-1}).$$

- **Output** sequence:

$$a_0, a_1, \cdots, a_n, \cdots .$$

- If the boolean function is nonlinear, then it is referred to as **nonlinear feedback shift register (NLFSR)** sequence.

Linear Feedback Shift Register (LFSR) Sequences

- **LFSR**: when the feedback function is a linear function

$$f(x_0, x_1, \cdots, x_{n-1}) = c_0x_0 + c_1x_1 + \cdots + c_{n-1}x_{n-1}, c_i \in F.$$

- ***m*-sequences**: If an LFSR sequence of n stage has the maximal period $2^n - 1$, then it is called a *maximal length sequence*, shortened as m-sequence, or pseudo noise sequence in communications.

- We associate a linear boolean function with a polynomial in the following fashion,

$$c_0x_0 + c_1x_1 + \cdots + c_{n-1}x_{n-1} \to x^n + c_{n-1}x^{n-1} + \cdots + c_1x + c_0 = f(x).$$

- The linear recursive sequence $\{a_i\}$ is an *m*-sequence if and only if $f(x)$ is primitive over F.

- **Example.** Let $n = 3$, $f(x_0, x_1, x_2) = x_0 + x_1$, with an initial state $(a_0, a_1, a_2) = (1, 0, 0)$, we have an *m*-sequences of periods 7 where the corresponding primitive polynomial is $f(x) = x^3 + x + 1$.

How do applications of stream cipher in practice lead us?

- It has been more than **forty years** without much progress for analysis of NLFSR sequences. Can we be stopped?
- The path:
  1. **Find some NLFSRs** with special feedback functions for which we hope to be able to determine periods and other unpredictable properties of the output sequences.
  2. **Use an LFSR** as an input to an NLFSR in order to have a large period of the output of the NLFSR.
  3. **Employ a large** number of internal states, i.e., stages of the registers, in hoping to obtain a large period of the output sequences.

Grain 2

- The stream cipher Grain is a **filtering sequence** for which a filtering function is applied to two FSRs where one is an LFSR and the other is an NLFSR.
- The filtering function is the sum of two functions where one is a **nonlinear** function in 5-variables which takes four taps from the LFSR and one from the NLFSR, and the other is a linear function which takes 7 taps from the NLFSR.
- In other words, the output is the sum of the output of the nonlinear function and the sum of the sequences from 7 tap positions at the NLFSR.
- **Both** LFSR and NLFSR have 80 stages.

Key Stream Generator

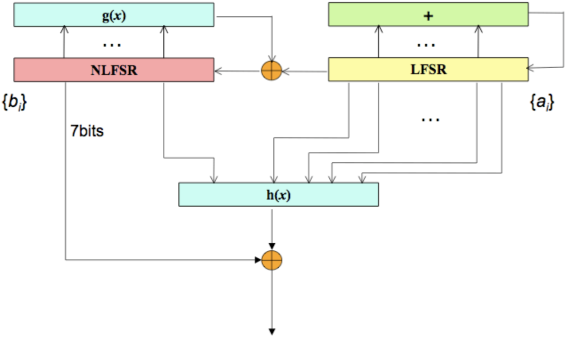


Figure: **A Diagram of Grain 2 Stream Cipher**

Construction

- Let $\mathbf{a} = \{a_i\}$ be the output of the LFSR with the characteristic polynomial $t(x)$

$$t(x) = x^{80} + x^{62} + +x^{51} + x^{38} + x^{23} + x^{13} + 1.$$

The linear recursive relation is given by

$$a_{i+80} = a_{i+62} + a_{i+51} + a_{i+38} + a_{i+23} + a_{i+13} + a_i, \; i \geq 0.$$

Construction (cont.)

- Let $\mathbf{b} = \{b_i\}$ be the output of the NLFSR with the following feedback boolean function $g(\underline{x})$ where $\underline{x} = (x_0, x_1, \cdots, x_{79})$.

$$
\begin{aligned}
& g(x_0, x_1, \cdots, x_{79}) = \\
& = x_{62} + x_{60} + x_{52} + x_{45} + x_{37} + x_{33} + x_{28} + x_{21} \\
& + x_{14} + x_9 + x_0 \\
& + x_{63}x_{60} + x_{37}x_{33} + x_{15}x_9 \\
& + x_{60}x_{52}x_{45} + x_{33}x_{28}x_{21} \\
& + x_{63}x_{45}x_{28}x_9 + + x_{60}x_{52}x_{37}x_{33} + x_{63}x_{60}x_{21}x_{15} \\
& + x_{63}x_{60}x_{52}x_{45}x_{37} + x_{33}x_{28}x_{21}x_{15}x_9 \\
& + x_{52}x_{45}x_{37}x_{33}x_{28}x_{21}.
\end{aligned}
$$

- The feedback bit is the masked by the output of the LFSR, i.e., the recursive relation is given by

Construction (cont.)

- The filtering function $f(x_0, x_1, \cdots, x_{11}) = h(x_0, x_1, x_2, x_3, x_4) + \sum_{j=5}^{11} x_j$ where $h(x)$ is given by

$$h(x_0, x_1, x_2, x_3, x_4) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4$$
$$+x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 + x_2x_3x_4.$$

- The tap positions are $(d_1, d_2, d_3, d_4, r_1, \cdots, r_8)$ where

$$d_1 = 3, d_2 = 25, d_3 = 46, \text{ and } d_4 = 64 \text{ from the LFSR}$$
$$r_1 = 63, \{r_i\}_{i=2}^{8} = \{1, 2, 4, 10, 31, 43, 56\} \text{ from the NLFSR.}$$

- Let $\mathbf{u} = \{u_i\}$ be the output of $h(x)$ whose elements are given by

$$u_i = h(a_{i+3}, a_{i+25}, a_{i+46}, a_{i+64}, b_{i+63}), i = 0, 1, \cdots .$$

- The output sequence of the generator, denoted by $\mathbf{s} = \{s_i\}$, is defined as

8

Key Initialization

- Before any keystream is generated the cipher must be initialized with the key and the IV, the initial vector.
- Let the key $K = (k_0, k_1, \cdots, k_{79})$ **(80-bits)** and the $IV = (IV_0, IV_1, \cdots, IV_{63})$ (64-bits).
- The initialization of the key is done as follows. First load the key K as an initial state of the **NLFSR** and load $T = IV||\underbrace{(1, \cdots, 1)}_{14}$ as an initial state of the LFSR where $x||y = (x_0, \cdots, x_{t-1}, y_0, \cdots, yr - 1)$, the concatenation of two vectors $x = (x_0, \cdots, x_{t-1})$ and $y = (y_0, \cdots, yr - 1)$.
- Then the generator is **clocked** 160 times without producing any running key. Instead the output function is fed back and xored with the input, both to the LFSR and to the NFSR, see the following figure.

Key Initialization

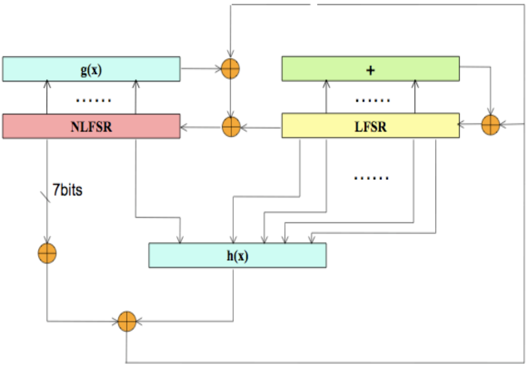


Figure: **Key Initialization of Grain 2**

Property of the Key Stream

- The boolean function g is **2-resilient** since x_0, x_{14}, x_{62} occur only in the linear terms.
- Filtering function is bent in 5-variables with **nonlinearity** 12 which is maximum.
- **Period** of the output sequence is at least $2^{80} - 1$.
- The exhaustive search space is 2^{80}.

Grain-like Generator: NLFSR masked by LFSR

- Let $\mathbf{a} = \{a_i\}$ be an output of the **LFSR** with a primitive polynomial $t(x) = \sum_{i=0}^{m} c_i x^i$ of degree m, i.e., the linear recursive relation is given by

$$a_{i+m} = c_{n-1} a_{i+m-1} + \cdots + c_1 a_{i+1} + c_0 a_i, i = 0, 1, \cdots .$$

- Let an **NLFSR** have n stages and the feedback boolean function $g(x_0, x_1, \cdots, x_{n-1})$.
- Let $\mathbf{b} = \{b_i\}$ be an output of the NLFSR, which is masked by the output of the LFSR, i.e., the recursive relation is given by

$$b_{i+n} = a_i + g(b_i, b_{i+1}, \cdots, b_{i+n-1}), i = 0, 1, \cdots .$$

We called **b** a ***Grain-like sequence***, denoted by $NLFSR(LFSR(m), n)$.

Remark

- **Grain generator** is the case where $n = m = 80$ and then a filtering function in five variables operates on **four taps** at LFSR and one at NLFSR which is further masked by the sum of 7 taps from NLFSR.

A Simple Known Result on Periods of Grain-like Generators

A Known Result:

An output of the NLFSR is at least $2^m - 1$. Furthermore it is multiple of $2^m - 1$ (Hu-Gong10).

Trivium and Trivium-like Generator

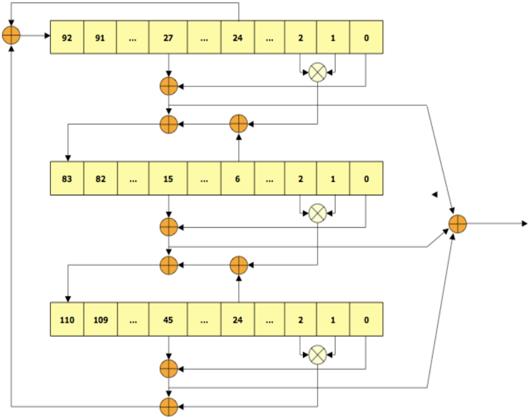


Figure: Trivium Generator (De Canniere-Preneel, 2005)

Trivium-like Generator (cont.)

- Let **three feedback shift registers** be FSR1 with l stages, FSR2 with m stages, and FSR3 with n stages.
- **Outputs** of FSR1, 2, and 3: $\{a_i\}$, $\{b_i\}$, and $\{c_i\}$.
- The output of the Trivium-like generator is given by

$$s_i = a_i + a_{i+k_1} + b_i + b_{i+k_2} + c_i + c_{i+k_3}, i = 0, 1, \cdots$$

  where k_js are nonzero constants with $k_1 < l, k_2 < m, k_3 < n$.
- The **update functions** of FSRs: let d_i be nonzero constants with $d_1 < l, d_2 < m, d_3 < n$, and

$$\begin{aligned} b_{i+m} &= f_1(a_i, \cdots, a_{i+l-1}) + b_{i+d_2}, i \geq m \\ c_{i+n} &= f_2(b_i, \cdots, b_{i+l-1}) + c_{i+d_3}, i \geq n \\ a_{i+l} &= f_3(c_i, \cdots, c_{i+l-1}) + a_{i+d_1}, i \geq l \end{aligned}$$

  where $f_i(x_0, \cdots, x_{v_i-1}) = x_0 + x_{k_i} + g_i(x_1, \cdots, x_{v_i-1})$ where $v_1 = l, v_2 = m, v_3 = n$ where g_is are quadratic functions.
- This generator is denoted as $FSR(l, m, n)$.

- For **Trivium key stream generator**, the parameters are given by

$l = 93$	$m = 84$	$n = 111$
$k_1 = 27$	$k_2 = 15$	$k_3 = 45$
$d_1 = 6$	$d_2 = 24$	$d_3 = 24$
$f_1(x_0, \cdots, x_{92}) = x_0+$	$f_2(x_0, \cdots, x_{83}) = x_0+$	$f_3(x_0, \cdots, x_{110}) = x_0+$
$+x_{27} + g_1(x_1, \cdots, x_{92})$	$+x_{15} + g_2(x_1, \cdots, x_{83})$	$+x_{45} + g_3(x_1, \cdots, x_{110})$
$g_1 = x_1 x_2$	$g_2 = x_1 x_2$	$g_3 = x_1 x_2$

Note that in Trivium g_i are the same, which is equal to $x_1 x_2$.

Period Property, Hu-Gong10

If the periods of the outputs of three FSRs are greater than $\max\{l, m, n\}$, then the outputs of three FSRs have the same periods.

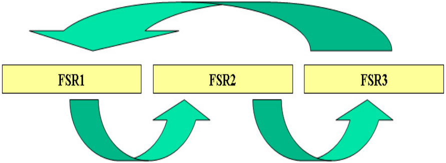


Figure: The control mode in three FSRs

WG Stream Cipher Family and Its Lightweight Variant WG-7

Outline of WG Family [Nawaz-Gong, 08]

- Synchronous stream cipher
- Based on **Welch-Gong (WG) transformation** sequences, well studied in sequence design for communications
- The keystream **possesses** the cryptographic properties of WG sequences
- **WG can output** multiple bits instead of one bit, named as **MOWG** [Lam-Aagaard-Gong, 09].

WG (MOWG) Cipher

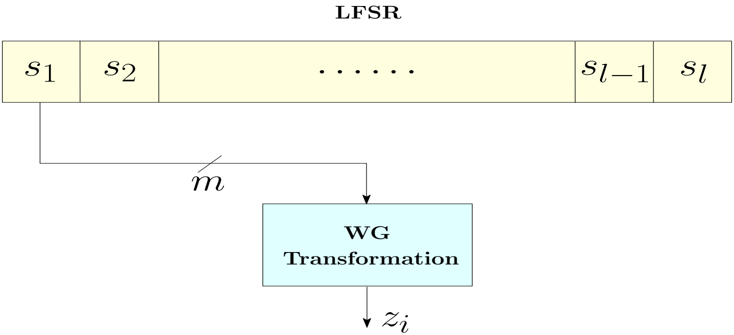
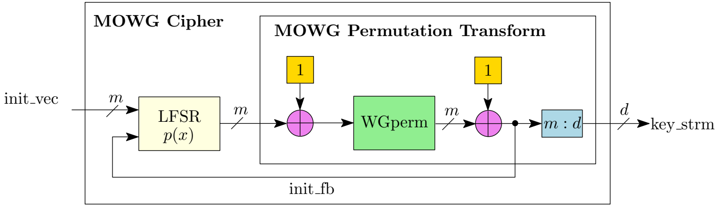


Figure: **A Diagram of WG (MOWG) Generator**

- LFSR generates an *m*-sequence over $GF(2^m)$ of degree *l*.
- The elements of *m*-sequence are filtered by a WG transform: $GF(2^m) \rightarrow GF(2^m)$, to produce a *d*-bit keystream sequence where $d < m/2$.
- For $d = 1$, this is the original WG cipher.

Description of WG (MOWG) Cipher



$$\mathrm{MOWG}(x) = \mathrm{WGperm}(x+1) + 1$$

$$\mathrm{WGperm}(x) = x + x^{r_1} + x^{r_2} + x^{r_3} + x^{r_4}, \quad \text{where } x \in \mathbb{F}_{2^m}$$

Mathematical parameters:

- m – Bit-width of cipher
- $g(x)$ – Generating polynomial for $\mathbb{F}_{2^m}$
- $p(x)$ – Primitive polynomial for LFSR
- l – Degree of $p(x)$

Find k such that $3k \equiv 1 \pmod{m}$

- $r_1 = 2^k + 1$
- $r_2 = 2^{2k} + 2^k + 1$
- $r_3 = 2^{2k} - 2^k + 1$
- $r_4 = 2^{2k} + 2^k - 1$

Hardware Architecture of WG (WOMG)

Simplifications:

- $2^{2k} + 2^k = (2^k + 1) \times 2^k$
- $2^{2k} - 2^k = (2^k - 1) \times 2^k$

WGperm exponents:

- $s = 2^k - 1$
- $r_1 = 2^k + 1$
- $r_2 = 2^{2k} + r_1$
- $r_3 = s \times 2^k + 1$
- $r_4 = 2^{2k} + s$

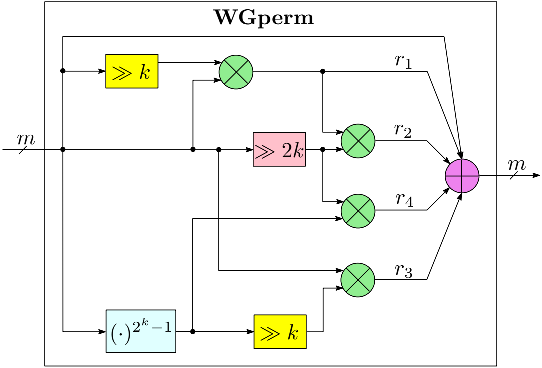


Figure: **Multiplier-based implementation of WGperm**

Key Initialization and Re-synchronization

- **Key initialization** for WG and MOWG are the same.
- **We denote** the state of the LFSR as $S_0, \cdots, S_{l-1}$ where $S_i \in \mathbb{F}_{2^m}$, an m-bit vector, secret key bits, $(k_0, \cdots, k_{r-1})$, and initial vector, $IV = (IV_0, \cdots, IV_{K-1})$ where $k_i, IV_i \in \mathbb{F}_2$ and r is the size of the secret key.
- **Once** the LFSR has been loaded with the key and IV (in some specified procedure), the keystream generator is run for $2l + 1$ clock cycles, after which the outputs of WGperm may be used as the keystream.

Example WG (MOWG) ciphers and security levels

m	l	n	$LC_{WGK} \geq$	C_{AA}	ONB
7	23	161	$2^{30.02}$	$2^{66.11}$	$*$
11	15	165	$2^{23.36}$	$2^{80.57}$	$\surd$
11	16	176	$2^{24.7}$	$2^{81.7}$	$\surd$
11	24	264	$2^{27.6}$	2^{90}	$\surd$
11	32	352	$2^{29.6}$	$2^{95.8}$	$\surd$
29	11	319	2^{45}	2^{182}	$\surd$

*	look-up table with no finite field computation
n	Size of the internal state (or LFSR) in bits
LC_{WGK}	Linear complexity of the cipher
C_{AA}	Complexity of an algebraic attack on the cipher
ONB	$\surd$ denotes that optimal normal bases exist for using finite field multipliers

Randomness of the Cipher

- **Randomness** Properties of Keystream
  - Period is $2^n - 1$
  - Balanced
  - 2-level autocorrelation
  - Ideal t-tupledistribution $(1 \leq t \leq l)$
  - Linear complexity $\geq ml^{m/3}$ or $\geq ml^{m-1}$ for look-up table
- **Cryptographic** Properties of WG Transformation
  - Algebraic degree $m/3$
  - Nonlinearity is maximized
  - Additive autocorrelation between $WG(x + a)$ and $WG(x)$ has three values.

Security against Known Attacks

- **Time/Memory/Data** Tradeoff Attacks: size of internal state is 2^n.
- **Algebraic Attacks**: the number of linear equations $\approx \begin{pmatrix} n \\ l \end{pmatrix}$; and the attack complexity is very high.
- **Correlation** Attacks: WG transformation has maximized nonlinearity.
- Resist to various known attacks.

Features of WG (MOWG) Stream Ciphers

- **Guaranteed** key stream randomness properties
- **Secure** against time/memory/data tradeoff attacks, algebraic and correlation attacks
- **Provide** a wide spectrum of possible levels of security, with corresponding increases in area and decreases in optimality in hardware implementation as the security increases.
- Can be implemented in **hardware** with low complexity for some parameters.

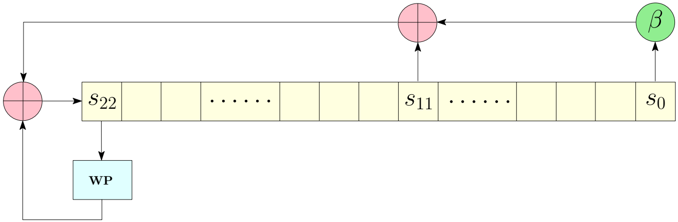
Impact of Good Correlation Property

- **The use** of the WG or MOWG steam cipher fulfills multiple security requirements, especially when it is applied for encryption and authentication in RFID systems.
- Usually, there is no **tamper resistant** module in low cost RFID systems. However, the ideal 2-level autocorrelation of WG cipher and the low autocorrelation property possessed by MOWG cipher provide tamper resistance for RFID tags.
- The WG cipher also can prevent **side-channel** attacks since the power spectral density of a WG keystream sequence is flat.
- **Another benefit** of good correlation of WG or MOWG keystream sequences is for reader anti collision in RFID systems at signal transmission level which is obtained for free, i.e., the reader collision is prevented since different readers sends different ciphertext for which the keystream sequences having low correlation, an analogue to code division multiple access (CDMA) technologies.

Description of Lightweight Stream Cipher WG-7

- Defining polynomial of $\mathbb{F}_{2^7}$: $g(x) = x^7 + x + 1$
- Characteristic polynomial of the LFSR: $f(x) = x^{23} + x^{11} + \beta$, where $g(\beta) = 0$
- **WG Permutation (WP)**: $WG_{perm}(x) = t(x^3)$, where $t(x) = h(x+1) + 1$ and $h(x) = x + x^{33} + x^{39} + x^{41} + x^{104}$
- **WG Transform (WT)**: $WG_{trans}(x) = Tr(t(x^3)) = Tr(x^3 + x^9 + x^{21} + x^{57} + x^{87})$

WG-7: Initialization

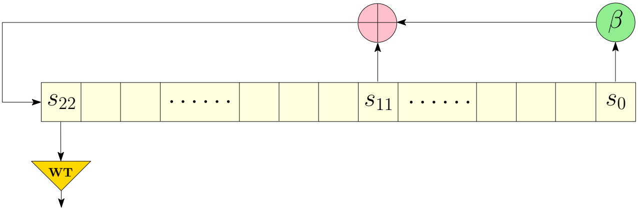


1. Load IV and Key as the initial state of the LFSR: for $0 \leq i \leq 10$

$$
\begin{aligned}
S_{2i} &= K_{7i}K_{7i+1}K_{7i+2}K_{7i+3}IV_{7i}IV_{7i+1}IV_{7i+2} \\
S_{2i+1} &= K_{7i+4}K_{7i+5}K_{7i+6}IV_{7i+3}IV_{7i+4}IV_{7i+5}IV_{7i+6} \\
S_{22} &= K_{77}K_{78}K_{79}IV_{77}IV_{78}IV_{79}IV_{80}
\end{aligned}
$$

2. Run for 46 clock cycles with the output added to the feedback of the LFSR which is then used to update the LFSR

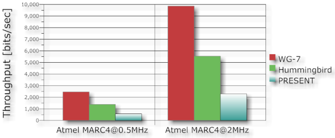
WG-7: Key Stream Generation & Security Properties

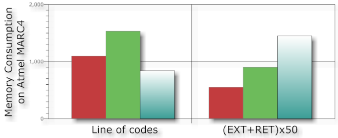


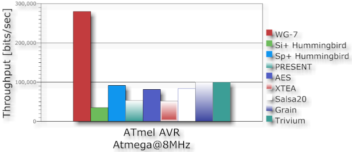
Verified security properties of WG-7

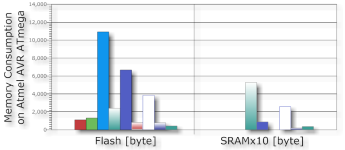
- The long period, i.e., $2^{161} - 1$
- The balance property
- First order resiliency property of WG_{trans}
- Ideal two-level autocorrelation property
- Acceptable linear complexity $\approx 2^{25.5}$

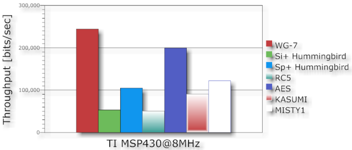
Software Implementation on Microcontrollers

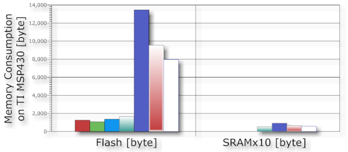












Lightweight Block Ciphers

Algorithm	Reference	key size	block size	datapath width	cycles/ block	Throughput [Kbps]	Logic [μm]	Area [GE]
KATAN-32	[Cannière et al.'09]	80	32	–	256	12.5	0.13	802
KATAN-64		80	64	–	255	25.1	0.13	1,027
PRESENT-80	[Rolfes et al.'08]	80	64	4	547	11.7	0.18	1,075
PRESENT-128		128	64	4	559	11.45	0.18	1,391
DES	[Leander et al.'07]	56	64	4	144	44.4	0.18	2,309
DESXL		184	64	4	144	44.4	0.18	2,168
AES-128	[Feldhofer et al.'04]	128	128	8	1,032	12.4	0.35	3,400
AES-128	[Hämäläinen et al.'06]	128	128	8	160	44.4	0.13	3,100
HIGHT	[Hong et al.'06]	128	64	64	1	6,400	0.25	3,048
mCrypton	[Lim et al.'05]	96	64	64	13	492.3	0.13	2,681
SEA	[Standaert et al.'06]	96	96	96	93	103.23	0.13	3,758

Table: The implementation results for a variety of **hardware-oriented block ciphers**. The throughput is measured when clocked at a typical RFID tag frequency of 100 KHz.

Ultra-Lightweight Block Cipher: PRESENT

- Simple substitution-permutation network (**SPN**)
  - 64-bit block size
  - 80-bit key size (optionally but not recommended 128-bit)
  - 31 rounds
- 16 4×4 S-boxes (16 copies of the **same** S-box!)
- Simple bit permutation, **no linear layers**

generateRoundKeys()

for $i = 1$ to 31 **do**

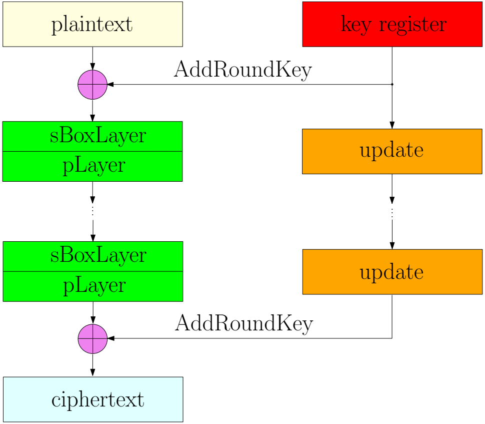
AddRoundKey(STATE, K_i)

sBoxLayer(STATE)

pLayer(STATE)

end for

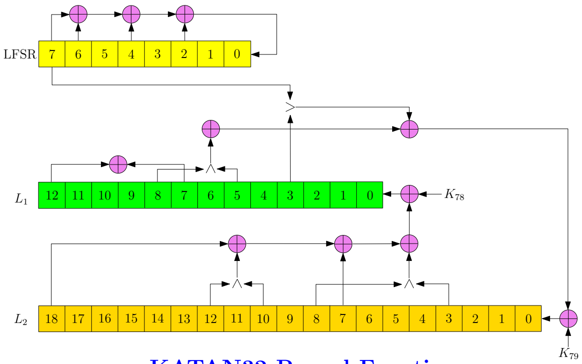
AddRoundKey(STATE, K_{32})



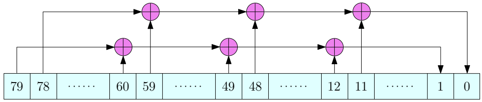
The KATAN/KTANTAN Block Ciphers

- **Bivium** (**Trivium** with two registers) in a block cipher mode
  - 32/48/64-bit block size
  - 80-bit key size
  - 254 rounds
- KATAN and KTANTAN are the same up to the **key schedule**
- The key is **fixed** and **cannot** be changed in KTANTAN.
- **LFSR** counts rounds (rather than a counter)
- **Two** round functions (one of them is controlled by a bit of the LFSR)

Specification of KATAN32



KATAN32 Round Function



KATAN32 Key Schedule

Lightweight Hash Functions

Hash Functions	Reference	output size	datapath width	cycles/ block	Throughput [Kbps]	Logic [μm]	Area [GE]
U-QUARK	[Aumasson et al.'10]	128	–	33	242.42	0.18	$1,379$
D-QUARK		160	–	547	14.63	0.18	$1,702$
T-QUARK		224	–	33	387.88	0.18	$2,296$
PRESENT80-based	[Bogdanov et al.'08]	64	64	33	242.42	0.18	$2,213$
			4	547	14.63	0.18	$1,600$
PRESENT128-based	[Bogdanov et al.'08]	128	64	33	387.88	0.18	$2,530$
			4	559	22.9	0.18	$1,886$
AES128-based	[Bogdanov et al.'08]	128	8	$> 1,032$	< 12.4	estimiate	$> 4,400$

Table: The implementation results for **hardware-oriented hash functions**. The throughput is measured when clocked at a typical RFID tag frequency of 100 KHz.

- Standard hash functions such as MD5 and SHA-1 are not suitable for RFID tags due to their large hardware footprint ($7,000$**+GE**).

Message Authentication Codes (MACs)

- MACs can be constructed from a **lightweight block cipher** (e.g., PRESENT) in an appropriate operation mode.
- MACs can also be built from a **secure hash function**. However, this brings us back to the underlying problems with finding a lightweight hash function.
- It is still not clear how to construct MACs from a **lightweight stream cipher**.

Hummingbird (Engels-Fan-Gong-Hu-Smith09)

Basic Idea of Hummingbird Design

- Enigma belongs to a group of **rotor-based** crypto machines.
- The number of possible internal connections of Enigma has been estimated to be approximately $3 \cdot 10^{114}$.
- The basic idea of Hummingbird cipher lies in implementing **extraordinarily large virtual rotors** with custom **block ciphers**.

Figure: Enigma Machine

Hummingbird in a Nutshell

- Hummingbird is a **rotor machine** and has a **hybrid structure** of block cipher and stream cipher.
    - 16-bit block size
    - 256-bit key size
    - 80-bit internal state
- 4 **identical block ciphers** with 16-bit input and 16-bit output
- 4 16-bit registers acting as 4 **rotors**
- A 16-bit linear feedback shift register (**LFSR**)
- Simple arithmetic and logic operations ($\oplus, \boxplus, \boxminus, \lll$)

Hummingbird Initialization Process

Nonce Initialization:

1. $RS1_0 = \text{NONCE}_1$
2. $RS2_0 = \text{NONCE}_2$
3. $RS3_0 = \text{NONCE}_3$
4. $RS4_0 = \text{NONCE}_4$

Four Iterations:

5. for $t = 0$ to 3 do

$$V12_t = E_{k_1}((RS1_t \boxplus RS3_t) \boxplus RS1_t)$$
$$V23_t = E_{k_2}(V12_t \boxplus RS2_t)$$
$$V34_t = E_{k_3}(V23_t \boxplus RS3_t)$$
$$TV_t = E_{k_4}(V34_t \boxplus RS4_t)$$
$$RS1_{t+1} = RS1_t \boxplus TV_t$$
$$RS2_{t+1} = RS2_t \boxplus V12_t$$
$$RS3_{t+1} = RS3_t \boxplus V23_t$$
$$RS4_{t+1} = RS4_t \boxplus V34_t$$

6. end for

LFSR Initialization:

7. LFSR = $TV_3 \mid 0\text{x}1000$

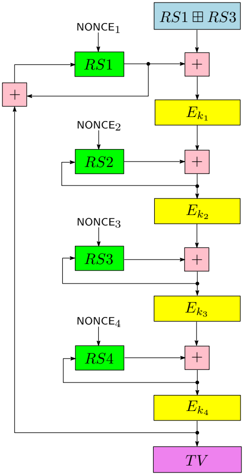


Figure: Initialization Process

Hummingbird Encryption Process

Block Encryption:

1. $V12_t = E_{k_1}(PT_i \boxplus RS1_t)$
2. $V23_t = E_{k_2}(V12_t \boxplus RS2_t)$
3. $V34_t = E_{k_3}(V23_t \boxplus RS3_t)$
4. $CT_i = E_{k_4}(V34_t \boxplus RS4_t)$

Internal State Updating:

5. $\mathrm{LFSR}_{t+1} \leftarrow \mathrm{LFSR}_t$
6. $RS1_{t+1} = RS1_t \boxplus V34_t$
7. $RS3_{t+1} = RS3_t \boxplus V23_t \boxplus \mathrm{LFSR}_{t+1}$
8. $RS4_{t+1} = RS4_t \boxplus V12_t \boxplus RS1_{t+1}$
9. $RS2_{t+1} = RS2_t \boxplus V12_t \boxplus RS4_{t+1}$

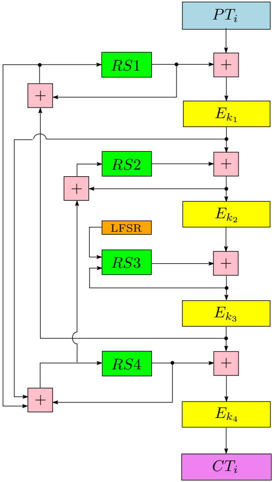


Figure: Encryption Process

Top-Level Specification of 16-bit Block Cipher

- Simple substitution-permutation network (SPN)
- 64-bit key size
- 16-bit block size
- Four 4 S-boxes (can use **only one** and repeat four times!)
- Simple linear transform for permutation layer
- 5 rounds (4 regular rounds + 1 final round)

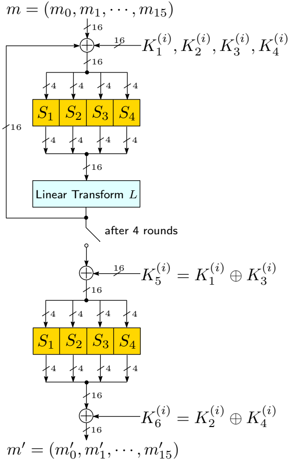


Figure: 16-bit Block Cipher

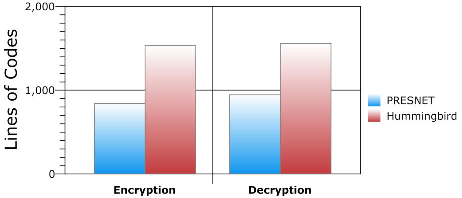
Security Analysis of Hummingbird Cryptographic Algorithm

- Differential Cryptanalysis
- Linear Cryptanalysis
- Structure Attack
- Algebraic Attack
- Cube Attack
- Slide and Related-key Attack
- Interpolation and Higher Order Differential Attack
- ......

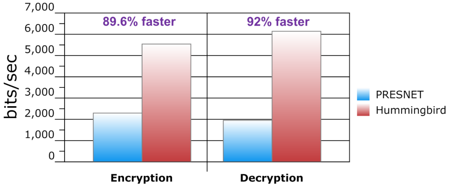
Software Implementation on Low-Power Microcontrollers

- The **compact version** of Hummingbird is implemented.
  - A **single** 4×4 S-box S_1 is used four times in the 16-bit block cipher
- Three popular **low-power** microcontrollers for embedded applications:
  - 4-bit microcontroller ATAM893-D from Atmel
  - 8-bit microcontroller ATmega128L from Atmel
  - 16-bit microcontroller MSP430 from Texas Instrument (TI)
- Two implementation variants:
  - **Size** optimized implementation (8- and 16-bit platforms)
  - **Speed** optimized implementation (4-, 8- and 16-bit platforms)

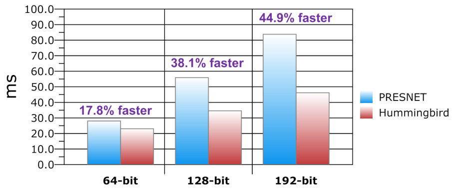
Software Implementation on 4-bit Microcontrollers



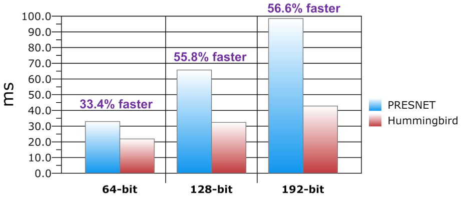
Memory Consumption



Throughput @ 2MHz



Overall Encryption Performance @ 2MHz



Overall Decryption Performance @ 2MHz

Software Implementation on 8- and 16-bit Microcontrollers

- **Hummingbird** vs. PRESENT on a 8-bit microcontroller ATmega128L
  - 13% less memory and 40 times faster throughput (**size**-optimized)
  - 78% more memory and 71.3% faster throughput (**speed**-optimized)
- **Hummingbird** vs. PRESENT on a 16-bit microcontroller MSP430
  - 69% less memory and 148 times faster throughput (**size**-optimized)
  - 96% less memory and 4.7 times faster throughput (**speed**-optimized)

MICAz mote equipped with Atmega128L microcontroller

TelosB mote equipped with MSP430 microcontroller

Performance Comparison of FPGA Implementations

Cipher	Key Size	Block Size	FPGA Device	Total Occupied Slices	Max. Freq. (MHz)	Throughput (Mbps)	Efficiency (Mbps/# Slices)
Hummingbird	**256**	**16**	**Spartan-3 XC3S200-5**	**273**	**40.1**	**160.4**	**0.59**
PRESENT [Poschmann'09]	80	64	Spartan-3 XC3S400-5	176	258	516	2.93
	128	64		202	254	508	2.51
PRESENT [Guo et al.'08]	80	64	Spartan-3E XC3S500	271	–	–	–
XTEA [Kaps'08]	128	64	Spartan-3 XC3S50-5	254	62.6	36	0.14
			Virtex-5 XC5VLX85-3	9,647	332.2	20,645	2.14
ICEBERG [Standaert et al.'08]	128	64	Virtex-2	631	–	1,016	1.61
SEA [Mace et al.'08]	126	126	Virtex-2 XC2V4000	424	145	156	0.368
AES [Chodowiec & Gaj'03]	128	128	Spartan-2 XC2S30-6	522	60	166	0.32
AES [Good & Benaissa'05]			Spartan-3 XC3S2000-5	17,425	196.1	25,107	1.44
			Spartan-2 XC2S15-6	264	67	2.2	0.01
AES [Rouvroy et al.'04]			Spartan-2 XC2V40-6	1,214	123	358	0.29
AES [Bulens et al.'08]			Spartan-3	1,800	150	1700	0.9

- **Hummingbird** can achieve **larger throughput** with **smaller area** requirement, when compared to block ciphers XTEA, ICEBERG, SEA and AES.

Concluding Remarks (1/2)

- **(Passive) RFID tags** have **extremely constrained resources**:
  - Small area
  - Low power
  - Low energy
  - Short message
- Lightweight cryptographic primitives should...
  - Have a **short internal state** (to lower area)
  - Allow **serial hardware implementation** (to lower power)
  - Have a **short processing time** (to lower energy)
  - Have a **short output** (to lower communication overhead)
  - Have **multiple functionalities** (e.g., PRF, MAC, PRNG, etc.)

Concluding Remarks (2/2)

- **Lightweight cryptographic primitives** are crucial for RFID security.
- The **key issue** of designing lightweight cryptographic primitives is to deal with the trade-off among **security**, **cost**, and **performance**.
- **WG (MOWG)** family, with **guaranteed randomness properties**, provides a wide spectrum of possible levels of trade-off between security and area and optimality in hardware implementation.

References

Y. Nawaz and G. Gong.
WG: A family of stream ciphers with designed randomness properties.
Information Sciences, Vol. 178, No. 7, April 1, 2008, pp. 1903-1916.

C.H. Lam, M. Aagaard, and G. Gong.
Hardware Implementations of multi-output Welch-Gong ciphers.
Preprint, University of Waterloo, 2009.
(The hardware implementation of 29-bit WG, submitted to ESTREAM by Nawaz and Gong in 2005, is reported in the Master Thesis of C.H. Lam at University of Waterloo, 2008.)

D. Engels, X. Fan, G. Gong, H. Hu, and E. Smith.
Hummingbird: Ultra-lightweight Cryptography for Resource-Constrained Devices.
Financial Cryptography and Data Security, FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, January 25-28, 2010, Tenerife, Canary Islands, Spain.

X. Fan, H. Hu, G. Gong, E. Smith, and D. Engels.
Lightweight Implementation of Hummingbird Cryptographic Algorithm on 4-Bit Microcontroller.
The 1st International Workshop on RFID Security and Cryptography 2009


SERIAL ATA
RFID
RED
GREEN
HI-SPEED
RFID-SATA SOLUTION
2.5" SECURITY HD ENCLOSURE

Questions?