On Lightweight Stream Ciphers with Shorter Internal States

Frederik Armknecht, Vasily Mikhailov
University of Mannheim

January 13, 2015
Overview

- Motivation
- Design Approach
- Stream Cipher Proposal Sprout
- Conclusions
Motivation
Why Considering Lightweight Crypto?

- Some reasons:
  - Relevant for industry (see next slide)
  - Interesting (see next but one slide)
  - Open questions (remainder of the talk)
Lightweight Crypto is Relevant

What to do next? Seriously...

- Fast and secure: Let’s call it done!
- Three remaining axes:
  - Small box: Lightweight crypto.
  - White-box: Software security.

Slide by Michaël Peeters (NXP)
Lightweight Crypto is Interesting

„Ronnie, what is the most interesting topic in modern cryptography?“
Some Lightweight Ciphers

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Area size (GE)</th>
<th>Throughput (Kb/s)*</th>
<th>Logic process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block ciphers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRESENT 80</td>
<td>1570</td>
<td>200</td>
<td>0.18 μm</td>
</tr>
<tr>
<td>PRESENT 80</td>
<td>1000</td>
<td>11.4</td>
<td>0.35 μm</td>
</tr>
<tr>
<td>KATAN32</td>
<td>802</td>
<td>12.5</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KATAN48</td>
<td>927</td>
<td>18.8</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KATAN64</td>
<td>1054</td>
<td>25.1</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KTANTAN32</td>
<td>462</td>
<td>12.5</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KTANTAN48</td>
<td>588</td>
<td>18.8</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KTANTAN64</td>
<td>688</td>
<td>25.1</td>
<td>0.13 μm</td>
</tr>
<tr>
<td><strong>Stream ciphers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mickey</td>
<td>3188</td>
<td>100</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>Trivium</td>
<td>2580</td>
<td>100</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>Grain 80</td>
<td>1294</td>
<td>100</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>Grain 80</td>
<td>1180</td>
<td>100</td>
<td>0.18 μm</td>
</tr>
</tbody>
</table>

*- The throughput is given for the clock frequency of 100 KHz*
Question

Is it possible to construct smaller stream ciphers which still provide the same level of security?
Why the Difference?

MIND THE GAP

Blockciphers

Streamciphers

Area Size

Rule of thumb for streamcipher designs:
Internal state size at least twice the security parameter

Realizing registers extremely costly
Keystream Generators

Initialization phase:

Initialization Vector → $Init$ → $State$

Key space $K = GF(2^K)$
State space $S = GF(2^\sigma)$
IV space $IV = GF(2^u)$

Keystream generation phase:

$State$ → $Upd$ → $Out$ → Keystream ($z_t$)

Initialization function $Init: K \times IV \rightarrow S$
Update function $Upd: S \rightarrow S$
Output function $Out: S \rightarrow GF(2)$
Keystream generation does not directly involve the key anymore.
Time-Memory-Data-Tradeoff Attacks

Given a function $F: M \rightarrow N$ and a value $y$, find $x$, such as $F(x) = y$

1) Brute-force: randomly pick value $x \in N$, until $F(x) = y$
2) Table look-up: precompute all values and store them in a table
3) Time-Memory-Data-Tradeoff attack

Two phases:

a) Precomputation (offline) phase: precompute the outputs for some values and store the corresponding pairs in a table
b) Real-time (online) phase: get function outputs and check if any of them is covered by the table
Time-Memory-Tradeoff Attacks costs

TMTO attacks are evaluated by looking at the following costs:

- **N** - search space
- **P** - the time effort of the precomputation phase
- **T** - the time effort of the online phase
- **M** - memory cost of the attack.
- **D** - number of usable data samples, i.e., outputs of $F$, during the online phase
Tradeoff Attacks Scenarios

- **Scenario A**: given one image $y \in N$, find a preimage $x \in N$ such that $F(x) = y$. The attack can only be successful if the given image $y$ has been considered during the precomputation phase. Hence precomputation time is equal to the search space $P = |N|$

- **Scenario B**: given D images $y_1, \ldots, y_D$ of $F$, find a preimage for any of these points, i.e., a value $x_i \in N$ such that $F(x_i) = y_i$
Knowledge of one internal state $S_{t}$ allows to find all succeeding (preceding) keystream bits
Goal: find any of the internal states

Search space $N = S = GF(2^\sigma)$

1) Offline phase: choose $M$ random internal states, precompute corresponding output prefixes of length $\sigma$ and store these pairs in a table.
   Precomputation time: $P = M$

2) Online phase: from the given keystream select $D$ possible keystream segments of length $\sigma$ and check if any of them is stored in the table.
   Online time: $T = D$

By birthday paradox: the intersection of $M$ and $D$ is likely if $MD = |S|$ or $MT = 2^\sigma$

When $T = M$ we have: $T = M = 2^{\sigma/2}$

Rule of thumb:

The internal state size needs to be at least twice the key length
Overall Question

Is it possible to avoid this rule of thumb with this design?
  • No – the attack is generic

⇒ Is it possible to tweak the stream cipher design to avoid this attack
  • We think „yes“.
Design Approach
Keystream-Equivalent States

Two states $S_t$ and $S'_t \in S$ are said to be **keystream-equivalent** ($S_t \equiv_{kse} S'_t$) if there exists an integer $r \geq 0$ such that

$$F_{out}^{compl}(Upd^r(St)) = F_{out}^{compl}(St').$$

$F_{out}^{compl}$ - function that outputs complete keysream

$Upd^r$ - $r$-times application of $Upd$.

Keystream-equivalent class of states:

$$[St] = \{St' \in S | St \equiv_{kse} St'\}$$

For most keystream generators (KSG):
Set of states $S = [St]$.

For arbitrary KSG:
Set of states $S = [St^{(1)}] \cup ... \cup [St^{(l)}]$.
Keystream-Equivalent States and TMDTO

Suppose that TMDTO attacker is given some keystream \( (z_t) \), based on state \( St^{(0)} \).

Then keystream-equivalent class \( [St^{(0)}] \) has to be considered either during the precomputation or online phase of TMDTO.

Hence: A TMDTO attack on the KSG will be a union of TMDTO attacks, one for each equivalence class.

If we have \( l \) equivalent classes: \( S = [St^{(1)}] \cup ... \cup [St^{(l)}] \) at least \( l \) TMDTO attacks are required

If we are able to design a KSG such that \( l \geq 2^k \), then the time effort is not less than of exhaustive key-search.
KSG with fixed internal states parts

---

**Update function**

**Fixed part**

**Variable part**

**Output function**

Keystream

---

\[ S_t^{(0)} = \{FP, VP_t\} \]

\[ \text{Upd} \]

\[ \text{FP} \]

\[ \text{VP}_t \]

\[ \text{Upd} \]

\[ \text{FP} \]

\[ \text{VP}_{t+i} \]

Complete keystream for class

Since fixed part (FP) cannot be changed, two different values of FP result into two different equivalence classes. Therefore \( l = 2^{|FP|} \)

Hence, if \(|FP| \geq \kappa\) then \( l \geq 2^\kappa \) and the attack effort is above exhaustive key search

---

Secret key as a part of the internal state?
KSG with Keyed Update Function

Key space \( K = GF(2^\kappa) \)
IV space \( IV = GF(2^\nu) \)
Variable state space \( S = GF(2^\sigma) \)

Initialization function \( Init: K \times IV \rightarrow S \)
Update function \( Upd: K \times S \rightarrow S, \)
Output function \( Out: S \rightarrow GF(2) \)

- The security level of \( \kappa \) wrt to TMDTO attacks is achievable independent of the length \( \sigma \) of variable state
6.2 Definition  A synchronous stream cipher is one in which the keystream is generated independently of the plaintext message and of the ciphertext.

The encryption process of a synchronous stream cipher can be described by the equations

$$\sigma_{i+1} = f(\sigma_i, k),$$
$$z_i = g(\sigma_i, k),$$
$$c_i = h(z_i, m_i),$$

where $\sigma_0$ is the initial state and may be determined from the key $k$, $f$ is the next-state function, $g$ is the function which produces the keystream $z_i$, and $h$ is the output function which combines the keystream and plaintext $m_i$ to produce ciphertext $c_i$. The encryption and decryption processes are depicted in Figure 6.1. The OFB mode of a block cipher (see §7.2.2(iv)) is an example of a synchronous stream cipher.

**Figure 6.1:** General model of a synchronous stream cipher.
Gain?

• Instead of a long internal state, we have now an internal state AND the key
• What can we gain?
• Observation 1
  • Storing a fixed value can be realized with significantly less GE than with a register
  • ⇒ if the key is fixed, one can save area (see also KATAN/KTANTAN)
• Observation 2
  • As the resistance against TMDTO attacks seems to depend on the key size only, the internal state may be smaller than the key size
  • ⇒ secure designs with small internal space (but sufficiently long key of course)
Stream Cipher Proposal Sprout
Main Design Idea

- **Primary goal**: show the feasibility of the approach for reducing the state size.
- Build upon an **existing established design**.
- The prototype cipher should be **lightweight**, **scalable** (at least to some extent), and has **undergone** already some **cryptanalysis**.
- **Grain 128a** meets our requirements
Our Changes:

- Reduce the FSR sizes
- Involve the key into the NLFSR update function
- Modify the functions

* Authentication part is excluded
Structure of Sprout

- $k_0$, $k_1$, ..., $k_{79}$
- Round key function
- $k_t^*$
- $g$
- NLFSR (40 bits)
- LFSR (40 bits)
- Initialization phase
Initialization

- IV = 70 bits are loaded into the FSRs
- Run 320 cycles without producing any output
Round Key Function (1)

- Counter
- LFSR: \( L_4 \), \( L_{21} \), \( L_{37} \)
- NLFSR: \( N_9 \), \( N_{20} \), \( N_{29} \)
- XOR
- AND
- \( k_0 \), \( k_1 \), \( \ldots \), \( k_{79} \)
- Select \( k_{(t \mod 80)} \)
- \( k_t^* (t \leq 79) \)
- \( k_t^* (t > 79) \)
Round Key Function (2)

\[ k_t^* = k_t, \quad 0 \leq t \leq 79 \]

\[ k_t^* = (k_t \mod 80) \cdot (L_4 + L_{21} + L_{37} + N_9 + N_{20} + N_{29}), \quad t > 79 \]

- The mechanism operates on key bits one by one
- Clocks 0,\ldots,79 ensure that at the end of the initialization, both registers depend on all key bits.
- At each of the following clocks the round key bit is used with the probability \( \frac{1}{2} \).
- We reuse the counter which is required for the initialization phase
## Implementation results

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Area size (GE)</th>
<th>Throughput (Kb/s)*</th>
<th>Logic process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Block ciphers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRESENT 80</td>
<td>1570</td>
<td>200</td>
<td>0.18 μm</td>
</tr>
<tr>
<td>PRESENT 80</td>
<td>1000</td>
<td>11.4</td>
<td>0.35 μm</td>
</tr>
<tr>
<td>KATAN32</td>
<td>802</td>
<td>12.5</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KATAN48</td>
<td>927</td>
<td>18.8</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KATAN64</td>
<td>1054</td>
<td>25.1</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KTANTAN32</td>
<td>462</td>
<td>12.5</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KTANTAN48</td>
<td>588</td>
<td>18.8</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>KTANTAN64</td>
<td>688</td>
<td>25.1</td>
<td>0.13 μm</td>
</tr>
<tr>
<td><strong>Stream ciphers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mickey</td>
<td>3188</td>
<td>100</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>Trivium</td>
<td>2580</td>
<td>100</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>Grain 80</td>
<td>1294</td>
<td>100</td>
<td>0.13 μm</td>
</tr>
<tr>
<td>Grain 80</td>
<td>1180</td>
<td>100</td>
<td>0.18 μm</td>
</tr>
<tr>
<td><strong>Sprout</strong></td>
<td><strong>810</strong></td>
<td><strong>100</strong></td>
<td><strong>0.18 μm</strong></td>
</tr>
</tbody>
</table>

Scenario: Fixed key which can be stored without using any additional GE
Security Analysis (1)

Algebraic attacks:

- **Classical** algebraic attacks do **not work** against NLFSR based constructions

Combination of algebraic and guess and determine attacks:

- Effort to guess internal state = effort to guess key
- Even if all the state bits are guessed at some clock $t$ it is not obvious how to compute the key:
  1. At each clock, the unknown round key bit is added to the last NLFSR bit $N_{t+39}$
  2. During the next clock cycle it influences the next state and moves to position $N_{t+38}$ afterwards
  3. There, it is used for the first time in the output function as a part of the monomial $N_{t+4}N_{t+38}L_{t+32}$
  4. The attacker can learn something about this key bit only if both $N_{t+4}$ and $L_{t+32}$ are equal to 1 (probability $1/4$).
Security Analysis (2)

Linear Approximations:

- In Grain family it is always possible to find a time-invariant biased linear relation between the LFSR bits and the keystream bits.
- In the case of Sprout, such a relation would have to include the round key bits, which are a nonlinear expression in the key bits and the state bits.
- Moreover, in Sprout we use the update and the output functions with the same cryptographic properties as in Grain 128a (secure against this type of attack).

Chosen IV attacks:

- The functions with the same properties as in Grain 128a
- Higher ratio between the number of rounds during the initialization phase to the state size
Security Analysis (3)

Cube Attacks:
- So far, no mechanisms are known to show the resistance against such attacks
- No attacks against Grain 128a are currently known
- Initialization phase is longer

Time-Data-Memory Trade-off Attacks:
- Key-dependent state update function
- The key can be considered as a fixed part of the state
- The number of equivalent classes is at least $2^{80}$

Weak Key-IV pairs:
- There exist Key-IV pairs which lead to the all-zero LFSR state after the initialization phase
- The number of keystream bits required for the attack on Grain 128 is $2^{86}$
- In case of Grain 128a (and Sprout) the cryptographic properties of the functions are better meaning that even more keystream bits will be required to detect a weak pair
- Moreover, we aim for a keystream length of $2^{40}$ (maximum period of the LFSR)
Fault attacks:
- All members of the Grain family have been broken
- However, all these attacks aim to recover the internal state.
- As elaborated above, knowing the internal state of Sprout does not automatically allow for efficiently recovering the secret key.
- The involvement of the round key bits should make this type of attacks harder.

Side channel attacks:
- Will depend on actual implementation
- Implementing countermeasures requires additional area and power recourses
- When less area and power are used to implement the scheme itself, there is more space left for the countermeasures.
Conclusions
**Contribution**

- Modified streamcipher design
- Update function involves directly the key
- Motivation/hope: internal state size can be reduced
- Proof of concept streamcipher Sprout
- Sprout is significantly smaller than all eStream finalists
Open Questions

- Further security analysis of Sprout
- Better/other round key functions?
- In general, new stream cipher designs?
- How small can the internal state be?
  - Example: fixed 128 bit key, 30 bit internal state
Questions?

How was your security experience today?

Tell us how we can make your journey better.