Turing: a fast software stream cipher

Greg Rose, Phil Hawkes
{ggr, phawkes}@qualcomm.com
DISCLAIMER!

- This version (1.8 of TuringRef.c) is what we expect to publish. Any changes from now on will be because someone broke it. (Note: we said that about 1.5 and 1.7 too.)
- This is an experimental cipher. Turing might not be secure. We've already found two attacks (and fixed them). We're starting to get confidence.
- Comments are welcome.
- Reference implementation source code agrees with these slides.
Introduction

- Stream ciphers
- Design goals
- Using LFSRs for cryptography
- Turing
- Keying
- Analysis and attacks
- Conclusion
Stream ciphers

• Very simple
  – generate a stream of pseudo-random bits
  – XOR them into the data to encrypt
  – XOR them again to decrypt

• Some gotchas:
  – can’t ever reuse the same stream of bits
    – so some sort of facility for Initialization Vectors is important
  – provides privacy but not integrity / authentication
  – good statistical properties are not enough for security… most PRNGs are no good.
Turing's Design goals

• **Mobile phones**
  – cheap, slow, small CPUs, little memory

• **Encryption in software**
  – cheaper
  – can be changed without retooling

• **Stream cipher**
  – two-level keying structure (re-key per data frame)
  – stream is "seekable" with low overhead

• **Very fast and simple, aggressive design**

• **Secure** (? – we think so, but it's experimental)
Using LFSRs for Cryptography

• Linear Feedback Shift Registers have been intensively studied
  – Good and proven distribution properties
  – Fast

• Empirical techniques thought to produce good characteristics
  – decimation, irregular clocking, *stuttering* (not used)
  – nonlinear function of state, or memory
  – combining multiple registers (not used in Turing)

• Theory all works for any field
  – but some things are more efficient in software
Elements in Turing

• LFSR structure based on SOBER-t32 and SNOW 2.0

• Nonlinear filter function is a keyed transformation
  – Based on a round of a block cipher
  – Blowfish/Twofish for the key-dependent s-box
  – SAFER for the pseudo-Hadamard diffusion function
  – concept spawned from Tom St Denis' *tc24*
LFSRs over GF(2^8)^4

- Elements of field are words-sized polynomials of byte-sized binary polynomials
- Addition operation is XOR, ⊕
- Multiplication is poly-mod multiplication, ⊗
  - only multiply by constant
  - use single 8→32 table lookup, shift word by 8 bits
- Instead of shifting the shift register, can use:
  - pointers to memory -- sliding window or circular buffer
  - or inline code for real speed
Turing block diagram

Register

Keyed block cipher round

Output

PHT
Mix

Keyed 8x32 s-boxes
produce mixed words

PHT
Mix

step five
and add

A
B
C
D
E

A'
B'
C'
D'
E'
The Shift Register

- Generates nearly maximal length sequence
  - period \((2^{544} - 1)/5\), 5 possible cycles
- Recurrence:
  \[ s_{n+17} = s_{n+15} \oplus s_{n+4} \oplus y \otimes s_n \]
- Each bit position behaves as output from a 544-bit binary shift register
  - recurrence relation has exactly half non-zero coefficients.
- Shift register is “free running”
  - that is, its state is used directly instead of its output
The Nonlinear Function

- Offsets are carefully chosen
  - feedback taps plus function inputs form “full positive difference set”
- Combine 10 words of state in a key-dependent manner
  - mix 5 words with PHT
  - pass bytes through keyed 8->32 S-boxes
  - mix words with PHT again
  - step LFSR five times and add other words mod $2^{32}$
Pseudo-Hadamard Transform

- Matrix multiply mod $2^{32}$:
  \[
  \begin{bmatrix}
  A \\
  B \\
  C \\
  D \\
  E
  \end{bmatrix} = \begin{bmatrix}
  2 & 1 & 1 & 1 & 1 \\
  1 & 2 & 1 & 1 & 1 \\
  1 & 1 & 2 & 1 & 1 \\
  1 & 1 & 1 & 2 & 1 \\
  1 & 1 & 1 & 1 & 2
  \end{bmatrix} \begin{bmatrix}
  A \\
  B \\
  C \\
  D \\
  E
  \end{bmatrix}
  \]

- Actually:
  - $E \leftarrow A + B + C + D$;
  - $A \leftarrow E$; $B \leftarrow E$; $C \leftarrow E$; $D \leftarrow E$

- Extend to n-PHT for key loading
Basic S-box

- Permutation
- Fairly nonlinear (min nonlinearity 104)
- made by:
  - keying RC4 with "Alan Turing"
  - throwing away 736 bytes
  - using its permutation
  - best observed nonlinearity in 10,000 cycles, used first one found.
QUT's "Q-box"

- Developed by Queensland University of Technology
- 8->32 bit S-box
- bit positions are:
  - Highly nonlinear (112)
  - Balanced
  - Pairwise uncorrelated
keyed S-boxes

• Push through fixed S-box multiple times:
  – next slide for details
• four logical S-boxes, one for each byte position, due to different key material
• words of key are mixed when loaded to help thwart related-key attacks.
• every byte of key affects each sbox
• S-boxes can be precalculated
More on keyed S-boxes

- \( S_i(x) = \text{Sbox}[K_{i,N-1} \oplus \text{Sbox}[K_{i,N-2} \oplus \ldots \text{Sbox}[K_{i,0} \oplus x] \ldots]] \)
- at each stage, XOR (Qbox[... ] rotated left \((i*8+j)\)) to temporary word
  - Thanks to David McGrew and Scott Fluhrer for observing that this was better mixing than the MDS matrix
- Clobber the byte corresponding to the input byte with \( S_i(x) \)
  - This ensures that the corresponding output bits are balanced w.r.t. \( x \).
  - If not, there's a possible bias introduced that might be exploitable.
- not invertible
Keying

- Two stage keying
  - secret key from 4 to 32 bytes (32 to 256 bits)
    - length is significant
    - must be multiple of 4
  - requires further keying operations (eg. frame IV)

- Keying process:
  - pass bytes through an invertible S-box construct, then words through PHT for mixing
  - use output in keyed S-boxes
  - can set up fast table lookups for the keyed S-boxes
Initialization Vector

- Mixed key material and initialization vector are used to fill the LFSR
  - IV
  - Mixed key
    - Word made from length of key and IV (0x010203kv, where k is key length in words, v is IV length in words)
    - Rest of words made by recurrence of some previous words
- Each word is mixed through S-box
  - IV goes through an invertible key-independent S-box-based transformation to avoid equivalent IVs
  - Key is already mixed
  - Others go through the keyed S-box
- Finally whole register is mixed with PHT
Polynomial details

- **Byte polynomial is**
  \[ z^8 + z^6 + z^3 + z^2 + 1 \]
- **Word polynomial is**
  \[ y^4 + 0xD0.y^3 + 0x2B.y^2 + 0x43.y + 0x67 \]
- **LFSR polynomial is**
  \[ x^{17} + x^{15} + x^4 + \alpha, \text{ where } \alpha \text{ is the polynomial } y \]
- "binary equivalent" polynomial has 272 out of 544 non-zero terms
Performance

- Generates 160 bits at a time
- Highly parallel operations
- 2304 bytes ROM tables, plus code
  - 8x8 S-box, 8x32 Q-box, 8x32 Multiplication table
- Fast implementation:
  - 4-5 cycles per byte on newer Pentium-style machines with multiple parallel instructions
  - Requires 4K RAM tables computed at key setup
- Small implementation:
  - 68 bytes RAM, very little key setup
Security

• LFSR guarantees good statistical properties input to the nonlinear function
• Strength is derived from the combination of unknown input from the LFSR and keyed non-linear transformation
• Either by itself is potentially weak
• Each frustrates attacks on the other
• If the "block cipher part" is secure, CTR mode proof applies (but we don't claim this)
## Numbers

<table>
<thead>
<tr>
<th>Cipher</th>
<th>cycles/B</th>
<th>Key</th>
<th>IV setup</th>
<th>tables</th>
<th>RAM</th>
<th>MByte/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>TuringRef</td>
<td>149.01</td>
<td>477.00</td>
<td>4272.31</td>
<td>2304.00</td>
<td>68.00</td>
<td>6.04</td>
</tr>
<tr>
<td>TuringLazy</td>
<td>33.44</td>
<td>1802.70</td>
<td>991.80</td>
<td>2304.00</td>
<td>4164.00</td>
<td>26.91</td>
</tr>
<tr>
<td>TuringTab</td>
<td>30.06</td>
<td>72457.93</td>
<td>900.90</td>
<td>2304.00</td>
<td>4164.00</td>
<td>29.94</td>
</tr>
<tr>
<td>TuringFast</td>
<td>6.12</td>
<td>72417.12</td>
<td>882.90</td>
<td>2304.00</td>
<td>4164.00</td>
<td>146.95</td>
</tr>
<tr>
<td>arrsyfor</td>
<td>37.49</td>
<td>0.00</td>
<td>10347.42</td>
<td>0.00</td>
<td>258.00</td>
<td>24.00</td>
</tr>
<tr>
<td>AES enc.</td>
<td>26.85</td>
<td>239.00</td>
<td>0.00</td>
<td>20480.00</td>
<td>176.00</td>
<td>33.53</td>
</tr>
</tbody>
</table>

MHz 900.00 (Mobile Pentium III IBM laptop)
Recent attack

- Basically, LFSR wasn't being stepped enough
- Reuse of words in final "add" phase allowed algebraic attack on LSB's
- Attack very specific to Turing 1.6.
- Solution:
  - step LFSR total 5 times between blocks
  - use a different full positive difference set to extract the words for the final addition round
- Attack doesn't actually work, but scared us
Conclusion

• Turing is *not* conservatively designed
  – I (ggr) think it may be secure, but it's clearly "close to the edge". Maybe too close.
• OK for hardware implementation
• Suitable for medium embedded applications
• Extremely fast in software
  – 146MBytes/sec on 900MHz PIII laptop, 6 cycles/byte
• Source code available worldwide:
• Being published, reviewed
Stop the presses

• Free licenses!
  – for Turing or SOBER (or future ciphers)
  – for any purpose
  – for hardware or software
  – for ever
  – our code or yours

• It only took me 5 years to get management agreement…