HS1-SIV

Ted Krovetz
Sacramento State

Goals:

• Assemble from off-the-shelf, well-known parts
• Nonce reuse tolerant
• Balanced performance
• Provable security
Main Idea

- SIV for nonce reuse tolerance [RS06]
- Universal hash for input consumption
- Stream cipher for byte production
SIV

- Secure against nonce-respecting adversaries
- Easy to see when \((N, AD, M)\) repeats
- Birthday bound: SIV repetitions
PRF-Only Version

- Let F be a VIL+VOL PRF
- $\text{SIV} = F(\text{Nonce} \ || \ \text{AD} \ || \ M) \ [1..16]$
- $\text{C} = F(\text{Nonce} \ || \ \text{SIV}) \ [64..] \oplus M$
- Send (Nonce, C, SIV)
Universal Hash: NH

- 64:8 compression. $2^{-32}$ collision probability.
- Efficient on vector and scalar CPUs.

[BHKKR99]
Universal Hash: Poly61

- After compression do poly-eval hash to 8 bytes.

- \( a=1 \)
  
  while (bytes remain)
  
  \[ M = \text{next 64 bytes} \]
  
  \[ a = ak + ( \text{NH}(M) \mod 2^{60} ) \mod 2^{61}-1 \]

- \( 2^{-28} + m2^{-67} \) collision probability hashing \( m \) bytes.
  
  \( 2^{-28} \) when messages are each \( < 2^{38} \) bytes.

- \( 2^{-112} \) when done 4 times.
## Security Levels

<table>
<thead>
<tr>
<th></th>
<th>Hash</th>
<th>Cipher</th>
<th>SIV</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hs1-siv-lo</strong></td>
<td>2 x</td>
<td>Chacha8</td>
<td>64 bits</td>
<td>56 bits</td>
</tr>
<tr>
<td><strong>hs1-siv</strong></td>
<td>4 x</td>
<td>Chacha12</td>
<td>128 bits</td>
<td>112 bits</td>
</tr>
<tr>
<td><strong>hs1-siv-hi</strong></td>
<td>6 x</td>
<td>Chacha20</td>
<td>256 bits</td>
<td>168 bits</td>
</tr>
</tbody>
</table>

- **Security**: Adversary wins if
  - Hash collision — birthday bound on “Security”
  - SIV collision — birthday bound on “SIV”
  - Chacha failure — $\text{Adv}^{\text{prf}}(\text{Chacha})$

- **Assumption**: Chacha core is a prf.
Targeting 32-bit Ops

- 64-bit CPUs perform 32-bit ops well in vectors.
- 32-bit CPUs don’t perform 64-bit ops well.
- Maximize 32-bit operations (NH-32, Chacha)
- Minimize 64-bit operations (64-bit mult is rare)
- Targeting 32-bits provides balanced performance.
## Performance

<table>
<thead>
<tr>
<th></th>
<th>MIPS32</th>
<th>Cortex-A9</th>
<th>Haswell</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x HS1 Hash</td>
<td>16 cpb</td>
<td>5 cpb</td>
<td>0.8 cpb</td>
</tr>
<tr>
<td>Chacha12</td>
<td>20 cpb*</td>
<td>7 cpb*</td>
<td>0.8 cpb*</td>
</tr>
<tr>
<td>AES128</td>
<td>60 cpb*</td>
<td>23 cpb*</td>
<td>9 cpb* (0.6HW)</td>
</tr>
</tbody>
</table>

- Preliminary: unoptimized HS1 Hash in C.

Romain Dolbeau reports 2 cpb for $2^{-168}$ security on Haswell (6 x HS1 Hash + Chacha20) in C.

* As reported by SUPERCOP (http://bench.cr.yp.to)
Questions
OCB AE API

- API developed by Rogaway and Krovetz for OCB

\[ \text{ae_encrypt}(\text{ctx, n, pt, ad, final}) : \text{returns ct} \]

<table>
<thead>
<tr>
<th>n</th>
<th>ad</th>
<th>final</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>True</td>
<td>All-in-one</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>True</td>
<td>Reuse AD</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes / No</td>
<td>False</td>
<td>New Incremental</td>
</tr>
<tr>
<td>No</td>
<td>—</td>
<td>False</td>
<td>Continue</td>
</tr>
<tr>
<td>No</td>
<td>—</td>
<td>True</td>
<td>Finalize</td>
</tr>
</tbody>
</table>