Implementação eficiente em software da função Lyra2 em arquiteturas modernas

Guilherme P. Gonçalves[1] Diego F. Aranha[1]

Nov/2015

[1]Laboratório de Segurança e Criptografia (LASCA) – UNICAMP

Introdução - Esquemas modernos de hash de senhas

Forma extremamente comum de autenticação

Esquema de *hash* deve ser **seguro**

- Resistência ao cálculo de pré-imagem
- Proteção contra adversários...
  - ...com acesso direto ao banco de dados
  - ...executando ataques de busca exaustiva
  - ...usando *hardware* dedicado

Ashley Madison: bcrypt :-) / md5 da senha "normalizada" :-(

Introdução - Esquemas modernos de hash de senhas

Interface: $h = HASH(password, salt, T, M)$

Esquemas parametrizados:

- bcrypt (1999), parâmetro de tempo
- PBKDF2 (2000), parâmetro de tempo
- scrypt (2012), um parâmetro de tempo e espaço
- *Lyra2 (2014), parâmetros independentes de tempo e espaço*

Como escolher os parâmetros?

O esquema de hash de senhas Lyra2

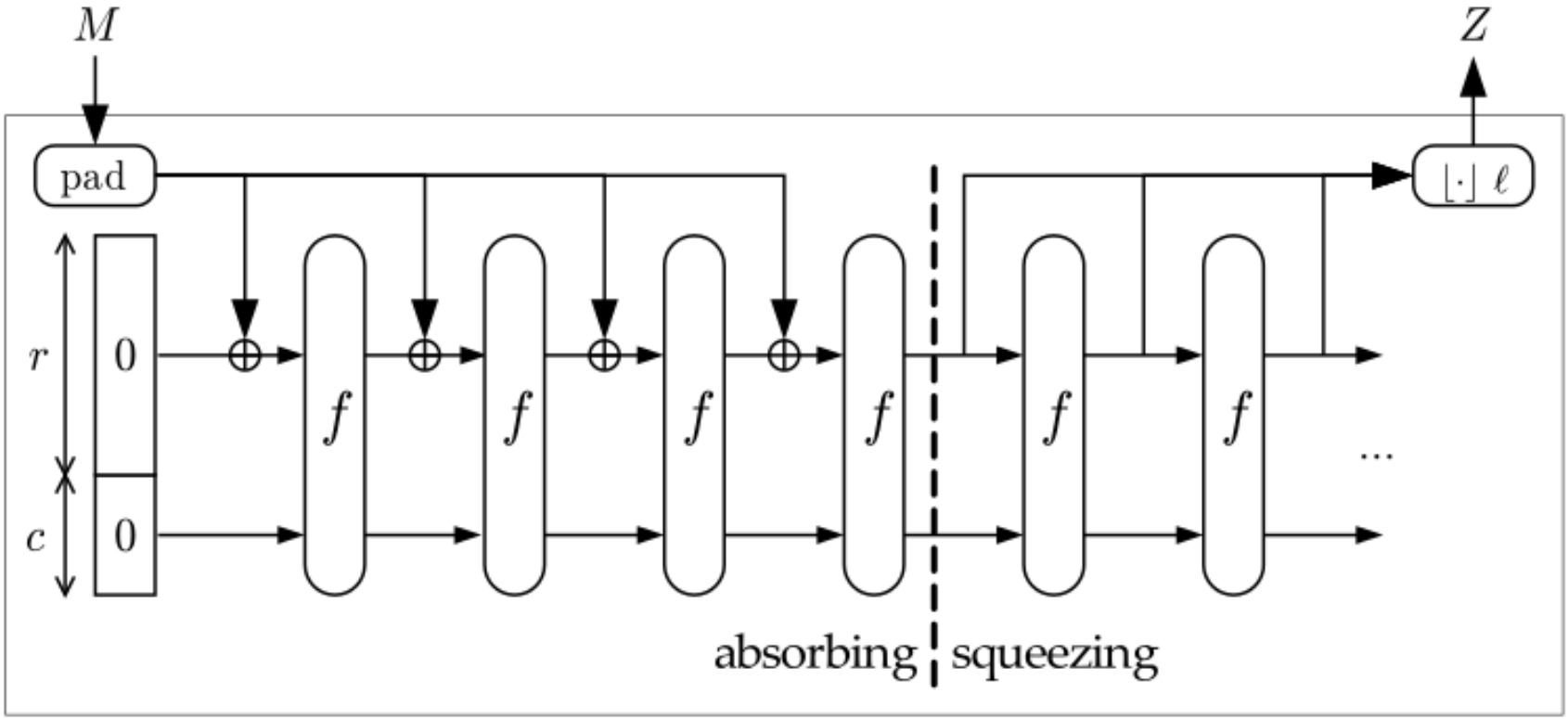
Finalista do Password Hashing Competition

Construção de esponja e função de compressão da BLAKE2b

Este trabalho:

- Implementação em *software* do Lyra2 v2.5 sequencial
- Inconsistências na especificação/implementação de referência
- Desempenho cerca de 17% (SSE2) e 30% (AVX2) superior

Lyra2 - A construção de esponja



sponge

Generaliza funções de *hash*

Construção similar: *duplex*

No Lyra2: esponja suporta *absorb*, *squeeze* e *duplex* independentes

Lyra2 - A função de compressão

Com a, b, c, d inteiros de 64 bits, define-se $G(a, b, c, d)$:

$$
\begin{aligned}
a &\leftarrow a + b \\
d &\leftarrow (d \oplus a) \ggg 32 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 24 \\
a &\leftarrow a + b \\
d &\leftarrow (d \oplus a) \ggg 16 \\
c &\leftarrow c + d \\
b &\leftarrow (b \oplus c) \ggg 63
\end{aligned}
$$

Na BLAKE2b: Matriz de estado 4×4 de inteiros

$G_{//}(colunas) + G_{//}(diagonais) = rodada = f$

Estado da esponja = matriz de estado BLAKE2b linearizada

Lyra2 - Descrição

Interface: $h = LYRA2(password, salt, T, R, C)$

Matriz de estado de $R \times C$ blocos de b bits

Três etapas:

- *Bootstrapping*: inicializa a esponja, *absorbs* das entradas
- *Setup*: inicializa a matriz de estado com *squeezes* e *duplexes*
- *Wandering*: *duplex* de células pseudoaleatórias, T iterações

Finalmente: $h = squeeze()$

Tempo: $O((T+1) \cdot R \cdot C)$

Espaço: $O(R \cdot C)$

Implementação proposta

Disponível em: https://github.com/guilherme-pg/lyra2.git

Licença: MIT

Versões SSE2 e AVX2 escritas em C (padrão C99)

Fácil extensão para novos conjuntos de instruções

Implementação proposta - AVX2

Conjunto de instruções recente: vetores 256 *bits*

Implementação nova da função de compressão

Matriz de estado BLAKE2b / estado da esponja Lyra2:

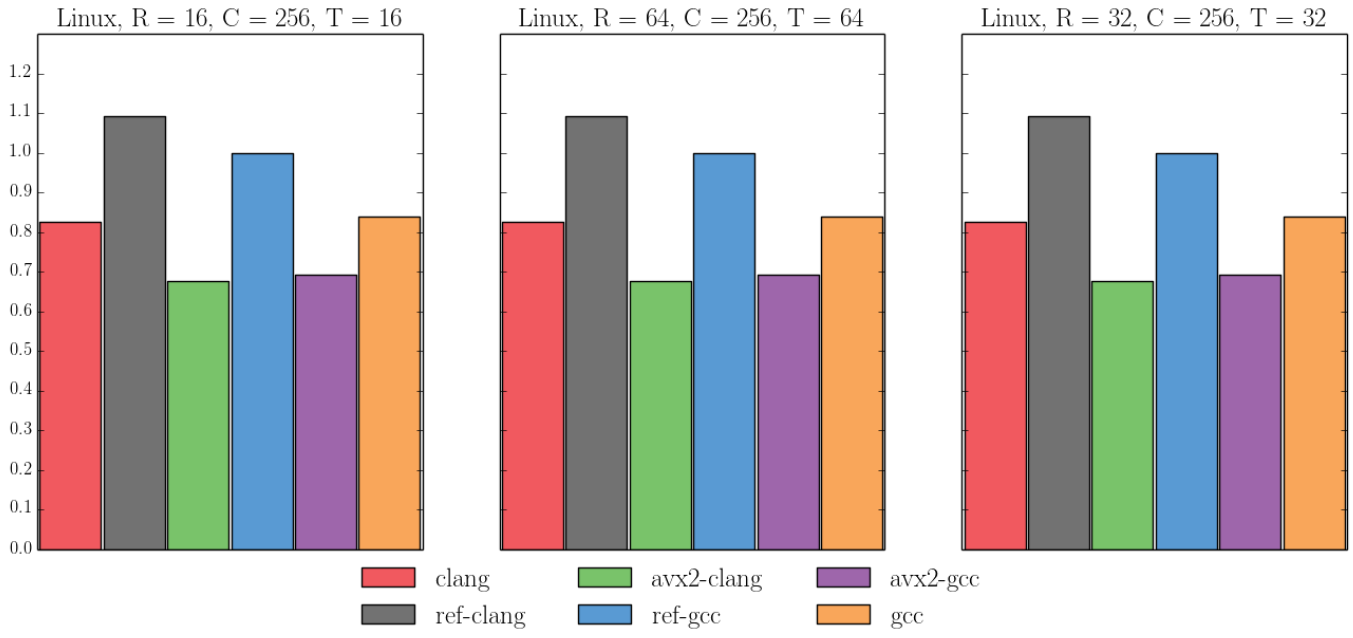
$$\begin{pmatrix} v_0 & v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 & v_7 \\ v_8 & v_9 & v_{10} & v_{11} \\ v_{12} & v_{13} & v_{14} & v_{15} \end{pmatrix}$$

Rodada: $G_{//}(\textit{colunas})$, *diagonalize*, $G_{//}(\textit{colunas})$, *undiagonalize*

Semelhante à versão SSE2, mas 1 linha $=$ 1 vetor

Resultados experimentais

Tempos de execução normalizados (quanto menor, melhor)



Legenda:

- ref-gcc, ref-clang: implementação de referência, SSE2
- gcc, clang: implementação proposta, SSE2
- avx2-gcc, avx2-clang: implementação proposta, AVX2

Trechos de código - Absorb

```
1  static inline void
2  sponge_absorb(sponge_t *sponge, sponge_word_t *data,
3                size_t databytes) {
4      size_t rate = SPONGE_RATE_LENGTH;
5      size_t datalenw = databytes / sizeof(sponge_word_t);
6      assert(datalenw % rate == 0);
7      while (datalenw) {
8          for (unsigned int i = 0; i < rate; i++) {
9              sponge->state[i] ^= data[i];
10         }
11         sponge_compress(sponge);
12         data += rate;
13         datalenw -= rate;
14     }
15     return;
16 }
```

Trechos de código - Função G

```
1  #define G(row1,row2,row3,row4)                \
2    row1 = _mm256_add_epi64(row1, row2); \
3    row4 = _mm256_xor_si256(row4, row1); \
4    row4 = _mm256_roti_epi64(row4, -32); \
5    row3 = _mm256_add_epi64(row3, row4); \
6    row2 = _mm256_xor_si256(row2, row3); \
7    row2 = _mm256_roti_epi64(row2, -24); \
8    row1 = _mm256_add_epi64(row1, row2); \
9    row4 = _mm256_xor_si256(row4, row1); \
10   row4 = _mm256_roti_epi64(row4, -16); \
11   row3 = _mm256_add_epi64(row3, row4); \
12   row2 = _mm256_xor_si256(row2, row3); \
13   row2 = _mm256_roti_epi64(row2, -63);
```