The TIB3 Hash

Miguel Montes-Daniel Penazzi

February 2009

Table of Contents

Table of Contents

Design choices

Design choices

- Nonlinearity not based exclusively on xor-sum interaction (use of Sboxes)

Design choices

- Nonlinearity not based exclusively on xor-sum interaction (use of Sboxes)
- Speed should be better than SHA2 family.

Design choices

- Nonlinearity not based exclusively on xor-sum interaction (use of Sboxes)
- Speed should be better than SHA2 family.
- Speed should be good on both 64 bit platforms and 32-bit platforms.

Design choices

- Nonlinearity not based exclusively on xor-sum interaction (use of Sboxes)
- Speed should be better than SHA2 family.
- Speed should be good on both 64 bit platforms and 32-bit platforms.
- Reuse of previous block to make attacks more difficult.

Design choices

- Nonlinearity not based exclusively on xor-sum interaction (use of Sboxes)
- Speed should be better than SHA2 family.
- Speed should be good on both 64 bit platforms and 32-bit platforms.
- Reuse of previous block to make attacks more difficult.
- Sboxes should be small to obtain efficient implementation via bitslicing.

Design choices

- Nonlinearity not based exclusively on xor-sum interaction (use of Sboxes)
- Speed should be better than SHA2 family.
- Speed should be good on both 64 bit platforms and 32-bit platforms.
- Reuse of previous block to make attacks more difficult.
- Sboxes should be small to obtain efficient implementation via bitslicing.
- Use of simple operations for ease of implementation on restricted environments

Overview

The main security differences of TIB3 with respect to SHA2 are:

Overview

The main security differences of TIB3 with respect to SHA2 are:

- Besides the usual previous hash and current message block, the compression function also uses the number of bits processed and the previous message block.

Overview

The main security differences of TIB3 with respect to SHA2 are:

- Besides the usual previous hash and current message block, the compression function also uses the number of bits processed and the previous message block.
- The last iteration is different from the intermediate ones, to prevent extension attacks.

Overview

The main security differences of TIB3 with respect to SHA2 are:

- Besides the usual previous hash and current message block, the compression function also uses the number of bits processed and the previous message block.
- The last iteration is different from the intermediate ones, to prevent extension attacks.
- The underlying block cipher uses non-linear bijective Sboxes in addition to the sum-xor interaction.

Overview

The main security differences of TIB3 with respect to SHA2 are:

- Besides the usual previous hash and current message block, the compression function also uses the number of bits processed and the previous message block.
- The last iteration is different from the intermediate ones, to prevent extension attacks.
- The underlying block cipher uses non-linear bijective Sboxes in addition to the sum-xor interaction.
- The expansion of the key is done in such a way that a backward recursion is unlikely to succeed.

Table of Contents

Padding

- $\ell =$ bitlength of message M.

Padding

- $\ell =$ bitlength of message M.
- $t = \lceil \frac{\ell}{r} \rceil$. ($r = 512$ or 1024).

Padding

- $\ell =$ bitlength of message M.
- $t = \lceil \frac{\ell}{r} \rceil$. ($r = 512$ or 1024).
- Divide the first $(t-1)r$ bits of M into blocks $m_1, ..., m_{t-1}$, each of length r.

Padding

- $\ell =$ bitlength of message M.
- $t = \lceil \frac{\ell}{r} \rceil$. ($r = 512$ or 1024).
- Divide the first $(t-1)r$ bits of M into blocks $m_1, ..., m_{t-1}$, each of length r.
- If ℓ is a multiple of r m_t equals the last r bits of M. Otherwise, m_t has the last $\ell - r(t-1)$ bits of M, then a 1, and then 0s as needed to complete r bits.

Padding

- $\ell =$ bitlength of message M.
- $t = \lceil \frac{\ell}{r} \rceil$. ($r = 512$ or 1024).
- Divide the first $(t-1)r$ bits of M into blocks $m_1, ..., m_{t-1}$, each of length r.
- If ℓ is a multiple of r m_t equals the last r bits of M. Otherwise, m_t has the last $\ell - r(t-1)$ bits of M, then a 1, and then 0s as needed to complete r bits.
- There is a last block m_{t+1} that does not depend on the message, (only on the length of the message) that consists of $\ell \bmod 2^{64}$ in the first 64 bits,followed by $r - 64$ zeroes.

Iteration

Iteration

Based on 256-bit or 512-bit block cipher.

$$h_i = \begin{cases} E^{\ell_i}_{m_i||m_{i-1}}(h_{i-1}) \oplus h_{i-1} & i \leq t \\ E^{\ell_i}_{m_i \oplus \hat{h}_{i-1}||m_{i-1}}(h_{i-1}) \oplus h_{i-1} & i = t+1 \end{cases}$$

Iteration

Based on 256-bit or 512-bit block cipher.

$$h_i = \begin{cases} E^{\ell_i}_{m_i||m_{i-1}}(h_{i-1}) \oplus h_{i-1} & i \leq t \\ E^{\ell_i}_{m_i \oplus \hat{h}_{i-1}||m_{i-1}}(h_{i-1}) \oplus h_{i-1} & i = t+1 \end{cases}$$

where

$$\ell_i = \begin{cases} i.r \bmod L & i = 1, ..., t-1 \\ \ell \bmod L & i = t \\ 0 & i = t+1 \end{cases}$$

Iteration

Based on 256-bit or 512-bit block cipher.

$$h_i = \begin{cases} E^{\ell_i}_{m_i||m_{i-1}}(h_{i-1}) \oplus h_{i-1} & i \leq t \\ E^{\ell_i}_{m_i \oplus \hat{h}_{i-1}||m_{i-1}}(h_{i-1}) \oplus h_{i-1} & i = t+1 \end{cases}$$

where

$$\ell_i = \begin{cases} i.r \bmod L & i = 1, ..., t-1 \\ \ell \bmod L & i = t \\ 0 & i = t+1 \end{cases}$$

and $\hat{h} = \overbrace{0...0}^{r-n}||h$ is the extension of an element of $\{0,1\}^n$ to an element of $\{0,1\}^r$ by appending zeroes to the left.

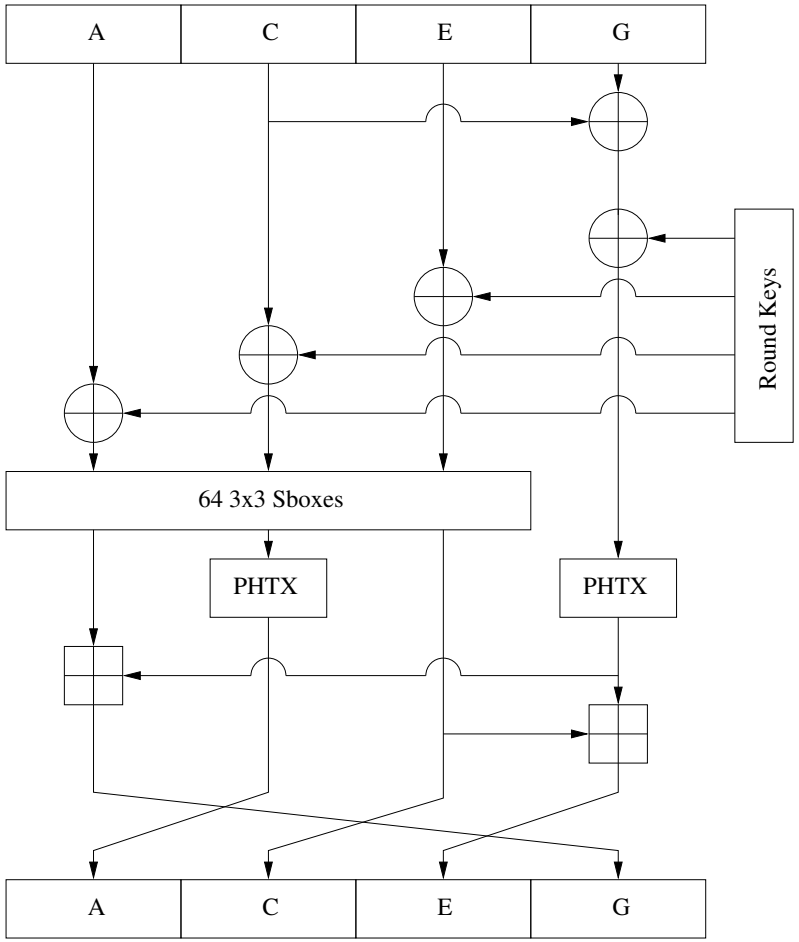
Block cipher (256 case)

The underlying cipher has 16 rounds. Dividing the 256 bit state into four sixty-four bit words as A,C,E and G. Each round is

$$\begin{aligned} G &:= G \oplus C \\ (A, C, E, G) &:= (A, C, E, G) \oplus \mathit{roundkeys} \\ (A, C, E) &:= \mathit{Sbox}(A, C, E) \\ G &:= \mathit{PHTX}(G) \\ C &:= \mathit{PHTX}(C) \\ A &:= A \tilde{+} G \\ G &:= E \tilde{+} G \\ (A, C, E, G) &:= (C, E, G, A) \end{aligned}$$

where *Sbox* is the passage in a bitslice way through 64 3 by 3 Sboxes. and $\tilde{+}$ is the sum of $(\mathbb{Z}/(2^{32}))^2$.

Round



Round Components

Sbox

$01234567 \mapsto 64170352$
$(001 = 1, 010 = 2$, etc)

Round Components

Sbox

$01234567 \mapsto 64170352$
$(001 = 1, 010 = 2$, etc)

PHTX

$D^* = PHTX(D)$:

$$\begin{aligned} \tilde{D} &= D + (D \ll 32) + (D \ll 47) \\ D^* &= \tilde{D} \oplus (\tilde{D} \gg 32) \oplus (\tilde{D} \gg 43) \end{aligned}$$

where $+$ is the 64 bit sum

Key Expansion

$K = (LK, RK)$, each of 512 bits.

Key Expansion

$K = (LK, RK)$, each of 512 bits. Load $LK \oplus RK$ into $D_0, ..., D_7$, then compute:

Key Expansion

$K = (LK, RK)$, each of 512 bits. Load $LK \oplus RK$ into $D_0, ..., D_7$, then compute:

$$
\begin{aligned}
D_8 &= \psi(D_3 \oplus RK_0, D_4 \oplus RK_1, D_5 \oplus RK_2, D_1 \oplus RK_3) \\
D_9 &= \psi(D_2 \oplus RK_4 \oplus const, D_7 \oplus RK_5 \oplus \ell_i, D_6 \oplus RK_7, D_0 \oplus RK_6)
\end{aligned}
$$

Key Expansion

$K = (LK, RK)$, each of 512 bits. Load $LK \oplus RK$ into $D_0, ..., D_7$, then compute:

$$
\begin{aligned}
D_8 &= \psi(D_3 \oplus RK_0, D_4 \oplus RK_1, D_5 \oplus RK_2, D_1 \oplus RK_3) \\
D_9 &= \psi(D_2 \oplus RK_4 \oplus const, D_7 \oplus RK_5 \oplus \ell_i, D_6 \oplus RK_7, D_0 \oplus RK_6) \\
D_j &= \psi(D_{j-10}, D_{j-8}, D_{j-3}, D_{j-2}) \qquad j \geq 10
\end{aligned}
$$

Key Expansion

$K = (LK, RK)$, each of 512 bits. Load $LK \oplus RK$ into $D_0, ..., D_7$, then compute:

$$\begin{aligned} D_8 &= \psi(D_3 \oplus RK_0, D_4 \oplus RK_1, D_5 \oplus RK_2, D_1 \oplus RK_3) \\ D_9 &= \psi(D_2 \oplus RK_4 \oplus const, D_7 \oplus RK_5 \oplus \ell_i, D_6 \oplus RK_7, D_0 \oplus RK_6) \\ D_j &= \psi(D_{j-10}, D_{j-8}, D_{j-3}, D_{j-2}) \qquad j \geq 10 \end{aligned}$$

where ψ is $V = \psi(W, X, Y, Z)$ given by:

$$\begin{aligned} V &:= (Y + (Z \ll 32)) \oplus W \oplus X \oplus (Z \gg 32) \\ V &:= V + (V \ll 32) + (V \ll 43) \\ V &:= V \oplus (V \gg 39) \end{aligned}$$

Round keys

Rnd 1:	D_0, LK_0, D_1, LK_0	Rnd 2:	D_2, LK_1, D_3, LK_1
Rnd 3:	D_4, LK_2, D_5, LK_2	Rnd 4:	D_6, LK_3, D_7, LK_3
Rnd 5:	D_8, LK_4, D_9, LK_4	Rnd 6:	$D_{10}, LK_5, D_{11}, LK_5$
Rnd 7:	$D_{12}, LK_6, D_{13}, LK_6$	Rnd 8:	$D_{14}, LK_7, D_{15}, LK_7$
Rnd 9:	$RK_0, D_{16}, RK_1, D_{16}$	Rnd 10:	$RK_2, D_{17}, RK_3, D_{17}$
Rnd 11:	$RK_4, D_{18}, RK_5, D_{18}$	Rnd 12:	$RK_6, D_{19}, RK_7, D_{19}$
Rnd 13:	$D_{20}, D_{21}, D_{22}, D_{21}$	Rnd 14:	$D_{23}, D_{24}, D_{25}, D_{24}$
Rnd 15:	$D_{26}, D_{27}, D_{28}, D_{27}$	Rnd 16:	$D_{29}, D_{30}, D_{31}, D_{30}$

Table of Contents

Advantages

- Use of “long pipe" design (reuse of previous block). This ensures that an attacker that finds a collision on the compression function by changing the message must also find a collision for the next iteration.

Advantages

- Use of “long pipe" design (reuse of previous block). This ensures that an attacker that finds a collision on the compression function by changing the message must also find a collision for the next iteration.
- Speed about twice that of SHA2.

Advantages

- Use of “long pipe" design (reuse of previous block). This ensures that an attacker that finds a collision on the compression function by changing the message must also find a collision for the next iteration.
- Speed about twice that of SHA2.
- Speed good on both 32 bit and 64 bit processors.

Advantages

- Use of “long pipe" design (reuse of previous block). This ensures that an attacker that finds a collision on the compression function by changing the message must also find a collision for the next iteration.
- Speed about twice that of SHA2.
- Speed good on both 32 bit and 64 bit processors.
- Does not depend only on the interaction between xor and sum.

Advantages

- Use of “long pipe" design (reuse of previous block). This ensures that an attacker that finds a collision on the compression function by changing the message must also find a collision for the next iteration.
- Speed about twice that of SHA2.
- Speed good on both 32 bit and 64 bit processors.
- Does not depend only on the interaction between xor and sum.
- Iteration based on variations of two of the secure PGV schemes, well studied.

Advantages

- Use of “long pipe" design (reuse of previous block). This ensures that an attacker that finds a collision on the compression function by changing the message must also find a collision for the next iteration.
- Speed about twice that of SHA2.
- Speed good on both 32 bit and 64 bit processors.
- Does not depend only on the interaction between xor and sum.
- Iteration based on variations of two of the secure PGV schemes, well studied.
- No table lookups, only logical and arithmetic operations.

Why should NIST choose TIB3 as one of the 15?

NIST should choose a variety of designs for the 15 (semi?) finalists. Among these there should be some of those that are AES based and some that are not, some that are sponge like and some that are not, some double pipe, some single pipe, many that should be fast. TIB3 is among the fastest candidates both on 32 bits and 64 bits. Some of the fastest candidates rely only on the sum-xor interaction, however TIB3 also uses Sboxes (but no table lookups). It is one of the non AES based, non sponge construction and it is the only long pipe design. The round structure allows that more rounds can be added, as stated in the specifications, if it is considered necessary.